Com SIS

Computer Science and Information Systems

Vol 10, No 4, October 2013

Computer Science and Information Systems

Published by ComSIS Consortium

Special Issue on Advances in Model Driven Engineering, Languages and Agents

Volume 10, Number 4
October 2013

ComSIS is an international journal published by the ComSIS Consortium

ComSIS Consortium:
University of Belgrade:
Faculty of Organizational Science, Belgrade, Serbia
Faculty of Mathematics, Belgrade, Serbia
School of Electrical Engineering, Belgrade, Serbia
Serbian Academy of Science and Art:
Mathematical Institute, Belgrade, Serbia
Union University:
School of Computing, Belgrade, Serbia
University of Novi Sad:
Faculty of Sciences, Novi Sad, Serbia
Faculty of Technical Sciences, Novi Sad, Serbia
Faculty of Economics, Subotica, Serbia
Technical Faculty "Mihajlo Pupin", Zrenjanin, Serbia
University of Montenegro:
Faculty of Economics, Podgorica, Montenegro

ComSIS Editorial Office:
University of Novi Sad, Faculty of Sciences,
Department of Mathematics and Informatics
Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia
Phone: +381 21 458 888; **Fax:** +381 21 6350 458
www.comsis.org; Email: comsis@uns.ac.rs

Volume 10, Number 4, 2013
Novi Sad

Computer Science and Information Systems

Special Issue on Advances in Model Driven Engineering, Languages and Agents

ISSN: 1820-0214

ComSIS Journal is sponsored by:

Ministry of Education, Science and Technological Development of Republic of Serbia - http://www.mpn.gov.rs/

Polskie Towarzystwo Informatyczne - PTI (Polish Information Processing Society) - http://www.pti.gda.pl/

Computer Science and Information Systems

AIMS AND SCOPE

Computer Science and Information Systems (ComSIS) is an international refereed journal, published in Serbia. The objective of ComSIS is to communicate important research and development results in the areas of computer science, software engineering, and information systems.

We publish original papers of lasting value covering both theoretical foundations of computer science and commercial, industrial, or educational aspects that provide new insights into design and implementation of software and information systems. ComSIS also welcomes survey papers that contribute to the understanding of emerging and important fields of computer science. In addition to wide-scope regular issues, ComSIS also includes special issues covering specific topics in all areas of computer science and information systems.

ComSIS publishes invited and regular papers in English. Papers that pass a strict reviewing procedure are accepted for publishing. ComSIS is published semiannually.

Indexing Information

ComSIS is covered or selected for coverage in the following:

- Science Citation Index (also known as SciSearch) and Journal Citation Reports / Science Edition by Thomson Reuters, with 2012 two-year impact factor 0.549,
- Computer Science Bibliography, University of Trier (DBLP),
- EMBASE (Elsevier),
- Scopus (Elsevier),
- Summon (Serials Solutions),
- EBSCO bibliographic databases,
- IET bibliographic database Inspec,
- FIZ Karlsruhe bibliographic database io-port,
- Index of Information Systems Journals (Deakin University, Australia),
- Directory of Open Access Journals (DOAJ),
- Google Scholar,
- Journal Bibliometric Report of the Center for Evaluation in Education and Science (CEON/CEES) in cooperation with the National Library of Serbia, for the Serbian Ministry of Education and Science,
- Serbian Citation Index (SCIndeks),
- doiSerbia.

Information for Contributors

The Editors will be pleased to receive contributions from all parts of the world. An electronic version (MS Word or LaTeX), or three hard-copies of the manuscript written in English, intended for publication and prepared as described in "Manuscript Requirements" (which may be downloaded from http://www.comsis.org), along with a cover letter containing the corresponding author's details should be sent to official Journal e-mail.

Criteria for Acceptance

Criteria for acceptance will be appropriateness to the field of Journal, as described in the Aims and Scope, taking into account the merit of the content and presentation. The number of pages of submitted articles is limited to 25 (using the appropriate Word or LaTeX template).

Manuscripts will be refereed in the manner customary with scientific journals before being accepted for publication.

Copyright and Use Agreement

All authors are requested to sign the "Transfer of Copyright" agreement before the paper may be published. The copyright transfer covers the exclusive rights to reproduce and distribute the paper, including reprints, photographic reproductions, microform, electronic form, or any other reproductions of similar nature and translations. Authors are responsible for obtaining from the copyright holder permission to reproduce the paper or any part of it, for which copyright exists.

Computer Science and Information Systems

Volume 10, Number 4, Special Issue, October 2013

CONTENTS

EDITORIAL

Keeping to our tradition of publishing two additional special issues, apart from two regular ones per year, this special issue is titled *Advances in Model Driven Engineering, Languages and Agents*.

Editors of this issue were inspired by several events they organized during 2012 in the following, somehow closely related domains: Advances in Model Driven Engineering; Programming Languages; Computer Languages, Implementations and Tools; and Multi-Agent Systems and Smart Grid Applications. These events included: (i) Workshop on Model Driven Approaches in System Development (MDASD) and International Workshop on Smart Energy Networks & Multi-Agent Systems (SEN-MAS), both organized within the scope of the Federated Conference on Computer Science and Information Systems (FedCSIS) in Wroclaw, Poland; (ii) Symposium on Computer Languages, Implementations and Tools (SCLIT) organized within the scope of the International Conference of Numerical Analysis and Applied Mathematics (ICNAAM) in Kos, Greece; and (iii) Symposium on Languages, Applications and Technologies (SLATE) in Braga, Portugal. After an open call to the prospective authors to submit their papers, and a rigorous reviewing procedure, the same as for regularly submitted papers, we finally accepted 14 papers presenting both theoretical and practical contributions in the field of *Advances in Model Driven Engineering, Languages and Agents*.

In the first paper, *Requirements-Level Language and Tools for Capturing Software System Essence*, Wiktor Nowakowski, Michał Smiałek, Albert Ambroziewicz, and Tomasz Straszak propose a model-based language for comprehensive treatment of domain knowledge, expressed through constrained natural language phrases that are grouped by nouns and include verbs, adjectives and prepositions. They also present an advanced tooling framework to capture application logic specifications making them available for automated transformations down to code. The tools were validated through a controlled experiment.

Sebla Demirkol, Moharram Challenger, Sinem Getir, Tomaž Kosar, Geylani Kardas, and Marjan Mernik in their paper *A DSL for the Development of Software Agents working within a Semantic Web Environment*, introduce a new DSL for Semantic Web enabled Multi-agent Systems. This new DSL is called Semantic web Enabled Agent Language (SEA_L). Both the SEA_L user-aspects and the way of implementing SEA_L are discussed in the paper. The practical use of SEA_L is also demonstrated using a case study which considers the modeling of a multi-agent based e-barter system.

Igor Rožanc and Boštjan Slivnik in their paper *Using Reverse Engineering to Construct the Platform Independent Model of a Web Application for Student Information Systems* present a methodology for extracting the domain knowledge from an existing three-tier web application and subsequent formulation of the platform independent model (PIM). As the paper is primarily aimed at practitioners, a case study illustrating the application of the presented method is also included.

Verislav Djukić, Ivan Luković, Aleksandar Popović, and Vladimir Ivančević, in the paper *Model Execution: An Approach based on extending Domain-Specific Modeling with Action Reports* present an approach to development and application of domain-specific modeling (DSM) tools in the model-based management of business processes. The level of Model-to-Text transformations in a typical DSM architecture is extended with action reports, which allow synchronization between models, generated code, and target interpreters. The applicability of action reports is demonstrated by examples from document engineering, and measurement and control systems.

In their paper *Possible Realizations of Multiplicity Constraints*, Zdeněk Rybola and Karel Richta summarize the process of the transformation of a binary association from a PIM into a PSM for relational databases. They suggest several possible realizations of the source class optionality constraint to encourage the automatically transformation and discuss their advantages and disadvantages. They also provide experimental comparison of the proposed realizations to the common realization where this constraint is omitted.

In their paper *Testing framework for embedded languages*, Dániel Leskó and Máté Tejfel describe a new advantage of embedding a new programming language into an existing one for purpose of software testing. Idea is to introduce a tool support for embedded languages by reusing existing tools for original languages and extend them with the interface to embedded language. Facing with non-extensibility of existing tools authors provide extendable and modular model of a testing framework. Main characteristics the framework ere: straightforward creation, test data generation, addressing the oracle problem, and the customizability of the whole testing phase.

Hemang Mehta, S J Balaji, and Dharanipragada Janakiram in their paper *Extending Programming Language to Support Object Orientation in Legacy Systems* propose an extension of a programming language such is C++ to support object orientation in legacy systems instead of completely redesigning them. They report major issues in providing the compile and runtime support for C++ in legacy systems, and provide a solution to these issues. This is demonstrated on a case study of Linux kernel. Authors provide a technique for converting a large C based software into C++ and experimentally test the results of the approach.

Jakub Křoustek and Dušan Kolář propose in the paper *Context Parsing (Not Only) of the Object-File-Format Description Language* a formal language that can be used for object file formats (OFF) description. They also present the design of a context parser for this language based on formal model. They highlight an ability to describe context-sensitive properties on the level of the language itself as important advantage of this approach. Furthermore they propose a possible usage in existing project.

“Infobots” are small-scale natural language question answering systems drawing inspiration from ELIZA-type systems. Their key distinguishing feature is the extraction of meaning from users’ queries without the use of syntactic or semantic representations. Peter Hancox and Nikolaos Polatidis in their paper *An evaluation of keyword, string similarity and very shallow syntactic matching for a university admissions processing infobot* analyze three approaches to identifying the users’ intended meanings: keyword based systems, Jaro-based string similarity algorithms and matching based on very shallow syntactic analysis. These were measured against a corpus of queries contributed by users of a WWW-hosted infobot for responding to questions about applications to MSc courses.

José Paulo Leal in the paper *Using proximity to compute semantic relatedness in RDF graphs* presents an approach for computing the semantic relatedness of terns in RDF graphs based on the notion of proximity. It is proposed a formal definition of proximity in terms of the set paths connecting two concept nodes, and an algorithm for finding this set and computing proximity with a given error margin.

In the paper *Manage experiments on cognitive processes in writing with HandSpy*, Carlos Monteiro and José Paulo Leal present a development of HandSpy, a collaborative environment for managing experiments in the cognitive processes in writing. The environment was designed to cover all the stages of the experiment, from the definition of tasks to be performed by participants, to the synthesis of results. Despite being a system independent from a specific collecting device, for the system validation, a framework for data collection was created.

In their previous work, the authors of the paper *Batched Evaluation of Linear Tabled Logic Programs*, Miguel Areias and Ricardo Rocha have developed a framework, on top of the Yap Prolog system, that supports the combination of different linear tabling strategies for local scheduling. In this paper, they propose an extension of their framework to support batched scheduling. In particular, they consider the two most successful linear tabling strategies, the DRA and DRE strategies. Their experimental results show that the combination of the DRA and DRE strategies can effectively reduce the execution time for batched evaluation.

Gregor Rohbogner, Ulf J. J. Hahnel, Pascal Benoit, and Simon Fey in the paper *Multi-Agent Systems' Asset for Smart Grid Applications*, recognize that although multi-agent systems are being increasingly employed within smart grid environments, there is a lack of practical understanding of the term "agent" in these scenarios. The authors first discuss why agents are much more than just controllers, optimizers, or learning systems, and then take a critical stance towards existing approaches that employ "multi-agent systems" in smart grids. Finally, they show that, if understood and applied correctly, agents can add significant value to distributed dynamic environments, such as smart grids.

In the paper *Model-based Integration of Constrained Search Spaces into Distributed Planning of Active Power Provision*, Jörg Bremer and Michael Sonnenschein deal with the electricity sector's need for new approaches regarding distributed planning, control and optimization of energy sources within smart grids. The core issue is that the grids are often decentralized and consist of large numbers of individually configured devices. Their proposed solution combines two new methodologies. Support vector-based black-box models are used for handling constraints in distributed optimization scenarios. Then, a distributed greedy approach is employed in order to find an optimal partition of the requested schedule for different distributed energy resources.

On behalf of the ComSIS Consortium and Editorial Board, let us express our great thanks to the reviewers and all the authors for their high-quality work and extraordinary enthusiasm.

Ivan Luković,
Alberto Simões,
Zoran Budimac, and
Mirjana Ivanović,
Editors of the special issue

DOI: 10.2298/CSIS121210062N

Requirements-Level Language and Tools for Capturing Software System Essence

Wiktor Nowakowski[1], Michał Śmiałek[1], Albert Ambroziewicz[12], and Tomasz Straszak[1]

[1] Warsaw University of Technology
pl. Politechniki 1, 00-661 Warsaw, Poland
{nowakoww, smialek, ambrozia, straszat}@iem.pw.edu.pl
[2] Infovide-Matrix S.A.
ul. Gottlieba Daimlera 2, 02-460 Warsaw, Poland

Abstract. Creation of an unambiguous requirements specification with precise domain vocabulary is crucial for capturing the essence of any software system, either when developing a new system or when recovering knowledge from a legacy one. Software specifications usually maintain noun notions and include them in central vocabularies. Verb or adjective phrases are easily forgotten and their definitions buried inside imprecise paragraphs of text. This paper proposes a model-based language for comprehensive treatment of domain knowledge, expressed through constrained natural language phrases that are grouped by nouns and include verbs, adjectives and prepositions. In this language, vocabularies can be formulated to describe behavioural characteristics of a given problem domain. What is important, these characteristics can be linked from within other specifications similarly to a wiki. The application logic can be formulated through sequences of imperative subject-predicate sentences containing only links to the phrases in the vocabulary. The paper presents an advanced tooling framework to capture application logic specifications making them available for automated transformations down to code. The tools were validated through a controlled experiment.

Keywords: requirements engineering, use cases, domain engineering, model-driven software development, model transformation, application logic, metamodel, formal languages.

1. Introduction and Related Work

As pointed out by Brooks back in the eighties [6], software systems possess essential (inherent) and accidental (technological) complexity. The essential complexity cannot be removed without reducing the problem at hand. In order to understand any software system we thus need to "extract" this essential complexity and make it clearly visible. This is especially important when modernising the existing systems. We normally would like to remove all the code, related to the old technology and retain just the problem-related essence. Then, we could transfer this essence (after possible improvement and extension) into a new technology.

An important attempt to enable capturing essential knowledge about software systems is the Knowledge Discovery Metamodel (KDM), as explained by Pérez-Castillo et al. [29]. Unfortunately, KDM operates mainly at quite low levels of abstraction, concentrating e.g. on defining a metamodel for abstract syntax trees capturing the code structure of the system. It also contains structures to represent conceptual-level artifacts but this part of the standard is very roughly defined. Moreover, it can be argued that capturing the detailed internal structure does not reduce the accidental complexity associated with the "twisted" internals of a legacy system. We need means to capture the essence of the system's logic and not e.g. the detailed code breakdown structure as implemented in the legacy system.

An innovative method for improving software application comprehension in order to simplify its maintenance was proposed by Vagač and Kollár in [38] and [24]. In this approach a legacy system, composed of well-known classes and standard libraries, is analysed and a metamodel for the selected features representing functional aspects of the system is automatically created. This provides feature-specific visualization which is closer to the application domain level than to implementation level. The main difficulty in this approach is associated with the construction of a knowledge base – for each recognized feature there must be aspects defined to trace feature implementation and algorithms to model traced implementation details in metamodel.

A very comprehensive approach to capturing essential knowledge (Domain Driven Development - DDD) was proposed by Evans [10]. He postulates organising software development around rigorously defined domain models. These models capture the domain logic of the system at a high level of abstraction. At the same time, the domain logic is the foundational basis to specify the application logic describing the observable interaction of the users with the system (called "workflow logic" by Fowler [13]). This approach was even strengthened in rigour by Bjôrner [4] who advocates mathematical precision in domain engineering. He identifies serious flaws in system specification whenever domain specifications are treated without enough care.

Domain engineering is thus argued as an important element in capturing the essential complexity. Unfortunately, it is normally treated as a second-class citizen in specifying systems. It is equated with a more-or-less complete list of noun-related domain elements with their definitions, placed somewhere close to the end of the overall specification (be it requirements, design or business description) and soon forgotten. Worse still, in many cases the vocabulary is in fact buried in text throughout the whole specification. All the definitions of domain notions are scattered everywhere leading in many places to contradictions (e.g. different definitions of the same term). This all calls for a tooling framework where the various domain notions could be used consistently through referring to a central vocabulary, as postulated by Śmiałek et al. [36].

The tooling for DDD has been developed in the context of the Romulus project (see work by Iglesias et al. [16]). However, the domain models in Romulus are at the level of design models rather than pure domain descriptions. A domain-driven approach was also taken by the creators of the Requirements Specification Language (RSL, see section 2 for an overview of the language basic constructs) within the ReDSeeDS project (www.redseeds.eu). The domain models in this language rely heavily on verbs used within requirements specifications. This is also similar to knowledge engineering approaches like the one described by Chan [7] and also pure ontology languages like RDF [1]. In effect, we result with a constrained language with embedded semantics, capable of representing domains along the proposition by Evermann and Wand [11]. Moreover, the language introduces a very strict relation between the domain logic (expressed through verbs associated with nouns) and the application logic.

In the current work we use RSL to enable capturing the essential complexity at the level of application logic of either existing or new systems (see section 2 for more details on this subject). This kind of "essential complexity" is meant as sequences of user-system, system-system and system-user interactions defining the observable system behavior. We propose to capture it through constrained-natural-language sentences that refer (hyperlink) directly to a domain model based on nouns, verbs and other parts of speech. Similar usage of hyperlinks was proposed by Kaindl [20], but such a comprehensive treatment with an extensive tooling environment is not found in the literature according to our best knowledge. What is more, we propose a method for capturing and migrating the essence from legacy systems. It is unique in generating application logic scripts from UI/GUI-ripping results. The users record their activity in the legacy system and this is transferred to the application logic (essential) specification. Due to precision of such specifications, this can be brought to the level of code in an MDA-style transformation process [22].

This paper constitutes a significant extension to a paper published at the Model-Driven Approaches in System Development workshop at the FedCSIS conference [35]. It provides details on the slightly improved RSL metamodel and gives more examples. It also presents an advanced version of the tools both for recovery and transformation of application logic to code. There are also presented in detail the results of a controlled experiment to validate the presented approach and tools.

2. Basic RSL Constructs for Specifying System Essence

The Requirements Specification Language (RSL) is a formal language for specifying software requirements. An important idea in the RSL approach is separation of concerns in regard to descriptions of the system's behaviour and descriptions of the system's domain. The behaviour in RSL is specified through use cases and their textual scenarios consisting of sentences in constrained

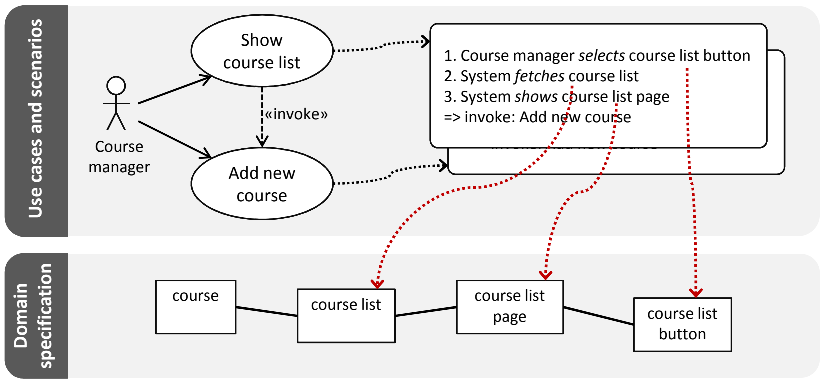


Fig. 1. Example RSL specification – use case scenarios linking to a domain vocabulary

natural language. Words and phrases used in scenario sentences are linked to elements of a separate domain model, as presented in Figure 1.

Such notation, with a centrally defined vocabulary, is easily understandable by different audiences – analysts, developers, architects and end-users. The aim is to facilitate communication during the software development process. The main focus of this communication is usually the outlining of the application logic. The application logic of an IT system defines sequences of interactions between the user and the system in relation to the domain logic within which this system operates. That is the exact information that is captured at the level of requirements through the use of RSL (more on capturing the application logic can be found in section 2.6).

In addition to being human-readable, the RSL notation is also very precise. All the language constructs are defined in a formal way through a grammar expressed as a MOF [27] metamodel. This allows automatic processing of specifications written in RSL (like, for example, MDA-style transformations [26]).

In sections below we describe basic RSL constructs in a bottom-up manner. Due to the extensiveness of the language, the description is limited only to the constructs that are used directly for capturing the software "essence" at the level of application logic. For the extended overview of the RSL language please refer to [34] and to [19] for the complete formal language definition.

2.1. Phrases – Basic Building Blocks for Specifications

In order to describe a domain, people normally use certain natural language phrases. Any entity in a given domain is expressed through a phrase containing prominently a noun. In sentences, nouns are normally used in the role of subjects or objects. Noun phrases are obviously not satisfactory to express the domain logic – its dynamics. We need verbs that can be composed of many words (e.g. phrasal verbs or aggregates of the Dixon's primary and secondary type verbs [8]). In a sentence, a verb occurs as a part of its predicate. It is

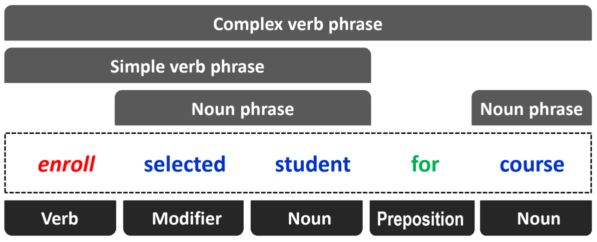


Fig. 2. Phrase structure example

strongly relevant to the noun: it describes behaviours, functions and events of the entity represented by that noun. These are important elements of domain descriptions as defined by Bjôrner [3]. One noun can have any number of behaviours, functions or events associated ("read book", "write book", "buy book"). Sometimes there is a need to enrich nouns with modifiers ("single book", "old book").

To capture the application logic we will thus define a language capable of expressing noun-based phrases. This is illustrated in Figure 2. A noun phrase contains just a noun ("course") possibly preceded by a modifier ("selected student"). A modifier is most often an adjective or an adverb. A simple verb phrase consists of a noun phrase preceded by a verb ("enroll selected student"). A complex verb phrase supplements a simple verb phrase with an additional noun phrase. These phrases are conjoined with a preposition, thus making a complex verb phrase capable of expressing constructs with a direct object and an indirect object ("enroll selected student for course").

The above can be seen as a constrained language and we can define a grammar for it. We want the language to be used for automatic transformations and thus we will use a metamodel to define it (work by Kleppe [23] can be used as a good introduction on this). This is shown in Figure 3.

All phrases are represented by an abstract metaclass Phrase, which has two subtypes: NounPhrase and VerbPhrase. A NounPhrase consists of exactly one NounLink that points to a specific Noun. A NounPhrase can also contain at most one ModifierLink pointing to a Modifier. Such NounPhrases are satisfactory for representing entity names (eg. "course", "selected student"). A VerbPhrase, in turn, describes some behaviour that can be performed in association with an entity represented by a NounPhrase. In the metamodel, VerbPhrase is an abstract subtype of Phrase and it exists in two concrete variants: SimpleVerbPhrase and ComplexVerbPhrase. The SimpleVerbPhrase is the basic structure for expressing the entity behaviour. It contains a NounPhrase in the role of its object (inherited from VerbPhrase), but it also includes a VerbLink pointing to a Verb (eg. "enroll selected student"). A ComplexVerbPhrase describes behavioural relation between two entities. It contains its own (inherited) NounPhrase which plays the role of the indirect object, but also contains a SimpleVerbPhrase possessing another NounPhrase – the direct object (eg. "enroll selected student to course").

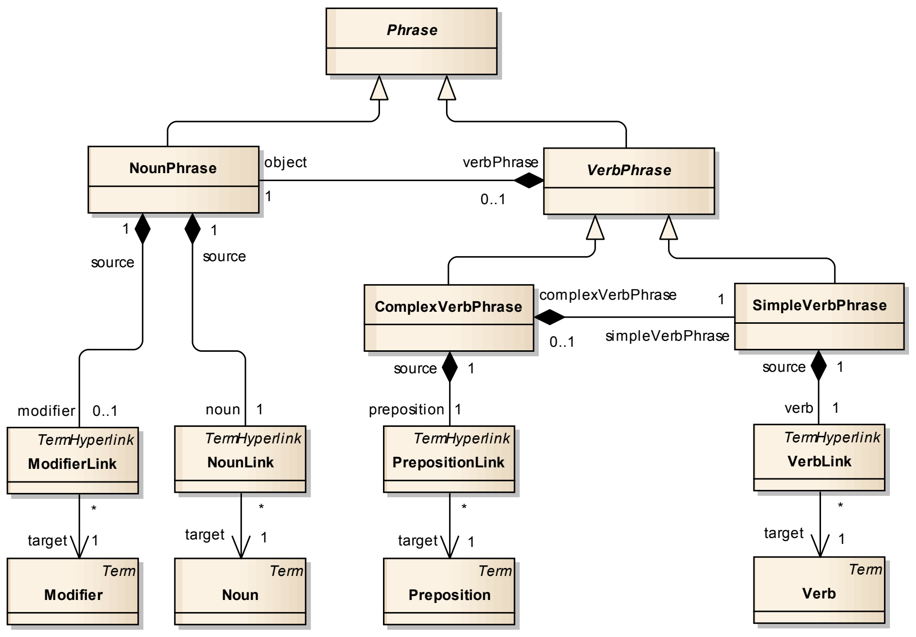


Fig. 3. Phrase metamodel

It is worth noting that the phrases constitute sequences of hyperlinks (subclasses of TermHyperlink) pointing at external terms (subclasses of Term – see Figure 4). These terms (with their forms, which depend on the context) can be stored in an external, global structure (Terminology). This structure defines the semantics of the terms through giving relations between them, and can be based on existing dictionaries/ontologies (e.g. WordNet [12]). This way, the phrases can be subject to semantic-based matching, as described by Wolter et al. [40].

2.2. Domain Specification – Phrases Grouped within Notions

To organise the phrases we will group them by the nouns defining the described domain entities. We will call such group a "notion". The appropriate metamodel for this part of the presented language is shown in Figure 5. Every Notion can include any number of DomainStatements referring to the same noun (eg. "save course", "enroll student for course"). Each DomainStatement contains exactly one Phrase – its name. It can also have a textual description of behavioural features of the related nouns. For example, "validate course" has a different meaning than "save course". The common Noun pointed by all the phrases grouped within the Notion as its statements is used as the name of that notion (see the relevant NounPhrase). Moreover, a notion can contain textual description that defines the notion in the context of the current domain.

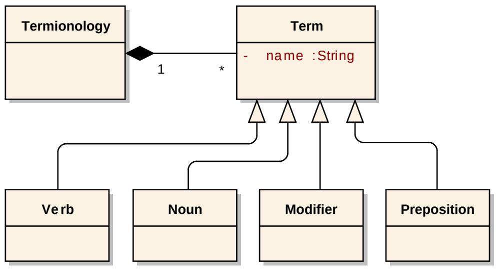


Fig. 4. Terminology metamodel

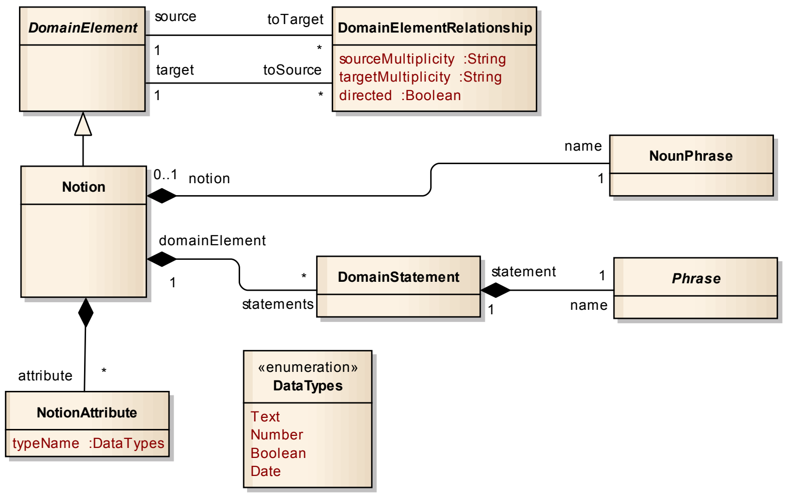


Fig. 5. Notion metamodel

To complete the domain structure, we need to define relationships between notions. This is done through DomainElementRelationships which denote relationships between two DomainElements. Both the source and the target of DomainElementRelationship can have constrained multiplicity described respectively by the sourceMultiplicity and the targetMultiplicity property. The directed property indicates whether a relationship is directed (from source to target) or is bidirectional.

In addition to domain statements and relationships, notions can also have attributes which characterize domain entities. Attributes are represented by NotionAttribute metaclass. The type of an attribute is specified by one of the values from the DataTypes enumeration. For example, the “student” can have such attributes as “name” or “index number” of primitive type Text and Number respectively.

```
Course
Atomic unit of learning and marking for the
[n:students] at the Faculty. The [n:courses] are
taught by [n:academic teachers] to the [n:students].
The [n:courses] result in [n:marks]. Every course can
have different [n:classes].

[v:enroll n:student p:for n:course]
[v:validate n:course]
[v:save n:course]
[v:add n:class p:to n:course]
```

Fig. 6. Example of textual notation for notions

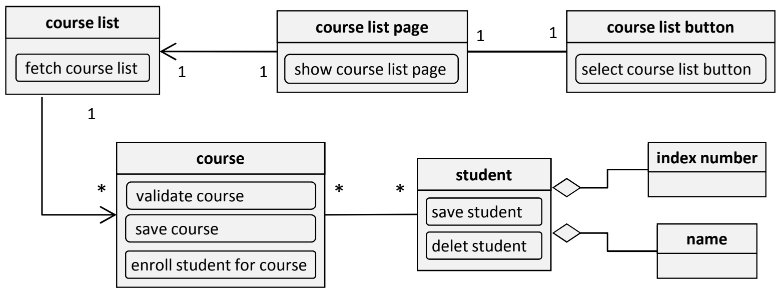


Fig. 7. Example of graphical notation for notions

The above abstract syntax calls for appropriate concrete syntactic elements. Our metamodel introduces a special kind of structural domain representation that explicitly focuses on domain elements. It can be seen as possessing some of the key object-oriented principles: domain elements can be connected through domain element associations adorned with multiplicities. We could thus simply use UML class model notation. However, where a graphical notation is necessary, we propose a notation clearly distinguishing domain elements from classes. This is to stress its domain modelling (cf. ontological modelling) purpose. This is illustrated in Figures 6 and 7. The first Figure presents a textual description of the notion “course” with several phrases. It can be noted that the notion description contains phrases (represented by hyperlinks in the description). In Figure 7 we can see a graphical notation that includes the same notion. The phrases have a notation that clearly distinguishes them from e.g. class operations.

2.3. Hyperlinking Phrases to Build Sentences

The metamodel we have presented allows to organise the domain definition in the form of a dictionary of phrases. We have already shown that the phrases can be hypelinked from within the domain element descriptions (see Fig. 6). However, this can be easily extended to any textual specification. For instance,

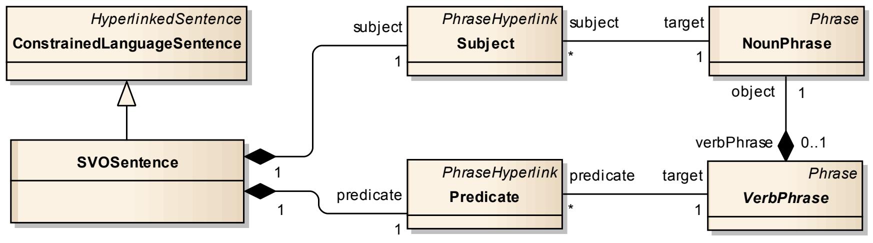


Fig. 8. Constrained language sentence metamodel

we could organise this way the functional requirements. Through consistent use of hyperlinks we could significantly raise precision and unambiguity of such specifications. For this purpose we will thus extend the presented language to allow formulating full sentences constructed out of hyperlinks.

We have already seen that all the elements used in phrases link to Terms in the terminology. In fact, phrases consist only of hyperlinks to specific Terms through the TermHyperlink construct. This idea is extended to use phrases as targets of hypelinking and to construct specifications as sequences of hyperlinks to phrases. Now, instead of copying the same phrase in many places, we just point to its definition placed in a central domain specification. This provides consistency as every hyperlink may point at exactly one element. This is in line with the findings by Kaindl [18] which indicate that hyperlinks applied in requirements specifications are basic facilitators of coherence. However, the approach is beneficial only with strong tool support, which we will discuss in Section 3.

The precision of system specifications is assured by using hyperlinks that link interaction flow descriptions with definitions of phrases. In textual specifications, this leads to the idea of a wiki-like language. Hyperlinks can be inserted into free-form text using the notation presented in the previous section (see Figure 6). They can consist of linked term names preceded by a letter with a colon (":") indicating the term type ("n:" for a noun (NounLink), "m:" for a ModifierLink, "v:" for a VerbLink, "p:" for a PrepositionLink. Each hyperlink text is surrounded by a pair of square brackets.

Unfortunately, free (although hypelinked) text used in specifications has serious limitations. Namely, it is not suitable for automatic processing (e.g. translation into design or code, comparison, structured editing, semantic operations), and it can be formulated still in an unreadable way. To cater for these two problems we would need to introduce much more rigour and limit the language used. We will now present such a limited language with SVO sentences. They will use phrases (or rather: hyperlinks to phrases) as their atomic "lexemes".

The overall structure of an SVO sentence is shown in Figure 8. It consist of a Subject that points to a regular (noun-only) Phrase and a Predicate that points to one of the concrete subtypes of VerbPhrase.

In the simplest case, the Predicate points to a SimpleVerbPhrase. This results in a grammar that follows the Subject – Verb – Object (SVO) scheme, as pro-

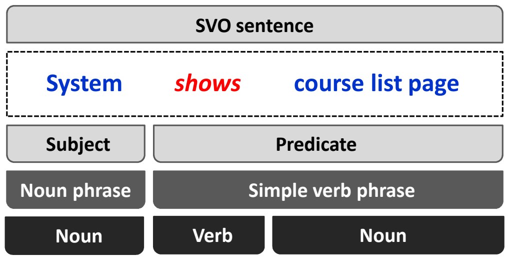


Fig. 9. Example of SVO sentence with simple verb phrase

posed by Graham [15]. An example of such a sentence structure is illustrated in Figure 9. It can be seen that the Predicate of this sentence is a hyperlink to a SimpleVerbPhrase, and the Subject hyperlinks to a NounPhrase. These phrases are further hyperlinked to appropriate terms in the terminology.

In a more complex case, the Predicate points to a ComplexVerbPhrase. In such situation, the sentence is extended by an additional indirect object (SVOO) allowing to express more complex behaviour involving more than one noun phrase (eg. “System adds class to course”).

2.4. Use Case Scenario – Sequence of Sentences

It can be argued that most of the observable behaviour of a software system (its application logic) can be described at the level of requirements with sentences presented in the previous section. For the purpose of defining system’s behaviour, RSL employs use cases as units of system’s functionality. Each use case can be detailed with one or more textual scenarios consisting of sentences in constrained natural language that links to elements of the domain model. RSL defines only one type of relationship between use cases – the ≪invoke≫ relationship. This relationship denotes the situation where scenarios of a use case can be invoked from within another (invoking) use case. A detailed example of notation for use cases and scenarios is shown in Figure 10.

Figure 11 shows a fragment of the RSL metamodel that deals with use case scenarios. Every RSLUseCase contains at least one ConstrainedLanguageScenario. Scenarios, in turn, are composed of ordered set of scenarioSteps forming paths of scenario execution. Every such step is a subtype of an abstract ConstrainedLanguageSentence: an SVOSentence, InvocationSentence or ConditionSentence. The two latter sentences are special types of ControlSentence.

As it was explained above, the predicate of an SVO sentence in a scenario describes an operation that can be performed in association with some entity (eg. “fetch course list”, “save course”). The subject, in turn, indicates who performs the action (eg. “course manager” or “system”).

Every such action can be performed under a certain condition. Condition in a scenario can be expressed with a ConditionSentence. It is a point where the scenario flow is determined: a scenario step that follows the ConditionSentence

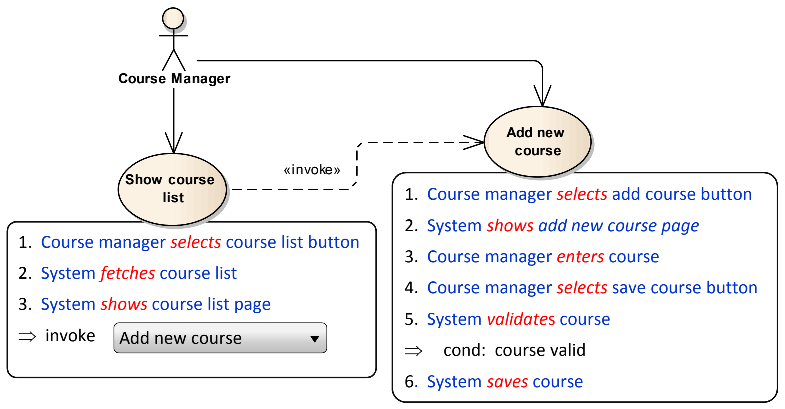

Course Manager
Show course list
Add new course
«invoke»
1. Course manager selects course list button
2. System fetches course list
3. System shows course list page
⇒ invoke Add new course
1. Course manager selects add course button
2. System shows add new course page
3. Course manager enters course
4. Course manager selects save course button
5. System validates course
⇒ cond: course valid
6. System saves course

Fig. 10. Concrete syntax for use case scenarios

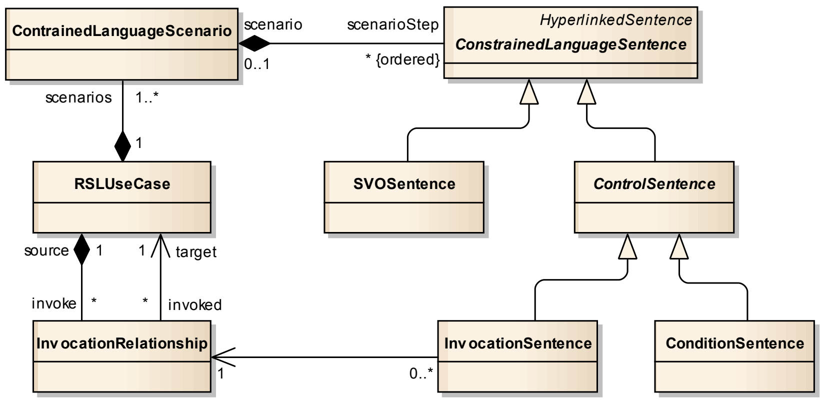


Fig. 11. Use case and scenario metamodel

can be executed only if the condition is met. ConditionSentences always exist in sets of at least two such sentences causing alternative scenarios. The concrete notation for this type of sentence comprises the "cond" keyword followed by a single free-text word, as illustrated in Figure 10.

The ≪invoke≫ relationship has simple abstract syntax reflected through the InvokeRelationship metaclass. What is important, every invocation relationship can have several related InvocationSentences within the invoking use case scenarios. In concrete syntax, such sentences are denoted with the "invoke" keyword, followed by the name of the invoked use case, as illustrated in Figure 10.

The presented simple constructs are satisfactory for capturing even complex application logic expressed through related use case scenarios. By the fact that the RSL grammar is precisely defined through a metamodel, such logic can be

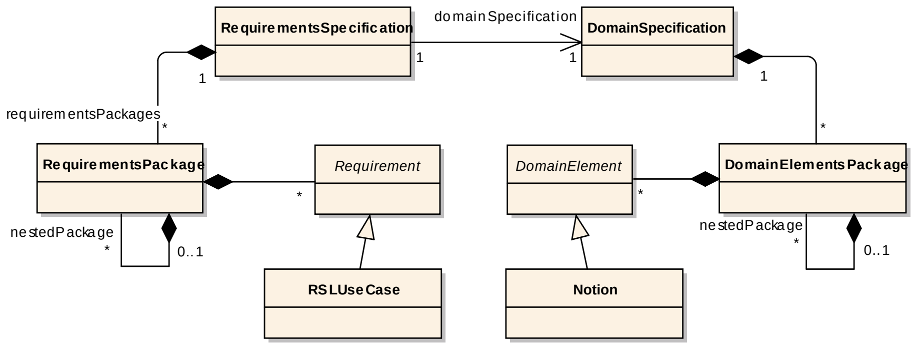


Fig. 12. Requirements specification metamodel

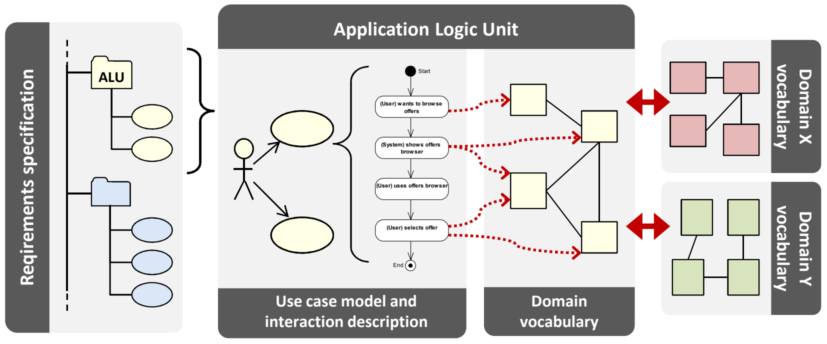

Reqirements specification
ALU
Application Logic Unit
Start
(User) wants to browse offers
(System) shows offers browser
(User) uses offers browser
(User) selects offer
End
Use case model and interaction description
Domain vocabulary
Domain X vocabulary
Domain Y vocabulary

Fig. 13. Levels of application logic management

automatically transformed into design-level models and fully operational code as well.

2.5. Requirements Specification – Container for Requirements and Notions

All the use cases and their scenarios along with linked notions are contained within a requirements specification (see RequirementsSpecification metaclass in Figure 12). It consists of RequirementsPackages that groups Requirements – RSLUseCases in particular. RequirementSpecification also includes a vocabulary of notions used in use case scenarios. These notions are grouped in DomainElementPackages within DomainSpecification. The example structure of requirements specification in the form of a project tree is shown in Figure 14.

2.6. Application Logic Extension to RSL

To fully facilitate creating solution-independent application logic descriptions, core RSL was extended with elements enabling efficient management of the application logic building blocks. This extension (called RSL-AL) builds upon RSL concepts – mainly the separation of system's dynamics description and the domain it pertains to (see Figure 13). This gives the possibility of utilizing patterns of behaviour, to apply the application logic elements at different levels of abstraction: at the level of individual interactions (described as a scenario), at the level of functional units (through use cases) and at the level of application logic units (groups of use cases). The separation between the dynamics and the domain description facilitates using similar interaction flows in different business domains, which leads to defining patterns. As core RSL is sufficient to capture requirements and basic application logic structures, further explanation of RLS-AL concepts is out of scope of this paper; for more details on the subject please refer to Ambroziewicz and Śmiałek [2].

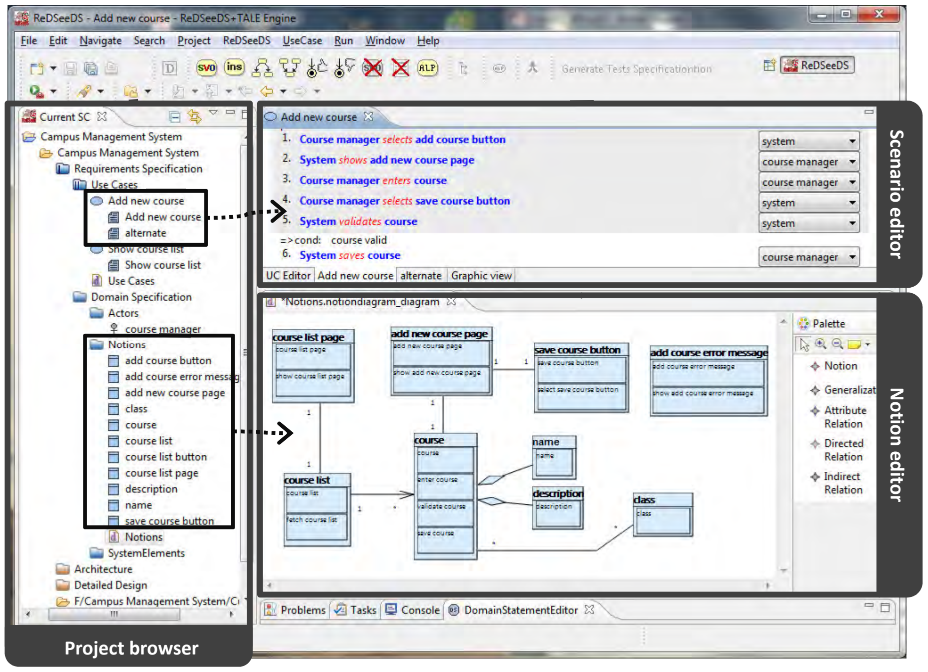
3. Process and Tools

In order to evaluate the presented approach, a tooling framework was constructed. The intent was to enable processing the models according to the notation and metamodel presented in the previous sections. This included support for automatic transformations to design-level artifacts and the process of recovery and migration of the legacy systems to a new architectures. In the paragraphs below we explain the tooling framework based on these objectives.

3.1. Model-driven Development with ReDSeeDS

The central part of the tool chain is the ReDSeeDS tool, that implements the RSL metamodel (see sections 1 and 2). The tool offers a set of editors dedicated to different types of domain elements (see Figure 14, bottom-right). The central point of the tool is the use case scenario editor (as illustrated in Figure 14, top-right). It allows for writing use case scenarios in RSL. The sentences are referencing domain specification elements and marked with colours according to hyperlink types. The tool allows to manage the domain specification elements directly from the use case editor or using a typical tree-like browser (see Figure 14, left).

The process from requirements to code using the ReDSeeDS tool is shown in Figure 15. The first step is to produce the RSL model from the user requirements using the RSL Editor. The second step is to execute a transformation using a transformation engine that produces target models and code. The engine developed within this work uses transformation programs written in MOLA [21] that is a graphical language with an activity-diagram-like notation. Any transformation expressed in MOLA consists of the meta-models for the transformation source and target models, together with one or more transition procedures. The MOLA meta-modelling language is close in its specification to that of EMOF (Essential MOF [27]). MOLA procedures form the executable part of the MOLA transformation.


1. Course manager *selects* add course button — system
2. System *shows* add new course page — course manager
3. Course manager *enters* course — course manager
4. Course manager *selects* save course button — system
5. System *validates* course — system

=>cond: course valid

6. System *saves* course — course manager

Fig. 14. Editors and browsers of the ReDSeeDS tool

The structure and notation of the target model depends on the chosen transformation profile as shown in Figure 16. Currently “RSL to UML” and “RSL to Java” transformation profiles are ready to use and “RSL to SoaML” is planned. The “RSL to UML” transformation profile (see work by Śmiałek [34]) implements the MDA concepts (according to [9]) with the requirements specification as the CIM (Computation Independent Model), 4-layer solution architecture as the PIM (Platform Independent Model) and detailed design based on abstract factory in Java as the PSM (Platform Specific Model) [5]. The target models also contains sequence diagrams describing the behaviour based on use case scenarios. All messages exchanged in sequence diagrams are generated as operations in the corresponding interfaces thus keeping the target model coherent.

The “RSL to Code” transformation generates full structure of the system following the MVP architectural pattern (see [30] for the pattern definition), including complete method contents for the application logic (Presenter) and presentation (View) layers. It also provides a code skeleton for the domain logic (Model) layer. This is presented in Figure 17, that has been generated from the model in Figure 10. According to one of the transformation rules, each use case is transformed into an application logic class. The realisation of this simple rule is the generation of classes CAddNewCourse and CShowCourseList in Figure 17. We can even go further and generate important parts of dynamic code, as it was shown recently by Šimko et al. [39] and Śmiałek [33]. For instance, Figure

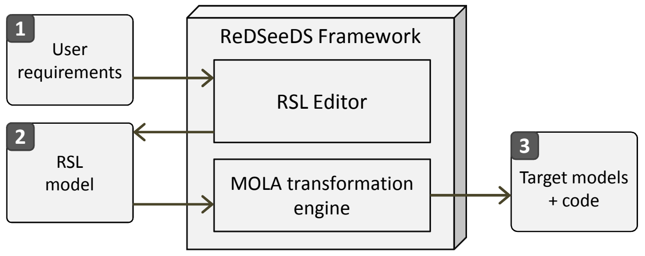


Fig. 15. Model-driven forward engineering with ReDSeeDS

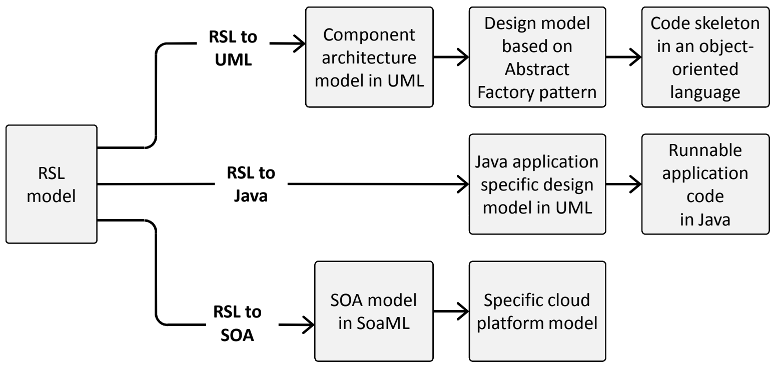

RSL model
RSL to UML
Component architecture model in UML
Design model based on Abstract Factory pattern
Code skeleton in an object-oriented language
RSL to Java
Java application specific design model in UML
Runnable application code in Java
RSL to SOA
SOA model in SoaML
Specific cloud platform model

Fig. 16. Transforming RSL-AL model into different target models

18 presents a small fragment of application logic code generated automatically from the model in Figure 10. As it can be seen, all the “user” sentences (1, 3 and 5) were transformed into operations in the presenter classes. Furthermore, the “system” sentences (2, 4 and 6) were transformed into operation calls to appropriate “view” (denoted by “v”) or “model” (denoted by “m”) objects. The resulting code can be fully operational in regard to the application logic, i.e. it can fully control all the flows of user-system interaction. What is important, the code can also contain decisions (“if” statements) that control the interaction flow depending on the user decisions or the current system state. Such decisions can be generated on the basis of alternative scenarios, but a detailed discussion is out of scope of this paper. A more detailed description of use case scenario translational semantics can be found in [37].

The planned “RSL to SoaML” transformation, similarly to the “RSL to UML” transformation, will implement the MDA concepts. The Service oriented architecture Modeling Language (SoaML) is a new specification from the Object Management Group (OMG) that provides a metamodel and a UML profile supporting different service modelling scenarios [28]: single service description, service-oriented architecture modelling, or service contract definition. Due to the fact that SoaML and UML have the common metamodel, transformations

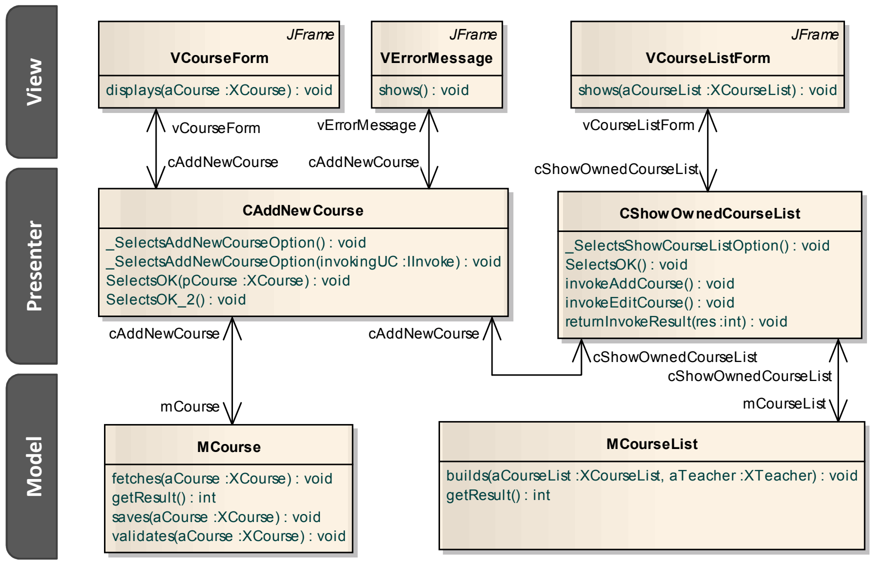

View
Presenter
Model
JFrame
VCourseForm
displays(aCourse :XCourse) : void
JFrame
VErrorMessage
shows() : void
JFrame
VCourseListForm
shows(aCourseList :XCourseList) : void
vCourseForm
cAddNewCourse
vErrorMessage
cAddNewCourse
vCourseListForm
cShowOwnedCourseList
CAddNewCourse
_SelectsAddNewCourseOption() : void
_SelectsAddNewCourseOption(invokingUC :IInvoke) : void
SelectsOK(pCourse :XCourse) : void
SelectsOK_2() : void
CShowOwnedCourseList
_SelectsShowCourseListOption() : void
SelectsOK() : void
invokeAddCourse() : void
invokeEditCourse() : void
returnInvokeResult(res :int) : void
cAddNewCourse
cAddNewCourse
cShowOwnedCourseList
cShowOwnedCourseList
mCourse
mCourseList
MCourse
fetches(aCourse :XCourse) : void
getResult() : int
saves(aCourse :XCourse) : void
validates(aCourse :XCourse) : void
MCourseList
builds(aCourseList :XCourseList, aTeacher :XTeacher) : void
getResult() : int

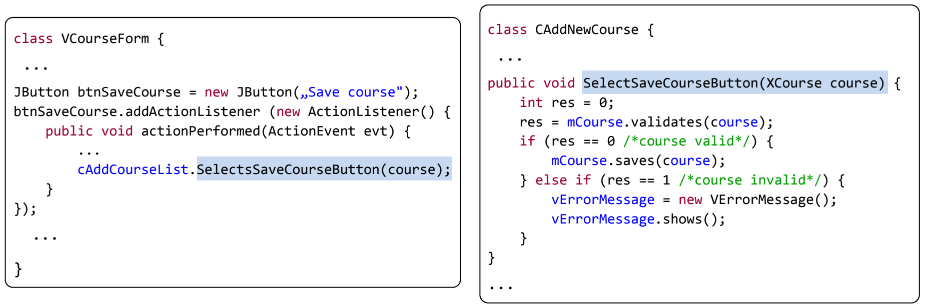
Fig. 17. Fragment of the Java application design model generated with the RSL to Java transformation

into SoaML UML are expected to be similar. The output model of both groups of transformations is an UML-based logical system design at different levels of abstraction, relevant to the structure of the source requirements specification (use cases, notions and packages). The "RSL to SoaML" transformation is expected to generate the structured model of services constructed with stereotyped packages, components, interfaces and classes.

3.2. Recovery and Migration of Legacy System Essence with TALE

The recovery and migration process outline, supported by the tool-chain, is presented in the Figure 19. The main objectives of the process are recovery of the system essence and migration of application logic information from the existing systems, with an intermediate step of storing the application logic information using the RSL metamodel and its RSL-AL extension.

The recovery phase encompasses the idea of semi-automatic reverse engineering while the migration phase is based on model-driven forward engineering techniques described in the previous section. In the process we first analyse the legacy system's UI by using a GUI-ripping tool (see a discussion on this notion by Memon et al. [25]). While performing this step, the GUI-ripping tool records the interactions representing the system's application logic. This includes the user inputs (buttons clicked, data entered, widget focus gained, etc.) and respective system responses (windows displayed, messages shown to the user or even textual console behaviour). An example of such "recorded"



```
class VCourseForm {
 ...
JButton btnSaveCourse = new JButton(„Save course");
btnSaveCourse.addActionListener (new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        ...
        cAddCourseList.SelectsSaveCourseButton(course);
    }
});
  ...
}
```

```
class CAddNewCourse {
 ...
public void SelectSaveCourseButton(XCourse course) {
    int res = 0;
    res = mCourse.validates(course);
    if (res == 0 /*course valid*/) {
        mCourse.saves(course);
    } else if (res == 1 /*course invalid*/) {
        vErrorMessage = new VErrorMessage();
        vErrorMessage.shows();
    }
}
...
```

Fig. 18. Fragment of the code generated automatically from Java application design model

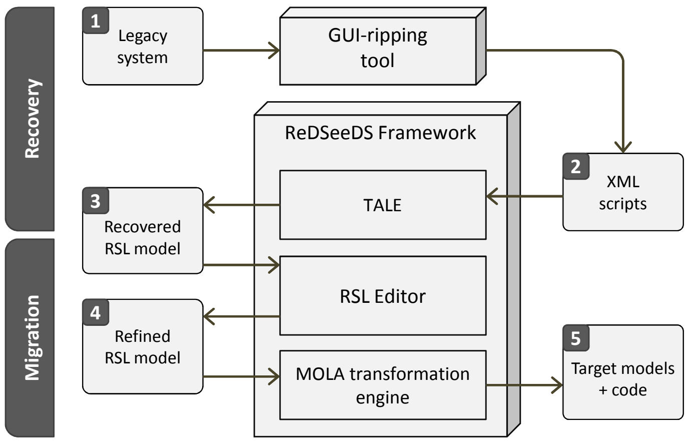
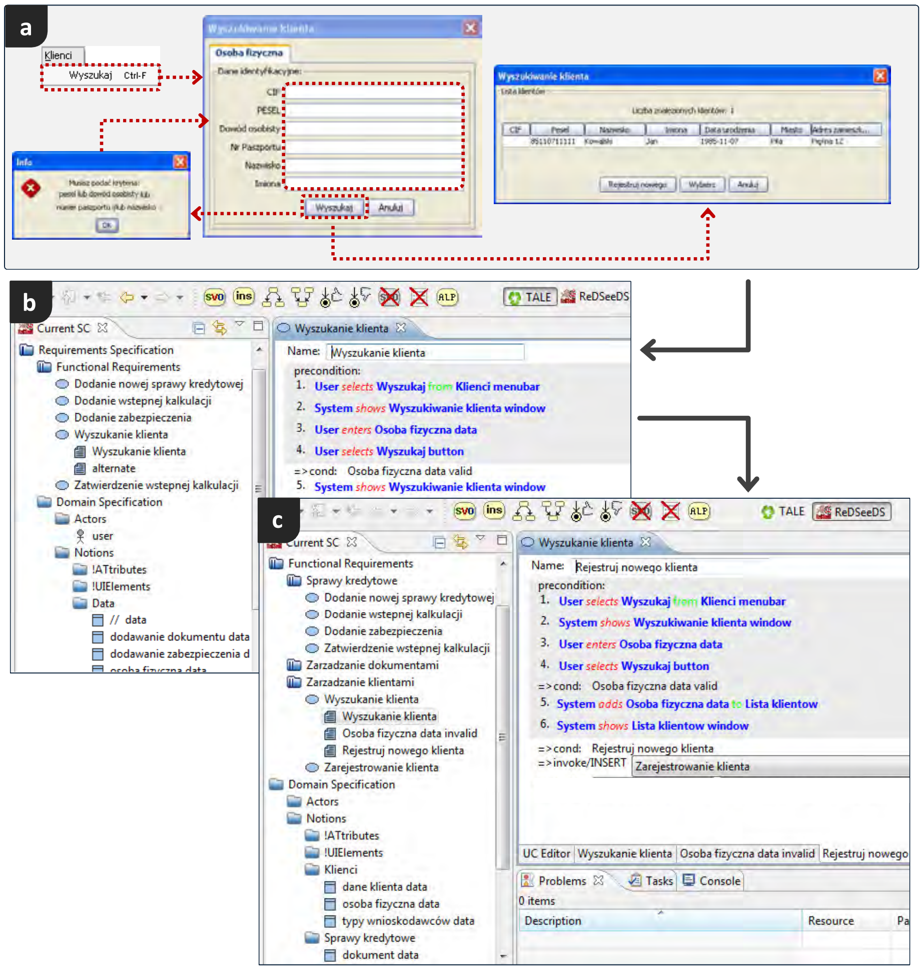


Fig. 19. Overview of the recovery and migration process and tools

interaction is illustrated in Figure 20a. This flow of event concerns functionality of searching a client (in Polish: Wyszukiwanie klienta) in our case study legacy banking system. During this, the GUI-ripping tool records the flows of interaction representing the system's application logic.

In our evaluation, for GUI-ripping we have used a commercial test management tool (Rational Functional Tester, `www.ibm.com/software/awdtools/tester/functional/`). However, any tool allowing for interaction recording to some form of structured text files can be integrated with our software. The tool we used, records sequences of interactions into XML-based scripts (see 2 in the process outline in Figure 19).

The next step of the recovery process is to transform scripts obtained from the GUI-ripping tool into an RSL model (see 3 in Figure 19). This is done with the TALE – Tool for Application Logic Extraction. This novel tool automatically extracts sequences of user-system interactions producing scenarios with SVO



CIF	Pesel	Nazwisko	Imiona	Data urodzenia	Miasto	Adres zamieszk...
	85110711111	Kowalski	Jan	1985-11-07	Piła	Pęgina 12

(b)

Current SC
- Requirements Specification
  - Functional Requirements
    - Dodanie nowej sprawy kredytowej
    - Dodanie wstepnej kalkulacji
    - Dodanie zabezpieczenia
    - Wyszukanie klienta
      - Wyszukanie klienta
      - alternate
    - Zatwierdzenie wstepnej kalkulacji
  - Domain Specification
    - Actors
      - user
    - Notions
      - !ATtributes
      - !UIElements
      - Data
        - // data
        - dodawanie dokumentu data
        - dodawanie zabezpieczenia d
        - osoba fizyczna data

Wyszukanie klienta

Name: Wyszukanie klienta

precondition:
1. User selects Wyszukaj from Klienci menubar
2. System shows Wyszukiwanie klienta window
3. User enters Osoba fizyczna data
4. User selects Wyszukaj button

=>cond: Osoba fizyczna data valid
5. System shows Wyszukiwanie klienta window

(c)

Current SC
- Functional Requirements
  - Sprawy kredytowe
    - Dodanie nowej sprawy kredytowej
    - Dodanie wstepnej kalkulacji
    - Dodanie zabezpieczenia
    - Zatwierdzenie wstepnej kalkulacji
  - Zarzadzanie dokumentami
  - Zarzadzanie klientami
    - Wyszukanie klienta
      - Wyszukanie klienta
      - Osoba fizyczna data invalid
      - Rejestruj nowego klienta
    - Zarejestrowanie klienta
- Domain Specification
  - Actors
  - Notions
    - !ATtributes
    - !UIElements
    - Klienci
      - dane klienta data
      - osoba fizyczna data
      - typy wnioskodawców data
    - Sprawy kredytowe
      - dokument data

Wyszukanie klienta

Name: Rejestruj nowego klienta

precondition:
1. User selects Wyszukaj from Klienci menubar
2. System shows Wyszukiwanie klienta window
3. User enters Osoba fizyczna data
4. User selects Wyszukaj button

=>cond: Osoba fizyczna data valid
5. System adds Osoba fizyczna data to Lista klientow
6. System shows Lista klientow window

=>cond: Rejestruj nowego klienta
=>invoke/INSERT Zarejestrowanie klienta

UC Editor | Wyszukanie klienta | Osoba fizyczna data invalid | Rejestruj nowego

Problems | Tasks | Console

0 items

Description	Resource	Pa

Fig. 20. An example of GUI interaction (a), the automatically recovered RSL-AL model (b) and the manually refined final model (c)

sentences. Figure 20b shows an automatically extracted scenario representing the interaction illustrated in Figure 20a. All the extracted scenarios are attached to use cases, which are grouped within the “Functional Requirements” package forming the recovered model (see the project tree in Figure 20b).

Furthermore, the TALE tool also re-creates the domain vocabulary containing domain notions (created mainly based on data passed to and from the user) and UI elements (windows, buttons, input fields, etc.) used in the recovered interaction description. All this elements are stored in the “Domain Specification” package. The important capability of the tool is ability to extract information about the composition of specific notions. For example, when there is a form displayed to enter personal data (such as first name, last name, PESEL number, etc. – see the “Osoba fizyczna” tab in Figure 20a), a composite notion for “Osoba fizyczna data” is created. Such notion contains attributes for every field filled on the form, instead of a number of unrelated notions coresponding to these fields. This reduces the unnecessary complexity of the recovered model by minimizing the amount of simple notions created from the GUI recordings.

The recovered initial model, thanks to the characteristics of the RSL language, is easily understandable to people (even those barely knowledgeable of the original system internals) thus giving the possibility of its easy extension and modification. This can be made in the ReDSeeDS tool. First of all, some modifications are needed because not all of the application logic information can be automatically retrieved from the recording scripts. This includes sentences that control flow of scenario execution (conditions and ≪invoke≫ sentences) and sentences expressing internal system operations (eg. calls to business logic operations), such as “System verifies data”, “System stores information”, “System deletes item from item list” etc. Also the domain vocabulary usually needs renaming some of the automatically recovered notions. The generated use case specification can also be subject to manual modifications and additions. Changes can be done to cater the migrated system for new or changed functionality or just to optimize some scenario flows, eg. by applying standard application logic patterns [2]. Also, we need to reorganise the model according to the needs of the selected transformation rules. Figure 20c shows the recovered model after refinements.

The refined model (see 4 in the process outline in Figure 19) contains both the still relevant “legacy” specifications and the “new” ones. This constitutes the “essence” of the application logic. We can now use this essence to migrate to a new system design. The migration phase is realised as described in the previous section (as denoted in Figure 19).

4. Evaluation

By using the presented tooling environment several studies are currently undertaken. First, there is performed a larger case study based on a legacy credit management system, used by several banks in Poland (see examples in the previous section). This study is performed in cooperation with Infovide-Matrix S.A. (large Polish software consultancy/provider). The system’s observable application logic has been already recovered into RSL models. The current work focuses on improving existing transformation programs in order to enable migration of the legacy system to the new system architecture fulfilling specific requirements. The current results show very promising levels of application logic that can be recovered from a legacy system. What is important, this recovery is to large extent automatic. Furthermore, the recovered logic is brought to the level of requirements understandable to the users. It was already shown by Jedlitschka et al. [17] that such structured specifications with precisely defined domain vocabularies are well accepted as simply being a better way of expressing requirements. While working within such a “discover notions – write structured sentences” framework, the analysts are encouraged to be acquainted with software system’s environment and are stimulated to write precise, clearly formulated requirements statements.

Further studies, in order to validate the ReDSeeDS model-driven software development approach, were performed with students attending the “Model Driven Software Engineering” course at the Warsaw University of Technology. The students were instructed on RSL constructs and had previously gained knowledge about constructing Model-View-Controler/Presenter style systems, using UML and Java. During the classes, they were formed into 8 groups consisting of 3-4 students each. All the groups were assigned a ready use case model of a Campus Management System, containing 12 use cases with invoke relationships. The first assignment consisted in writing scenarios for the use cases. Four groups wrote the scenarios using the ReDSeeDS tool, while four other groups used a structured use case editor built into Enterprise Architect (EA). The EA editor did not enforce any syntax for the story sentences, although allowed for almost identical structure of scenarios with conditions and notation for alternatives. Moreover, it allowed for hyperlinking of sentence parts to other model elements and the students were asked to introduce links to classes that represented concepts.

The students had 4 hours (2 lab sessions) to write their scenarios and were asked to write them only during the classes. All the groups managed to write good quality scenarios for all the assigned use cases. There were no significant differences between the groups using EA and ReDSeeDS. The groups produced from 121 to 159 scenario sentences (more than 10 sentences per use case) of all types. The average values are illustrated in Figure 21. Based on this, the groups were asked to implement their systems in Java having 10 hours (5 lab sessions). Each scenario sentence was treated as complete if the system managed to pass appropriate data between layers and output “debug” messages. Two of the groups used the RSL to Java transformation, two groups used the standard RSL to UML transformation. The remaining four groups performed manual translation into UML and then code generation within the EA. The first two groups of students managed to implement almost half of the functionality, where on average 68 out of 141 sentences were implemented. It has to be noted that these two groups had extended acceptance criteria where the “debug” messages for the presentation layer were substituted with Swing-style GUI forms. The last four groups of students managed to implement 21 sentences on average. The groups that used the standard RSL to UML transformation performed somewhat better with the average of 28 sentences. A visual comparison is given in Figure 21.

The above simple experiment shows significant improvement in productivity when using fully automatic transformation from RSL to code. However, it needs to be pointed out that it has certain threats to validity. First, the groups could be composed of students with imbalanced qualifications. This was reduced by selecting eight team leaders that performed best during previous classes. These

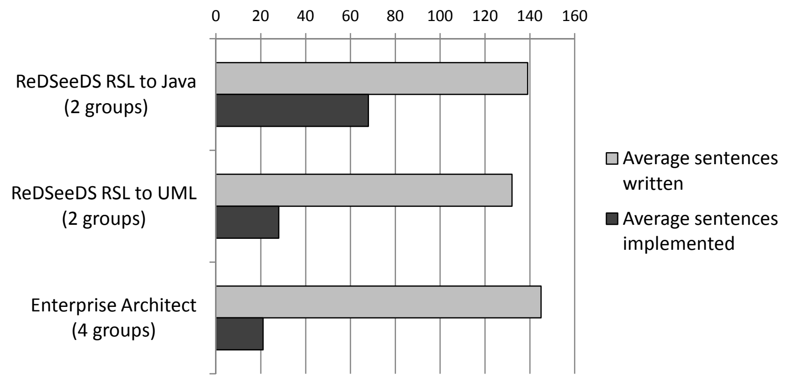


Fig. 21. Student group performance during the evaluation experiment

team leaders chose their group members during a “draft” session thus balancing qualifications between teams. Second, the results could be influenced by lack of necessary proficiency in software design by the groups not using the fully automatic translation. This threat is to some extent reduced by the fact that all the students had previous experience in designing non-trivial three-tier design models during a “Software Design” course. Third, the tooling environment could influence the students’ performance. The EA system was stable and no problems were reported, but the ReDSeeDS system caused some issues due to its prototypical characteristics. In order to assess the last two threats, certain additional (“anecdotal”) information from the students was collected. This confirms that the students from the “EA” and “RSL to UML” groups had problems in designing the systems (or implementing the application logic code within the generated design) by hand and this took most of their implementation time. The automatically generated code gave significant guidance thus improving the performance of the respective groups. The students using ReDSeeDS have reported several problems with using the system, although this did not interfere significantly with their flow of work.

5. Conclusion and Future Work

The presented language aims at capturing the essence of the system’s functionality. It can be noted that the specifications are written at the level of detailed functional requirements. What is important, these requirements are written in near-natural language thus making it accessible to the end-users (see relevant work by Śmiałek [32]). At the same time, specifications are based very coherently on the domain definition by pointing to centrally defined domain statements (phrases). To define the application logic, the specifications can contain only pointers (hyperlinks) to centrally defined noun and verb phrases. A sequence of such hyperlinks forms a scenario describing the user-system interactions. Our experience shows that such application logic scenarios are easy to write by inexperienced developers (analysts) and even the end-users. This can be done using any tool that allows for hyperlink management. This prominently includes wiki systems, but also some CASE tools enable this (see e.g. the scenario editor of Enterprise Architect, `www.sparxsystems.com`).

Writing scenarios hyperlinked to a central vocabulary gives important element of coherence to specifications. However, in order to be able to perform automatic transformations or semantic-based matching [40], we need a tool that implements the presented (or analogous) metamodel. In the current work we have shown that it is also possible to use such a tool as a repository for essential application logic recovered from legacy systems. This repository gives an additional advantage of generating code directly from high-level scenarios. This includes not only the code structure (classes, method signatures) but also the dynamics (method bodies) for the application logic layer.

It can be noted that the presented results can be extended in the direction of creating a more expressive language at the “essential” level. It has to be stressed that the language is not meant for data processing. Thus, it will not possess typical data-processing constructs like loops or variables. Instead, it concentrates on capturing application logic, where loops are implicit through repeated system-user interaction. The currently ongoing work focuses on improving utilization of application logic patterns as proposed by Ambroziewicz [2]. The presented language can be used as a pattern language where the noun and verb phrases can be abstracted from a particular problem domain. The patterns can operate on a generalised domain and then can be instantiated for a specific domain.

Future work will also include extending the TALE tool to be able to recover scenarios combined into use cases on the basis of analysis of GUI-ripping results. It will also consist in extending the language into a language fully capable of performing “programming” at the level of essential application logic. The goal is to move much of such programming activity to a significantly higher level of abstraction than currently. This way, the application logic programming can become accessible even to the end-users. It has to be noted that this language would not yet capture all the essence of a software system functionality. The domain logic will not be expressed in any way. The domain statements would indicate the necessary domain functionality (data processing algorithms etc.), but not define this functionality.

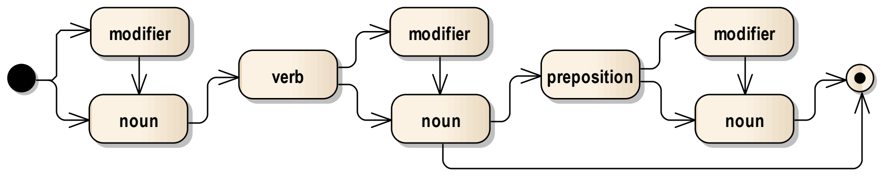


Fig. 22. SVO sentence grammar state machine

Finally, it has to be noted that the SVO grammar is a kind of controlled language with formal grammar as presented in Figure 22 (see also e.g. work by Fuchs et al. [14] or Sleator and Temperley [31]). In this grammar, subclasses of Term metaclass in Figure 3 are terminal symbols. We can thus use a simple analyzer based on a finite state machine to parse SVO sentences.

The grammar as such does have some difficulties with reflecting different natural languages. Some heavily inflected languages, like Polish, need suffixes and prefixes for words, even in sentences with similar structure and meaning. Another problem is that some languages (e.g. German, Turkish) allow for different order of words in a sentence. This can be solved by adding, for example, attributes to sentence classes, indicating word order or language used for this sentence. Nonetheless handling of multi-language specifications is a very interesting challenge for future research and should be investigated further.

Acknowledgments. This research has been carried out in the REMICS project (`http://www.remics.eu`) and partially funded by the EU (ICT-257793 under the 7th Framework Programme).

References

1. Resource Description Framework (RDF), `http://www.w3.org/RDF/`
2. Ambroziewicz, A., Śmiałek, M.: Application logic patterns – reusable elements of user-system interaction. In: Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, vol. 6394, pp. 241–255 (2010)
3. Bjôrner, D.: Software Engineering 3: Domains, Requirements, and Software Design. Texts in Theoretical Computer Science. An EATCS Series, Springer (2006)
4. Bjôrner, D.: Rôle of domain engineering in software development. why current requirements engineering is flawed! Lecture Notes in Computer Science 5947, 2–34 (2010), PSI 2009
5. Bojarski, J., Straszak, T., Ambroziewicz, A., Nowakowski, W.: Transition from precisely defined requirements into draft architecture as an MDA realisation. In: Smiałek, M., Mukasa, K., Nick, M., Falb, J. (eds.) Model Reuse Strategies Workshop, Beijing. pp. 35–42 (2008)
6. Brooks, F.P.: No silver bullet: Essence and accidents of software engineering. IEEE Computer 20(4), 10–19 (April 1987)
7. Chan, C.W.: Knowledge and software modeling using UML. Software and Systems Modeling 3(4), 294–302 (2004)
8. Dixon, R.M.: A new approach to English Grammar, on semantic principles. Oxford University Press (1991)
9. Elvesaeter, B., Berre, A.J., Sadovykh, A.: Specifying services using the service oriented architecture modeling language (SoaML) - a baseline for specification of cloud-based services. In: Leymann, F., Ivanov, I., van Sinderen, M., Shishkov, B. (eds.) CLOSER. pp. 276–285. SciTePress (2011)
10. Evans, E.: Domain Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley (2004)
11. Evermann, J., Wand, Y.: Toward formalizing domain modeling semantics in language syntax. IEEE Transactions on Software Engineering 31(1), 21–37 (January 2005)

12. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press (1998)
13. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
14. Fuchs, N.E., Höfler, S., Kaljurand, K., Rinaldi, F., Schneider, G.: Attempto controlled english: A knowledge representation language readable by humans and machines. Lecture Notes in Computer Science 3564, 213–250 (2005)
15. Graham, I.M.: Task scripts, use cases and scenarios in object-oriented analysis. Object-Oriented Systems 3(3), 123–142 (1996)
16. Iglesias, C.A., Fernández-Villamor, J.I., Pozo, D., Garulli, L., García, B.: Combining domain-driven design and mashups for service development. In: Dustdar, S., Li, F. (eds.) Service Engineering, pp. 171–200. Springer Vienna (2011)
17. Jedlitschka, A., Mukasa, K.S., Weber, S.: Case reuse verification and validation report. Project Deliverable D6.2, ReDSeeDS Project (2009), www.redseeds.eu
18. Kaindl, H.: Using hypertext for semiformal representation in requirements engineering practice. The New Review of Hypermedia and Multimedia 2, 149–173 (1996)
19. Kaindl, H., Śmiałek, M., , Wagner, P., et al.: Requirements specification language definition. Project Deliverable D2.4.2, ReDSeeDS Project (2009), www.redseeds.eu
20. Kaindl, H., Snaprud, M.: Hypertext and structured object representation: A unifying view. In: Proceedings of the Third ACM Conference on Hypertext. pp. 345–358 (1991)
21. Kalnins, A., Barzdins, J., Celms, E.: Model transformation language MOLA. Lecture Notes in Computer Science 3599, 14–28 (2004), MDAFA'04
22. Kleppe, A.G., Warmer, J.B., W, B.: MDA Explained, The Model Driven Architecture: Practice and Promise. Addison-Wesley (2003)
23. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Professional, 1 edn. (2008)
24. Kollár, J., Vagač, M.: Aspect-oriented approach to metamodel abstraction. COMPUTING AND INFORMATICS 31(5), 983–1002 (2012), `http://www.cai.sk/ojs/index.php/cai/article/view/1184`
25. Memon, A.M., Banerjee, I., Nagarajan, A.: GUI ripping: Reverse engineering of graphical user interfaces for testing. In: Proceedings of the 10th Working Conference on Reverse Engineering. pp. 260–269 (2003)
26. Miller, J., Mukerji, J. (eds.): MDA Guide Version 1.0.1, omg/03-06-01. Object Management Group (2003)
27. Object Management Group: Meta Object Facility Core Specification, version 2.0, formal/2006-01-01 (2006)
28. Object Management Group: Service Oriented Architecture Modeling Language (SoaML) Specification, version 1.0, formal/2012-03-01 (2012)
29. Pérez-Castillo, R., de Guzmán, I.G.R., Piattini, M.: Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. Comput. Stand. Interfaces 33(6), 519–532 (2011)
30. Potel, M.: MVP: Model-View-Presenter the taligent programming model for C++ and Java. Taligent Inc (1996)
31. Sleator, D.D.K., Temperley, D.: Parsing english with a link grammar. Tech. Rep. CMU-CS-91-196, Department of Computer Science, Carnegie Mellon University (1991)
32. Śmiałek, M.: Accommodating informality with necessary precision in use case scenarios. Journal of Object Technology 4(6), 59–67 (2005)
33. Śmiałek, M.: Requirements-level programming for rapid software evolution. In: Barzdins, J., Kirikova, M. (eds.) Databases and Information Systems VI, chap. 3, pp. 37–51. IOS Press (2011)

34. Śmiałek, M., Ambroziewicz, A., Bojarski, J., Nowakowski, W., Straszak, T.: Introducing a unified requirements specification language. In: Proc. CEE-SET'2007, Software Engineering in Progress. pp. 172–183. Nakom (2007)
35. Smialek, M., Ambroziewicz, A., Nowakowski, W., Straszak, T., Bojarski, J.: Using structured grammar domain models to capture software system essence. In: FedCSIS. pp. 1349–1356 (2012)
36. Śmiałek, M., Bojarski, J., Nowakowski, W., Straszak, T.: Writing coherent user stories with tool support. Lecture Notes in Computer Science 3556, 247–250 (2005), XP'05
37. Smialek, M., Jarzebowski, N., Nowakowski, W.: Runtime semantics of use case stories. In: VL/HCC. pp. 159–162 (2012)
38. Vagač, M., Kollár, J.: Improving program comprehension by automatic metamodel abstraction. Computer Science and Information Systems 9(1), 235–247 (2012)
39. Šimko, V., Hnětynka, P., Bureš, T.: From textual use-cases to component-based applications. Studies in Computational Intelligence 295, 23–37 (2010)
40. Wolter, K., Śmiałek, M., Hotz, L., Knab, S., Bojarski, J., Nowakowski, W.: Mapping MOF-based requirements representations to ontologies for software reuse. In: CEUR Workshop Proceedings (TWOMDE'09). vol. 531 (2009)

Wiktor Nowakowski is currently pursuing his PhD at the Institute of Theory of Electrical Engineering, Measurement and Information Systems at the Warsaw University of Technology. His main area of research interest is in Requirements Engineering, Model-Driven Software Development, metamodeling and Software Language Engineering. He is also engaged in teaching in these areas. Wiktor has an extensive experience working on small- to large-scale projects in roles covering all stages of the software development life cycle.

Michał Śmiałek is a Professor of Computer Science at the Warsaw University of Technology. He lectures mainly in the area of Model-Driven Software Development both in the academia and for software professionals. Prof. Śmiałek has also over 20 years of experience in software development as a programmer, analyst, process engineer and project manager. He has published over 70 peer reviewed papers and a popular book on UML. Michał Śmiałek leads the SMoG research group that is involved in research and international projects in the area of Model-Driven Requirements Engineering.

Albert Ambroziewicz is professionally engaged in software engineering, mostly in topics related to modeling and metamodeling. He is interested in practical implementations of solutions connected with Model Driven Architecture issues, as well as CASE tools support for industrial usage of UML. Currently he participates in the REMICS project. In the past he took part in several commercial projects, mostly in the fields of enterprise architecture, analysis, R&D and prototyping.

Tomasz Straszak is a researcher interested in software modeling, requirements engineering and test engineering. He is an active member of the Software Modeling Group at the Warsaw Unieversity of Technology. He gained professional experience in telco and banking sectors working as a system/business analyst, software and solution architect and programmer.

Received: December 10, 2012; Accepted: July 3, 2013.

DOI:10.2298/CSIS121105044D

A DSL for the Development of Software Agents working within a Semantic Web Environment

Sebla Demirkol[1], Moharram Challenger[1], Sinem Getir[1], Tomaž Kosar[2], Geylani Kardas[1*], and Marjan Mernik[2]

[1] International Computer Institute, Ege University, Bornova, 35100 Izmir, Turkey
sebla.demirkol@ege.edu.tr, moharram.challenger@mail.ege.edu.tr, sinem.getir@ege.edu.tr, geylani.kardas@ege.edu.tr

[2] Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia
tomaz.kosar@uni-mb.si, marjan.mernik@uni-mb.si

Abstract. Software agents became popular in the development of complex software systems, especially those requiring autonomous and proactive behavior. Agents interact with each other within a Multi-agent System (MAS), in order to perform certain defined tasks in a collaborative and/or selfish manner. However, the autonomous, proactive and interactive structure of MAS causes difficulties when developing such software systems. It is within this context, that the use of a Domain-specific Language (DSL) may support easier and quicker MAS development methodology. The impact of such DSL usage could be clearer when considering the development of MASs, especially those working on new challenging environments like the Semantic Web. Hence, this paper introduces a new DSL for Semantic Web enabled MASs. This new DSL is called Semantic web Enabled Agent Language (SEA_L). Both the SEA_L user-aspects and the way of implementing SEA_L are discussed in the paper. The practical use of SEA_L is also demonstrated using a case study which considers the modeling of a multi-agent based e-barter system. When considering the language implementation, we first discuss the syntax of SEA_L and we show how the specifications of SEA_L can be utilized during the code generation of real MAS implementations. The syntax of SEA_L is supported by textual modeling toolkits developed with Xtext. Code generation for the instance models are supplied with the Xpand tool.

Keywords: Domain-specific Language, Metamodel, Multi-agent System, Semantic Web.

[*] Corresponding author. Tel:+90-232-3113223 Fax: +90-232-3887230

1. Introduction

Software agents [1] [2] are autonomous software components which are able to act on behalf of their users in order to perform a group of defined tasks. Many intelligent software agents interact with each other within a system called Multi-agent System (MAS). Their interactions can be either cooperative or selfish [45]. Software agents and MASs are recognized as both useful abstractions and effective technologies for the modeling and building of complex distributed systems. The implementation of these autonomous, responsive, and proactive systems is naturally a complex task.

Additionally, the Semantic Web improves the World Wide Web such that the web pages' contents can be interpreted using ontologies [46]. Therefore, this new-generation web helps machines to understand web content. It is apparent that the interpretation in question will be realized by autonomous computational entities (i.e. agents) in order to handle the semantic content on behalf of their users. Surely, a Semantic Web environment has specific architectural entities, and thus a different semantics needs to be considered for modeling a MAS within its environment. Thus, the Semantic Web evolution has spawned a new vision regarding agent research. Software agents are planned for collecting Web content from diverse sources, processing the information, and exchanging the results. Autonomous agents can also evaluate semantic data and collaborate with semantically defined entities of the Semantic Web like semantic web services, by using content languages. However, considering agent interactions with Semantic Web elements adds more complexity for designing and implementing these systems.

On the other hand, the Model Driven Development (MDD) is also one of the important software development approaches, moving software development from code to models [43], which increases productivity [26] and reduces development costs [47]. The design and implementation of a MAS may become more complex when new requirements and interactions for new agent environments like Semantic Web are considered. MDD can provide an infrastructure that simplifies the development of such MASs. Being able to work at a higher abstraction level is of critical importance for the development of MASs since it is almost impossible to observe the code level details of the MASs due to their internal complexity, distributedness and openness. Hence, such an MDD application can increase the abstraction level during MAS development. MDD uses different approaches for realizing its goals. One of these methods is Domain-specific Language (DSL) development [8, 14, 29, 32, 48]. DSLs are languages which are comprised of a domain's concepts and terminologies in order to supply the requirements of the domain. A DSL allows end-user programmers (domain experts) to describe the essence of a problem using abstractions related to a domain specific problem space.

We present a new DSL for designing and implementing MASs working within a Semantic Web environment, by motivating from the expressive powers of DSLs and MDD. We call this new DSL as Semantic web Enabled

Agent Language (SEA_L). An abstract syntax and a concrete syntax for SEA_L are discussed in the paper, that originated from the domain-specific metamodel, which is first introduced in [4]. Furthermore, transformations required for code generation from the specifications of SEA_L are defined in order to realize the implementation of modeled MAS in various agent execution platforms.

This paper is an extended version of the paper [6]. It differs from the latter by including a discussion of all viewpoints, the full specification of two crucial viewpoints of the proposed DSL, and a detailed discussion regarding the practical usage of the language within the scope of a case study. The case study covers the design and real implementation of an agent-based e-barter system. Again different from the paper [6], discussion of the agent-based e-barter business domain is elaborated as well as modeling and code generation for agent internals have been added in this paper. Moreover, in this paper the user and implementation aspects of the proposed DSL are discussed separately. Firstly, we present an overview of the SEA_L language, together with a case study. Then the implementation details are stated.

The remainder of the paper is organized as follows: An overview of the new language is given in Section 2 along with an example. The abstract syntax, the textual concrete syntax, and the code generation mechanism for new DSL are discussed in Section 3. In Section 4, the related work is presented. Finally, Section 5 concludes the paper, and states the future work.

2. The SEA_L Domain-Specific Language

In order to separate the 'user' aspects of the SEA_L from its implementation details, in this section we present SEA_L concepts and how to use them, along with a case study and in the next section a discussion on the implementation details of SEA_L.

Since SEA_L is designed for developers of MASs working within the Semantic Web environments, the language’s main concepts consist of both MAS and Semantic Web terminologies.

In a Semantic Web enabled MAS, software agents can gather Web contents from various resources, process the information, exchange the results, and negotiate with other agents. Within the context of these MASs, autonomous agents can evaluate semantic information and work together with semantically defined entities, like Semantic Web Services, using a content language.

SEA_L is divided into eight viewpoints in order to provide clear understanding and efficient usage. These viewpoints are:

1. Agent Internal Viewpoint: This viewpoint is related to the internal structures of semantic web agents (SWA) and defines those entities and their relations required for the construction of agents. It covers both reactive and Belief-Desire-Intention (BDI) [41] agent architectures.

2. Interaction Viewpoint: This aspect of the language expresses the interactions and communications in a MAS by taking messages and message sequences into account.

3. MAS Viewpoint: This viewpoint solely deals with the construction of a MAS as a whole. It includes those main blocks of which the complex system is composed as an organization.

4. Role Viewpoint: This perspective delves into the complex controlling structure of the agents. All role types such as Ontology Mediator Role or Registration Role are modeled in this viewpoint.

5. Environmental Viewpoint: Agents may need to access some resources (e.g. services and knowledge-bases (considering the facts about the surroundings)) within their environments. The usage of resources and the interactions of agents with their surroundings are covered in this viewpoint.

6. Plan Viewpoint: This viewpoint deals particularly with Plans' internal structures. Plans are composed of some Tasks and atomic elements such as Actions.

7. Ontology Viewpoint: SWAs know various ontologies as they work with Semantic Web Services (SWS) and also some ontological concepts which constitute agents' knowledge-bases (such as belief and fact).

8. Agent-SWS Interaction Viewpoint: This is probably the most important viewpoint of SEA_L. The interactions of agents with SWSs are described within this viewpoint. Entities and relations are defined for service discovery, agreement, and execution. The internal structures of SWSs are also modeled.

SemanticWebAgent (SWA) in SEA_L stands for each agent within the Semantic Web enabled MAS. A SemanticWebAgent is an autonomous entity which is capable of interaction with both other agents and *SemanticWebService*s (*SWS*) within the environment. SemanticWebAgents can be associated with more than one *Role* at any time (multiple classifications), and can change roles over time (dynamic classification). An agent can play roles within various environments, have a state (*Agent State*), and own a type (*Agent Type*) during its execution.

A SemanticWebAgent can interact with various services including SemanticWebServices. A SemanticWebService represents a service (except for an agent service), its capabilities, and its interactions, semantically. A SemanticWebService is composed of one or more *Web Service* entities. The corresponding services must have a semantic interface that is going to be used by platforms' agents.

A SemanticWebAgent applies Plans to perform their Tasks. 'Semantic Service Register Plan' (SS_RegisterPlan), 'Semantic Service Finder Plan' (SS_FinderPlan), 'Semantic Service Agreement Plan' (SS_AgreementPlan) and 'Semantic Service Executor Plan' (SS_ExecutorPlan) are extensions of the Plan. Agents use the SS_RegisterPlan for communication with a service registry to discover service capabilities. Other Plans are used to discover SemanticWebServices dynamically, call the services, obtain agreement with them and execute them, respectively. Finally, a SSMatchmakerAgent can play a RegistrationRole to advertise a SemanticWebService.

SEA_L also covers the already expected and traditional MAS entities (in addition to above mentioned items) such as *Capabilities*, *Goal*, *Belief,* and so on. SEA_L also defines various relations for these entities such as *appliesPlan*, *includesBelief*, *usesGoal*, *postCondition*, *realized_by,* and so on. When considering SWSs and their use within MASs, there are entities like *Grounding*, *Process*, *Interface*, *SSMatchmakerAgent*, *RegistrationRole*, and different types of plans. When taking into account the relations regarding agents and SWS interactions, SEA_L contains relations like *appliesPlan*, *playsRole*, *executes*, *uses*, *interactsWith*, *describes*, *presents*, and *supports*. Using these relations, a developer can model a high-level program for MASs working within Semantic Web environments.

2.1. Case Study: E-Barter System

SEA_L can be used in many instances for facilitating the design and development of agent-based systems for various domains such as agent-based business evaluation [30], stock exchange [24], document management [40] and the e-barter system [7]. In order to exhibit the use of the introduced DSL, the modeling of a simple multi-agent based e-barter system is considered during this study. A barter system is an alternative commercial approach where customers meet at a marketplace in order to exchange their goods or services without currency. In barter marketplaces, purchased goods or services are exchanged for manufactured goods or offered services [7].

An agent-based e-barter system consists of agents that the exchange goods or services of owners according to their preferences. In this application, the base scenario is achieved by the *Customer*, '*Barter Manager'* and *Cargo* agents. Interested readers may refer to [7] for a detailed discussion of barter proposals and the tracking of the bargaining process between Customer agents. After the finalization of bargaining, Customer agents send engagement message to the '*Barter Manager*' agent. The '*Barter Manager'* agent notifies the Cargo agent for transporting barter products between Customer agents. This scenario is completed by the acceptances of all participating agents.

For instance, two Customer Agents (one from the automotive industry and another from the healthcare sector) may need to exchange their offered goods and services such that: the car manufacturer offers to sell car spare-parts to a health insurance company (e.g., for the health company’s service cars), and wants to procure health insurance for its employees. Let us consider that the intention of the health insurance company is vice-versa. During bargaining between the agents of the car manufacturer and the health insurance company, our Barter Manager agent uses a semantic web service called '*Barter Service*'. In order to invoke this service, the '*Barter Manager'* first needs to discover the proper semantic web service. Then, it interacts with the candidate service(s) and after an agreement the exact execution of the semantic web service is realized [25]. Figure 1 portrays the partial instance model of an E-Barter system (conforming to the SEA_L's metamodel, as elaborated in Section 3.1).

In the following, we provide a description of the instances and constraint controls for this case study using SEA_L specifications.

Listing 1 shows the textual instance model for the Agent Internal viewpoint of the E-barter system. The instance model includes those variables and relations defined for the E-barter domain. Also, according to the syntax of SEA_L's Agent Internal viewpoint (which is discussed in subsection 3.2), there should be at least one instance of SemanticWebAgent and Capabilities within the system. Therefore, initially, a SemanticWebAgent and Capabilities have been defined for this example.

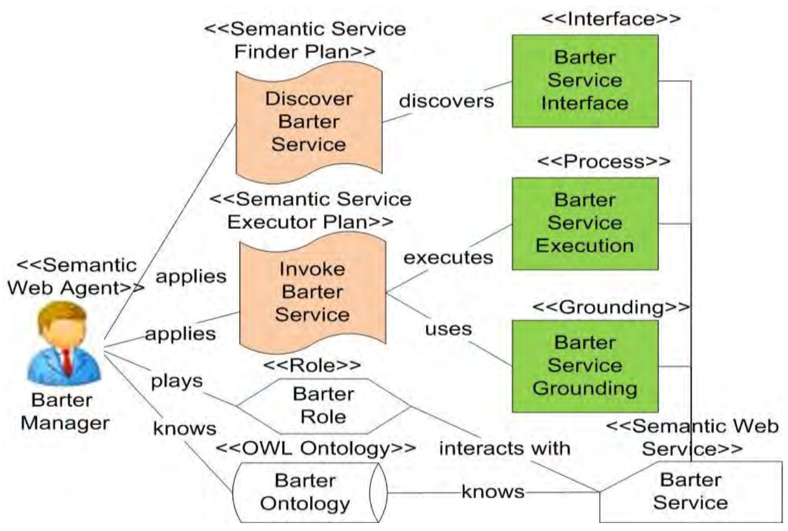


Figure 1. Overview of the E-Barter system as a SEA_L instance [25]

```
AgentInternalViewPoint e_barter {
    SemanticWebAgent barterManager
        "Barter Manager Agent"      // Agent Description
        "Properties"                // Agent Properties
        "Active"                    // Agent State
        "CustomerAgent";            // Agent Type
    Capabilities barterCap
        "Barter Manager Capability";
    Role barterRole;
    Goal bestMatching
        "Doing best matching" 1;     // Recur = 1
    Belief barterKnowledge
        "System facts" 2;           // Dynamic = 2
    Plan financialPlan
        "Cyclic Plan" 1;            // Priority = 1
    barterManager{
        includes barterCap;
        plays barterRole;
    }
    barterCap{
        appliesPlan financialPlan;
        includesBelief barterKnowledge;
        usesGoal bestMatching;
    }
    barterKnowledge{ precondition bestMatching; }
    bestMatching{
        postcondition barterKnowledge;
        realized_by financialPlan;
    }
}
```

Listing 1. Textual modeling for Agent Internal viewpoint of a multi-agent e-barter system in SEA_L

Listing 2 shows the use of SEA_L in textual modeling of Agent-SWS Interaction viewpoint of the multi-agent e-barter system in question. In order to infer about the semantic closeness between offered and purchased items based on the defined ontologies, a SemanticWebAgent is defined which can use a SemanticWebService called barterService.

```
SWSInteractionViewPoint  e_barter_Interaction{
    SemanticWebAgent barterManager
        "Barter Manager Agent"      // Agent Description
        "Properties";                // Agent Properties
    SWS barterService;
    SSMatchmakerAgent barterMatchAgent
        "E-Barter Matchmaker Agent"
        "Properties";
    Grounding barterServiceGrounding;
    Process barterServiceProcess;
    Interface barterServiceInterface;
    SS_RegisterPlan serviceRegistration;
    SS_FinderPlan discoverBarterService;
    SS_AgreementPlan negotiating;
    SS_ExecutorPlan invokeBarterService;
    Role barterRole;
    RegistrationRole matchRole;
    barterManager{
        appliesPlan discoverBarterService;
        appliesPlan negotiating;
        appliesPlan invokeBarterService;
        playsRole barterRole;
    }
    barterMatchAgent{
        appliesPlan serviceRegistration;
        playsRole matchRole;
    }
    invokeBarterService{
        executes barterServiceProcess;
        uses barterServiceGrounding;
    }
    discoverBarterService { interactsWith barterMatchAgent; }
    barterRole { interactswith barterService; }
    barterServiceProcess {describes barterService;}
    barterServiceInterface { presents barterService;}
    barterServiceGrounding { supports barterService;}
}
```

Listing 2. Textual modeling for Agent-SWS Interaction viewpoint of a multi-agent e-barter system in SEA_L

barterManager is an instance of the SemanticWebAgent, which has an important role named barterRole within the system, and applies the discoverBarterService plan, which is an instance of the SS_FinderPlan for finding the desired services. In addition, the agent applies a '*negotiating'* plan, which is an SS_AgreementPlan for negotiating with the discovered services. It also applies the invokeBarterService plan that is an instance of the SS_ExecutorPlan for executing the agreed service. discoverBarterService discovers the barterServiceInterface which presents a barterService. Moreover, invokeBarterService uses barterServiceGrounding for knowing about the execution protocol of the service, and executes barterServiceProcess which declares the internal process of the service.

barterService is an instance of the SemanticWebService, and is described by the barterServiceProcess. This system also has an SS_Matchmaker Agent called the barterMatchAgent, which applies serviceRegistration as an SS_RegistrationPlan for realizing the registration of Interfaces for SemanticWebServices.

In order to provide more readability for Agent-SWS interaction within the code, defining plans, SS_RegisterPlan, SS_FinderPlan, SS_AgreementPlan and SS_ExecutorPlan must be in order, as shown in Listing 2. Otherwise, the SEA_L editor will indicate an error.

As it is restricted in textual concrete syntax, each instance model must have at least one SemanticWebAgent and one SemanticWebService (see Listing 2). After the declarations, the barterManager, being a SemanticWebAgent, applies the *discoverBarterService* plan for finding candidate services, the '*negotiating'* plan for making an agreement with one of them, and the *invokeBarterService* for executing the agreed service. It also plays a *barterRole* for accomplishing these interactions. The *discoverBarterService* plan interacts with the barterMatchAgent and the Matchmaker Agent, in order to find the candidate services. After this interaction, the result is discovering a set of barterServiceInterfaces.

At the end of the SS_FinderPlan, the SS_ExecutorPlan starts which executes the Process and uses Grounding. Moreover, the Role interacts with the SemanticWebService which is presented by the Interface, describes the Process and is supported by the Grounding. Finally, the SemanticWebService depends on at least one '*Service Ontology'*.

As will be elaborated in subsection 3.3 of this paper, by applying the rules written in Xpand [50], the SEA_L's code generation feature enables agent developers to automatically obtain 1) agent software codes conforming to the JADEX [23] BDI platform which is one of the popular APIs for developing software agents, 2) Ontology files in OWL [36] format, and 3) OWL-S [37] representations of the modeled SWSs. Therefore, after running the code generation of SEA_L for the case study, a JADEX ADF file for the barterManager agent and a plan file for each Plan element are generated. The generated ADF file can be used inside a JADEX platform in order to initialize the designed barterManager agent and this agent then executes the generated Java plan code in order to do its tasks. An excerpt from the generated plan named the *financialPlan* for the Barter Manager agent is given in Listing 3. This given code is automatically generated as a result of applying the generation rules (as discussed in section 3.3). Based on the transformation, the modeled agents' behavior is implemented as a JADEX Plan class that owns the *'body'* method to cover the required codes for the agent tasks.

Part of the generated ADF file is shown in Listing 4. In this file, all of the keywords and their attributes correspond with the related tags. For example, the required descriptions for agent capabilities (Lines 14-19), plans to be applied (Lines 20-27), beliefs pertaining to the agent (Lines 28 -32), and the goal of the Barter Manager agent (Lines 33-46) modeled in SEA_L can now be included within a JADEX ADF.

With applying code generations of SEA_L, two ADF files, four plan files, four OWL-S files (Service, Service Process, Service Profile, and Service Grounding), and one WSDL file are generated for SEA_L's Agent-SWS Interaction viewpoint. The ADF and plan files are similar to the ones generated for Agent Internal viewpoint. Therefore, only one part of the generated OWL-S file for '*Barter Service*' is given as an example in Listing 5. Lines from 1 to 9 contain boilerplate text inserted directly from a template (as discussed in subsection 3.3). The barterService, barterServiceInterface, barterServiceProcess and barterServiceGrounding names in lines 24, 27, 30 and 33 of Listing 5 are supplied from the declarations in Listing 2. As previously discussed, a '*Barter Manager*' agent needs a '*Barter Service'* SWS during the bargaining process. Hence, the OWL-S documents referred to in Listing 5 for service interface (in Line 26), service process (Line 29), and finally grounding (Line 32) are used by the agent in order to find, process, and finally invoke the required service.

```
import java.util.*;
import jadex.runtime.*;
import java.util.StringTokenizer;
public class financialPlan extends Plan {
    // Plan attributes.
    ...
    // static block or constructor
    ...
    // Constructor code.
    public financialPlan() {
            ...
    }
    // Plan main code.
    public void body() {
            // Send request
            ...
            // Wait for reply
            …
    }
}
```

Listing 3. Generated plan file for financialPlan

```
<agent
  xmlns = "http://jadex.sourceforge.net/jadex"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://jadex.sourceforge.net/jadex
  http://jadex.sourceforge.net/jadex-2.0.xsd"
  name = "barterManager"
  description = "Barter Manager Agent"
  properties = "Properties"
  package=”jadex.examples.myProjects”
>
  <imports>
       <import>jadex.adapter.fipa.*</import>
  </imports>
  <capabilities>
    <capability>
       name = "barterCap" file=""
       description = "Barter Manager Capability"
    </capability>
  </capabilities>
  <plans>
      <plan name = "financialPlan"
        description = "Cyclic Plan"
        priority="1" />
     <plan name = "discoverBarterService" />
     <plan name = "negotiating" />
     <plan name = "invokeBarterService" />
  </plans>
  <beliefs>
       <belief name="barterKnowledge"
         description="system facts"
         dynamic="1" />
  </beliefs>
  <goals>
    <achievegoal name="bestMatching"
      recur = 1
      exclude = "when_tried"
      recalculate = "true" retry="true"
      exported = "false"
      posttoall = "false"  recurdelay = "0"
      randomselection = "false"
      retrydelay = "0">
      <creationcondition>
            <!-- Write Creation Condition -->
      </creationcondition>
    </achievegoal>
  </goals>
</agent>
```

Listing 4. Part of generated ADF file from Agent Internal viewpoint of barterManager in E-Barter System case study

```
<?xml version="1.0" encoding = ‘ISO-8859-1’?>
<!DOCTYPE ruidef[
   <!ENTITY barterService_profile
        “http://mas.ube.ege.edu.tr/ barterServiceProfile.owl”>
   <!ENTITY barterService_process
        “http://mas.ube.ege.edu.tr/ barterServiceProcess.owl”>
   <!ENTITY barterService_grounding
        “http://mas.ube.ege.edu.tr/ barterServiceGrounding.owl”>
]>
<rdf:RDF xmlns:rdf= "&rdf;#" xmlns:rdfs="&rdfs;#"
   xmlns:owl = "&owl;#" xmlns:service= "&service;#"
   …
   xml:base="&DEFAULT;" >
   <owl:ontology rdf:about="">
   <owl:versionInfo>
         $Id: barterService.owl,v 1.14 2012/10/08 15:27:40 $
   </owl:versionInfo>
       <rdfs:comment> "This ontology represents the OWL-S
           service description for the barterService service example."
       </rdfs:comment>
       <owl:imports rdf:resource="&service;" />
       …
   </owl:Ontology>
   <service:Service rdf:ID= "barterService">
         <!-- Reference to the Profile -->
         <service:presents rdf:resource="&barterService_profile;
             #barterServiceInterface"/>
         <!-- Reference to the Process Model -->
         <service:describedBy rdf:resource="&barterService_process;
             #barterServiceProcess"/>
         <!-- Reference to the Grounding -->
         <service:supports rdf:resource="&barterService_grounding;
             #barterServiceGrounding"/>
   </service:Service>
   <profile:Profile rdf:about=&
          "barterService_profile;#barterServiceInterface">
          <service:presents rdf:resource=#"barterService"/>
   </profile:Profile>
   <process:AtomicProcess rdf:about=&
          "barterService_process;# barterServiceProcess">
           <service:describedBy rdf:resource=#"barterService"/>
    </process:AtomicProcess>
    <grounding:WsdlGrounding rdf:about=&
          "barterService_grounding;# barterServiceGrounding">
          <service:supports rdf:resource=#"barterService"/>
   </grounding:WsdlGrounding>
  </rdf:RDF>
```

Listing 5. Part of generated OWL-S Service file

3. SEA_L Implementation

In this section, the implementation details of SEA_L language are provided including abstract syntax as a metamodel divided into several viewpoints, its textual concrete syntax, and the required code generation for presenting the operational semantics of the language.

3.1. Abstract Syntax

The abstract syntax of a DSL describes the concepts and their relations without any consideration of meaning. In terms of MDD, the abstract syntax is described by a metamodel that defines what the models should look like.

The Platform Independent Metamodel (PIMM) which represents the abstract syntax of SEA_L is divided according to the eight viewpoints which were previously given in section 2.

We discuss the metamodel over its Agent Internal viewpoint as well as Agent-SWS Interaction viewpoint throughout this paper due to the vital importance of these viewpoints. In addition, critical entities from other viewpoints are already considered during the following discussion. The related viewpoints are shown in Figures 2 and 3, respectively. In these Figures, the elements filled-in with light gray come from other viewpoints which are shown on the top or bottom of the element using '<<' and '>>'. In other words, these elements are common elements amongst the viewpoints, and tailor them to each other.

The Agent Internal viewpoint is related to the internal structures of the semantic web agents and defines the entities and their relations required for the construction of agents. A partial metamodel which represents this viewpoint, is given in Figure 2.

SEA_L's metamodel (hence abstract syntax) supports both reactive and Belief-Desire-Intention (BDI) agent architectures. BDI was first proposed by Bratman [3] and is used within many agent systems. In a BDI architecture, an agent decides about which Goals to achieve and how to achieve them. Beliefs represent the information an agent has about its surroundings, while Desires correspond to the things that an agent would like to have achieved. Intentions, which are the deliberative attitudes of agents, include the agent planning mechanism in order to achieve the goals. Taking concrete BDI agent frameworks (such as JADEX [23] and JACK [21]) into consideration, we propose an entity called *Capabilities* which includes each agent's *Goals*, *Plans* and *Beliefs* about the surroundings.

The Agent-SWS Interaction viewpoint focuses on the internal structure of the *SemanticWebServices* and the interaction of any SemanticWebAgent with SemanticWebServices within a MAS organization. Concepts and their relations for appropriate service discovery, agreement with the selected service, and execution of the service are all defined within this viewpoint. A partial metamodel of SEA_L which represents this viewpoint is shown in Figure 3.

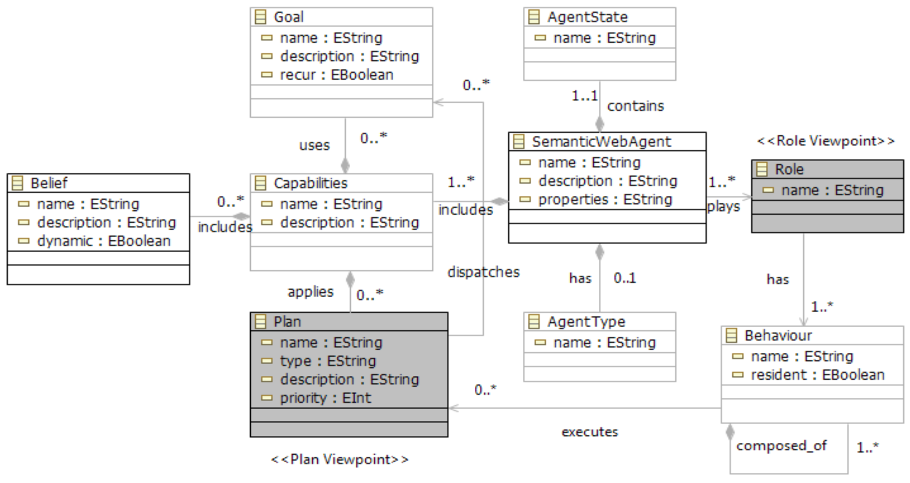


Figure 2. Agent Internal viewpoint

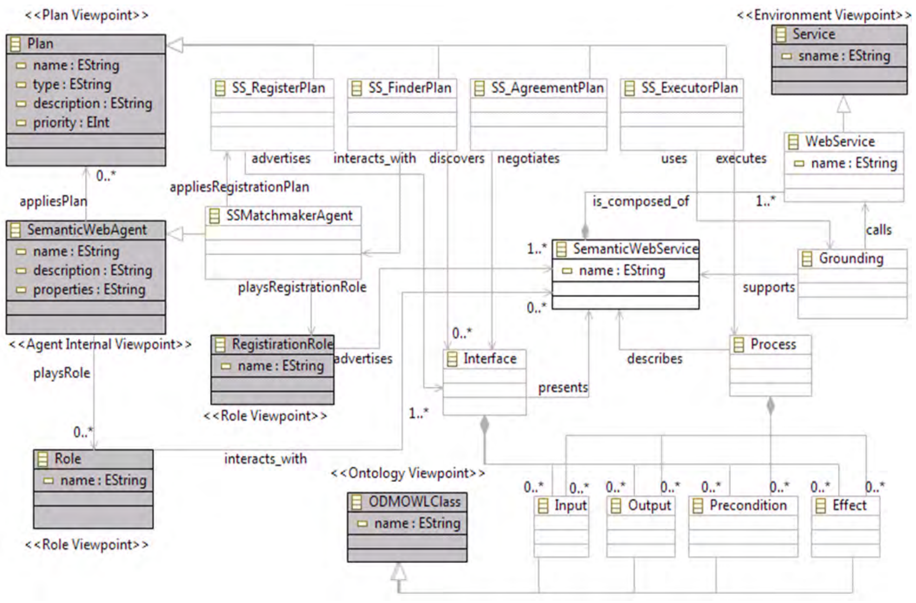


Figure 3. Agent-SWS Interaction viewpoint.

Semantic Web Service (SWS) modeling approaches (i.e. OWL-S [37]) generally define a service with three documents: '*Service Interface*', '*Process*

Model', and '*Physical Grounding'*. '*Service Interface*' is the capability representation of the service in which service inputs, outputs, and any other necessary service descriptions are listed. The '*Process Model*' defines a service's internal combinations and service execution dynamics. Finally, '*Physical Grounding*' defines the service's execution protocol. These meta-entities are shown in Figure 3 with *Interface*, *Process,* and *Grounding* entities, respectively. These components can use *Input*, *Output*, *Precondition,* and *Effect* which are extensions of the Web Ontology Language (OWL [36]) class from Object Management Group's (OMG) Ontology Definition Metamodel (ODM) [35].

On the other hand, agents need to communicate with a service registry element in order to discover service capabilities. Hence, the metamodel includes a specialized agent entity, called the *SSMatchmaker* Agent. This entity represents those matchmaker agents which store the capability advertisements of SemanticWebServices within a MAS, and match those capabilities with service requirements sent by the other platform agents.

When considering the other viewpoints of SEA_L, the MAS viewpoint solely deals with the construction of a MAS as an overall aspect of the metamodel. Plan viewpoint defines a Plan's internal structure. When an Agent applies a Plan, it executes its Tasks. In addition, message transaction is considered within this viewpoint. The Role viewpoint shows distinct types of roles. Agents can use several roles at any time and can alter these roles over time. The Interaction viewpoint focuses on agent communications and interactions in a MAS, and defines entities and relations such as *Interaction*, *Message*, and *MessageSequence*. The Environment viewpoint focuses on the relations between agents and to what they access. Environment contains all non-Agent *Resources*, *Facts,* and *Services*. The Ontology viewpoint brings all ontology sets and ontological concepts together. ODM OWL [36] Ontology from OMG is a standard for all of our ontology sets such as *Role*, *Organization,* and *ServiceOntologies*.

3.2. Textual Concrete Syntax

The textual concrete syntax of SEA_L is provided with Xtext [52]. Xtext is a language development framework for developing textual modeling languages. It can be used for creating a sophisticated Eclipse-based development environment. Xtext is based on EBNF (Extended Backus–Naur Form) [20] rules.

If the metamodel which represents the abstract syntax for SEA_L is considered as an analysis phase of the concrete syntax of SEA_L, the design phase will be the part describing the EBNF rules. One of the main advantages of DSLs is for validating domain-specific constraints. The constraints of the language can be implemented within the '*Validation Package*' in Xtext, which provides a dedicated hook for validation rules. Also, other features of SEA_L's textual concrete syntax are created using both manually-written code and Xtext features. When using Xtext features, the textual concrete syntax supplies auto completion, syntax coloring, rename refactoring, bracket matching, auto edit, an outline view that shows the semantic structure of the model and code formatting for properly indenting the documents. The above discussed constraints of SEA_L's metamodel, are realized by defining the EBNF rules. With these capabilities, the new DSL possesses both the structural and static semantics of the MAS domain. The structure is defined by the method signatures and the static semantics are defined by the constraint code.

During textual modeling with Xtext, the controls over the instance models can be realized via controlling packages. These packages include formatting, scoping, and validation.

The formatting package (Pretty Printing [12]) simply controls and applies the editorial rules for an instance model. In this package, by accessing the language grammar, we can define additional editorial controls (formatting configuration) in order to modify the written program automatically, which help the instance model to be more readable. For example, spaces for keywords, line-wrap rules, etc can be considered in an instance model of the DSL.

Using the scoping application programming interface (API), it is possible to define which elements are referable by a certain variable reference [12]. In other words, it can be controlled that from which parts of the program, a variable in a scope (a block of code), can be accessed.

One of the interesting aspects of developing a DSL is static analysis or validation of the written program. Validation package plays this role within the Xtext tool. The goal is that the users of the language obtain informative feedback as they type the program [12]. Some of the validations are performed automatically, e.g. syntactical and crosslink validations using parser and linker, respectively; although they can also be customized by the user. This type of validation is done with the help of grammar and scoping. However, in addition to the automatic validations, we can specify additional constraints specific for our Ecore model, called custom validation. For example, it is possible to control the number of specific elements. Although some of the constraints could be fulfilled by grammar terminal rules in Xtext (e.g. controlling the format of the defined variables), we implemented them using the validation package to ease providing desired messages (warning or error), and to provide the possibility for fixing the error or warning. In the remainder of this subsection, we discuss how the textual concrete syntax of SEA_L's major viewpoints is provided with Xtext.

3.3. Textual Concrete Syntax of Agent Internal Viewpoint

An Xtext grammar is structured with rules which are identified by the text to the left of a colon. There is at least one rule for each meta-element within the textual concrete syntax. EBNF rules are defined for Agent Internal viewpoint according to the constraints in the metamodel. The first constraint is that all

of the instance model’s elements must be in *AgentInternalViewpoint* tag. Also, the instance model must start and end with curly brackets. An example of another constraint is that each instance model must have at least one SemanticWebAgent and one Capabilities, in any order. These constraints are supplied within AgentInternalViewpoint, rule which is given in Listing 6.
According to Xtext syntax, the assignment operator, '=', denotes a single valued feature, the '+=' operator denotes a multi-valued feature, and the asterisk operator, '*', denotes a cardinality of 0..n. Also, within each rule, referring to predefined parser rules is possible using ‘[’ and ‘]’ characters (called '*cross referencing*'), as shown in Listing 7 Line 3.

```
AgentInternalViewpoint:
  'AgentInternalViewPoint'   '{'
       semanticwebagent+=SemanticWebAgent &
       capabilities+=Capabilities
     …
  '}';
```

Listing 6. A part of AgentInternalViewpoint rule.

```
01Capabilities:
02   'Capabilities'  name = ID description = STRING';' |
03   cap = [Capabilities]  '{'
04       ( 'includes'  belief = [Belief]';' |
05          'uses' goal = [Goal]';' |
06          'applies' plan = [Plan] ';'  )*
07   '}';
```

Listing 7. Capabilities rule

SEA_L’s metamodel conforms to BDI [41] architecture. Therefore, a group of meta-elements exists for supplying the BDI structure. When considering this structure, a Capabilities meta-element consists of Belief, Goal, and Plan meta-elements. The user can define numerous relations by considering the Agent Internal viewpoint. This structure is defined within the Capabilities rule, which is shown in Listing 7. The developer can define the Belief, Goal, and Plan meta-elements as often as needed and in any order, regarding lines 4 to 7 of Listing 7.

The agent state and type definitions are considered as string-terminals within the Agent Internal viewpoint, although they could be implemented as hard-coded enumerations or references to their definitions. This is because we believe that agents can conceptually have any user-defined state and type (not limited to specific states or types). Also, in order to have agent definition integration within a single line, we do not use references to agent type and state definitions.

Fewer constraints are defined within the Agent Internal viewpoint in comparison with the Agent-SWS Interaction viewpoint, since the elements are generally used arbitrarily, and most of the relations are independent within the Agent Internal viewpoint.

The user can assign a keyword to the name of an instance of any meta-element inadvertently. All of the keywords within the textual concrete syntax start with a lower-case letter. Therefore, a prevention mechanism is provided for preventing the users from defining a name starting with a lower-case for names which will not cause inconsistency between keywords and names. Validation Packages of Xtext are overridden for controlling user's variable definition. As illustrated in Listing 8, the editor will show an error if the developer defines a capability name starting with an upper-case. The corresponding code is written in the '*Validation Package*' in Xtext and some extra code is added to this package. These constraint controls are realized within the validation package (instead of grammar terminal rules) for enhancing the provision of customized error and warning messages, and also the possibility of fixing these errors and warnings. Similar controls are provided for other entities like Plan, SemanticWebAgent, SemanticWebService, etc.

```
@Check
public void CapabilitiesStartWithLowerCase(
    Capabilities cap) {
    if ( ! Character.isLowerCase(cap.getName().charAt(0)) ) {
        error("Name must start with lower case",
        AgentInternalDSLPackage. CAPABILITIES__NAME);
    }
}
```

Listing 8. Validation Package code for preventing the definition of an upper-case name within the Semantic Web Agent Internal viewpoint

Additional Xtext features are used to limit the user whilst creating instance models, for example, another control supplied with the Validation Package code which prevents the user entering an empty string to an attribute. The code block in Listing 9 provides an error in the editor, if the user gives an empty string to the *'type'* attribute of a *Behavior*. Within the Xtext validation package, '*@Check*' is a java annotation for defining a validation rule.

```
@Check
public void checkTypeIsNotEmpty (Behavior beh)
{
    if ( beh.getType().isEmpty() ) {
        error("Behavior type is empty",
        AgentInternalDSLPackage.BEHAVIOR__TYPE);
    }
}
```

Listing 9. Validation Package code to prevent defining an empty string

```
public void checkNegativeElement (Plan plan)
{
    If ( plan.getPriority ( ) < 0 )
        error ( "Negative value is not accepted",
              MyDslPackage.PLAN__DESCRIPTION);
}
```

Listing 10. Validation Package code to check the negative values for plan priority

In some part of the language, validity for a variable’s value is examined using an overridden Xtext validation package. For example, as shown in Listing 10, the value of the priority for the plan element is checked, and negative values are not accepted.

3.4. Textual Concrete Syntax of Agent-Semantic Web Service Interaction Viewpoint

When considering Agent-SWS Interaction viewpoint, instances of related meta-elements and their relations must be defined inside a *SWSInteractionViewpoint* code-block as part of the instance model. Similar to the Agent Internal viewpoint, in this viewpoint, the left-hand bracket must be at the beginning of the model and the right-hand bracket at the end of it. Every instance model must have at least one SemanticWebAgent and one SemanticWebService, and every command or declaration must end with a semicolon. Otherwise, an error will occur in the editor. According to Figure 3, a SemanticWebService must have relations with Grounding, Process, and Interface. Each instance model must contain these elements and the relations between them. Part of the Xtext code for supplying these relations is given in Listing 11. Line 4 forces the user to use the *'describes'* relation. Lines 10, 11, and 16 have similar meanings.

Some rules are written in order to provide a specific sequence of code, while another group of rules allows them to be independent of a sequence within the textual instance model, where it is required. For example, Lines 10 and 11 are written to supply the independency within the sequence of relations in Listing 11. The user can define the '*supports SWS*' relation before or after a '*calls WebService*' relation. In addition, the user can define the '*calls WebService*' relation as often as necessary, whereas it is restricted to defining only one '*supports SWS*' relation.

According to the Agent-SWS Interaction viewpoint, each instance model should have at least one SemanticWebAgent and one SemanticWebService supplied with the '*Validation Package*'. Listing 12 shows the implementation of the *checkAtLeastOneSWS* constraint.

In Listing 12, Lines 4 to 8 capture the SemanticWebServices from the AgentSWSInteractionViewpoint and place them on a list (swslist). In Line 9, the size of the *'swslist*' is controlled. If there is no element within the list, the editor will show an error.

```
Process:
    'Process' name=ID';'|
    process=[Process]  '{'
           'describes' sws=[SWS] ';'
           …
    '}';
Grounding:
    'Grounding'  name = ID';' |
    grounding  = [Grounding] '{'
          ('supports' sws=[SWS] ';') &
          ('calls' service += [WebService] ';')*
    '}';
Interface:
   'Interface'  name = ID ';' |
    interface=[Interface]  '{'
          'presents' sws=[SWS] ';'
          …
    '}';
```

Listing 11. Parts of Process, Grounding, and Interface rules

In regard to the constraints when creating plans, we can consider plan types in Agent-SWS Interaction viewpoint. According to SEA_L, textual concrete syntax, Semantic Service Plans (SS_RegisterPlan, SS_FinderPlan, SS_AgreementPlan and SS_ExecutorPlan), and their relations, must be in a specific order within the instance models. This order helps increasing readability of the program. These sequence restrictions are supplied with EBNF rules in Listing 13.

```
@Check
public void checkAtLeastOneSWS(
             AgentSWSInteractionViewpoint sws) {
     SWSInteractionViewpoint agent =
             EcoreUtil2.getContainerOfType(sws,
                   SWSInteractionViewpoint.class);
     List<SWS> swslist =
            EcoreUtil2.getAllContentsOfType(agent, SWS.class);
     if((swslist.size()<1))
            error("There must be at least one
                 SWS", AgentSWSInteractionPackage.Literals.
                    SWS_INTERACTION_VIEWPOINT__NAME);
}
```

Listing 12. Validation Package code for supplying at least one SWS constraint

According to Lines 2 and 3, any general Plan or Semantic Service Plan can be defined within the instance model. A Plan can be defined with or without its *'type'*, *'description'* and *'priority'* attributes. The '?' character at the end of each statement makes it optional. If Semantic Service Plans are considered, the order should be as defined in Lines 5 to 8. In Line 11, it is stated that one or more ‘advertises interface’ relation can exist. Similar rules are defined for other plan types in Lines 15-16, 20, and 24-25.

The Xtext can generate EBNF rules from a given metamodel. It can also generate a metamodel from the EBNF rules. However, we preferred to define EBNF rules manually in order to supply some syntactical restrictions and constraints such as defining relations in a specific order (Xtext cannot extract the order from the metamodel because the metamodel has not such an attribute by itself). It is worth noting that when starting from the already-existing metamodel and defining EBNF rules manually, one should be careful to properly match the metamodel with the grammar.

In this study, as mentioned previously, some controls are also used with a formatting package in addition to using some controls with a validation package. For example, some rules are defined for modifying the written program in order to rearrange the format of the code to gain more readability. Moreover, some other Xtext facilities are used, e.g. Wizard sample code, Highlighting (for keywords, comments, variables, etc), and Quick-fixing for errors.

```
01 Plan returns Plan:
02     ('Plan' name = ID (type=STRING)?
03     (description = STRING)?(priority=INT)? ';') | PlanSequence;
04 PlanSequence returns Plan:
05     reg  = SS_RegisterPlanDef
06     find = SS_FinderPlanDef
07     agree = SS_AgreementPlanDef
08     exe = SS_ExecutorPlanDef ;
09 SS_RegisterPlan:
10     plan=[SS_RegisterPlanDef]  '{'
11          ('advertises' interface+=[Interface] ';')+
12     '}';
13 SS_FinderPlan:
14     plan=[SS_FinderPlanDef] '{'
15          'interactsWith' matchmaker=[SSMatchmakerAgent]';'
16          ('discovers' interface+=[Interface]';')*
17     '}';
18 SS_AgreementPlan:
19     plan=[SS_AgreementPlanDef] '{'
20          ‘negotiates' interface=[Interface] ';'
21     '}';
22 SS_ExecutorPlan:
23     plan=[SS_ExecutorPlanDef]  '{'
24          'executes' process=[Process] ';'
25          'uses' grounding=[Grounding] ';'
26     '}';
```

Listing 13. Sample Plan rules

3.5. Code Generation

It is not sufficient to complete the DSL definition only by specifying the notions and their representations. A complete definition requires that one provides the semantics of language concepts in terms of other concepts, the meanings of which are already established. Therefore the syntax of the SEA_L is mapped into the metamodels of existing agent platforms that have well-defined, understood, and executable semantics. This mapping is provided through model transformations [5, 9, 31, 44]. Model to code transformations follow these model transformations and, finally, an executable software code is achieved for exact MAS.

In our study, code generation for the instance models is supplied with the Xpand tool [50]. Many of model driven engineering approaches accomplish code generation by writing strings to the text files. Xpand is a template engine which is used to make this process easier. It allows for creating textual output using EMF [10] models. The text output can be coded within any programming language. Xpand requires an EMF metamodel and one or more templates for translating the model into text. Once the requirements are provided, code generation can be provided by first defining an EMF model and running the generator. Xpand supplies traverse the abstract tree of the provided model and generate the code along the way [51].

In this study, Xpand is used for the generation of JADEX [23] code, along with OWL [36] and OWL-S [37] files from SEA_L specifications, and corresponding instance models. The code generation of JADEX agents from the SEA_L's Agent Internal viewpoint, and the generation of OWL-S SWS documents from SEA_L's Agent-SWS Interaction viewpoint, are exemplified in this paper.

JADEX is one of the popular APIs for developing software agents. JADEX code is composed of two files: the Agent Definition File (ADF), in which an agent's Beliefs, Goals, and Plans are defined using XML code, and the JADEX Plan File, in which Agent plans are defined using Java code. According to the JADEX platform, each agent has an ADF file. Therefore, in our study, an ADF file is generated for each SemanticWebAgent of a SEA_L instance model. The Beliefs, Goals, Plans, Behaviors, and Capabilities of SemanticWebAgents are defined within ADF with corresponding tags, but the JADEX Plan files include pure Java code for defining corresponding tasks.

In the generated code for SEA_L models, SEA_L ontological entities such as agent knowledge-bases are coded in OWL. Moreover, SWSs modeled in SEA_L instances are implemented according to OWL-S specifications. Both OWL and OWL-S are perhaps the most popular and in-use technologies for describing ontologies and SWS definitions.

An instance model, which conforms to the SEA_L metamodel, is in fact a platform independent model. In order to achieve its platform specific counterparts (e.g. its JADEX counterpart), mappings are needed between the SEA_L metamodel and metamodels of agent development frameworks (e.g., JADEX, JADE [22]). Since we focus on the JADEX platform in this study, we need to provide entity mappings between SEA_L and JADEX metamodels.

These mappings pave the way for transforming the source model (SEA_L) into the target model (JADEX). The mappings are illustrated in Table 1.

As discussed in subsection 3.1, the Agent Internal viewpoint focuses on the internal structure of every Agent within a MAS organization. Hence, in order to generate JADEX code, Agent Internal viewpoint is mapped to a JADEX metamodel. On the other hand, the Agent-SWS Interaction viewpoint represents the interaction between SemanticWebAgents and SemanticWebServices. Thus, it is mapped to both JADEX and OWL-S metamodels (see Table 1). The generated ontology files for Agent-SWS Interaction viewpoint are provided together with the ADF and Plan files for the Agent Internal viewpoint. Since the generations of ADF and Plan files for the Agent-SWS Interaction viewpoint are very similar to those for the Agent Internal viewpoint, it is not repeated here.

It is worth noting that both the mappings between SEA_L and JADEX and SEA_L and OWL-S take place simultaneously. In fact the SEA_L instance elements pertaining to agent and MAS viewpoints are transformed into JADEX instances while remaining elements of the same SEA_L instance model, which are used to model semantic web services, are transformed into OWL-S instances.

Table 1. Mapping between SEA_L, JADEX and OWL-S Metamodels

SEA_L	JADEX	OWL-S
SemanticWebAgent	Agent	
SSMatchmakerAgent	Agent	
Plan	Plan	
Behavior	Plan	
Capabilities	Capability	
Goal	AchieveGoal	
Goal	QueryGoal	
Goal	PerformGoal	
SS_AgreementPlan	Plan	
SS_ExecutorPlan	Plan	
SS_FinderPlan	Plan	
SS_RegisterPlan	Plan	
SemanticWebService		Service
Interface		ServiceProfile
Process		ServiceModel
Grounding		ServiceGrounding
Input		Input
Output		Output
Precondition		Condition
Effect		ResultVar

For code generation, a metamodel namespace is initially imported in order to make the meta-types known to the editor, as shown in Line 1 of Listing 14. Next, the main template is created. Each template is defined by a rule starting with the DEFINE keyword (see Line 2 of Listing 14). Xpand's keywords and meta-type references are always enclosed in '«' and '»' characters.

```
01  «IMPORT org::xtext::example::mydsl::myDsl»
02  «DEFINE main FOR SWSInteractionViewpoint»
03  …
04  «EXPAND owlservice FOREACH service»
05  «EXPAND owlsprofile FOREACH service»
06  «EXPAND owlsmodel FOREACH service»
07  «EXPAND owlgrounding FOREACH service»
08  «EXPAND wsdl FOREACH service»
09  «ENDDEFINE»
```

Listing 14. Defining main elements and invoking templates

Each template consists of a template name and meta-type on which the template can be called. In this way a template is rather like a sub-routine, parameterized by a meta-type and other optional parameters [27]. So, in our study, model transformations are supplied in a built-in way between the SEA_L, JADEX, and OWL-S metamodels. For example, a SemanticWebAgent element in an instance model of SEA_L is transformed into a JADEX Agent element while generating the code. These transformations are supplied regarding the mappings in Table 1.

In Listing 14, for each Service, *'owlservice'*, *'owlsprofile'*, *'owlsmodel'*, *'owlsgrounding'*, and *'wsdl'* (Web Service Definition Language) templates are invoked between lines 4 to 8. Each SemanticWebService is represented in a '*Service.owl*' file. For example, for an '*Electronic Barter Service*', an '*EBarterService.owl*' file will be produced. '*Service Profile*', '*Service Process*' and '*Service Grounding*' are described within the '*profile.owl*', '*process.owl*' and '*grounding.owl*' files, respectively.

According to the second line of Listing 15, a '*Service.owl*' file is created. The other lines of the code are added to the end of this file. The bold keywords (*int*, *pro* and *gro*) are the predefined variables representing the Interface, Process, and Grounding, respectively. Lines 4, 7 and 11 are the point references for the Profile, ProcessModel and Grounding, respectively. Also, the related service name will be written in generated code by using '«this.name»' in Lines 3, 5, 9, and 13.

Nested templates are defined for invoking input, output, precondition, and effect where they are needed. In the Agent Internal viewpoint, an ADF file is needed for each SemanticWebAgent, and a Plan file is needed for each Plan. Therefore, the Plans and SemanticWebAgent templates are invoked within the main template, as represented in Listing 16.

Listing 17 shows the Xpand code for creating Plan files. Lines 3 to 22 are all boilerplate texts for inserting into the plan file.

The code-block given in Listing 18 represents the belief definitions, as a sample element, within the generated ADF file. Beliefs are defined in <beliefs> tags. The attributes of a belief meta-entity are generated using Lines 3-5 of Listing 18.

Code generations for other parts of ADF (e.g. Goal and Capability) are realized in a similar manner.

```
«DEFINE owlservice FOR Service»
«FILE this.name + "Service.owl"»
<service:Service rdf:ID= "«this.name»">
    <!-- Reference to the Profile -->
    <service:presents rdf:resource="&«this.name»_profile;#
        «int.name»"/>
    <!-- Reference to the Process Model -->
    <service:describedBy
        rdf:resource="&«this.name»_process;#
        «pro.name»"/>
    <!-- Reference to the Grounding -->
    <service:supports
        rdf:resource="&«this.name»_grounding;#
        «gro.name»"/>
</service:Service>
```

Listing 15. A part of the Xpand code for defining the OWL-S Service File

```
«IMPORT  org::xtext::example::agentinternal::agentInternal»
«DEFINE   main  FOR AgentInternalViewpoint»
«EXPAND plans FOREACH plan»
«EXPAND semanticwebagents   FOREACH semanticwebagent»
«ENDDEFINE»
```

Listing 16. Sample template for invoking plans and semanticwebagents templates

Code generation for other viewpoints including the Environment, Role, Plan, and Interaction viewpoints are provided similarly. The required code generated from these viewpoints extend the agents' files, ADFs and plans, in the same way as Agent Internal and Agent-SWS Interaction viewpoints do.

As an expected result of applying MDD techniques, SEA_L simplifies the process of software development for MASs working within a semantic web environment. When considering the traditional approach for developing this type of software, a programmer should develop an ADF file (XML format) for each agent and a plan file (a Java file) for each plan of the agent, and then interconnect them. Also, the programmer should provide service, profile, grounding, process model, and WSDL documents for each semantic web service as required in the OWL-S standard. Meanwhile, the developer should consider the relation between these documents as well as the interaction between both the intra agents and agents with semantic web services. Therefore, the process is quite complex. However, in order to develop this type of software in SEA_L, the developer only needs to provide a program at the higher level (abstracting from the target platform constraints), which can help to produce all the above-mentioned documents and their interconnections, automatically.

```
«DEFINE plans FOR Plan»
«FILE name + ".java"»
import java.util.*;
import jadex.runtime.*;
import java.util.StringTokenizer;
public class «this.name»  extends Plan {
    // Plan attributes.
    ...
    // static block or constructor
    ...
    // Constructor code.
    public «this.name»() {
                        ...
    }
    // Plan main code.
    public void body() {
            // Send request
            ...
            // Wait for reply
             …
    }
}
«ENDFILE»
«ENDDEFINE»
```

Listing 17. Xpand code to generate JADEX Plan files

```
«DEFINE beliefs FOR Belief»
<beliefs
      Name = «this.name»
      Description = «this.description»
      dynamic = «this.dynamic»
/>
 «ENDDEFINE»
```

Listing 18. Sample Xpand code for defining beliefs in ADF

4. Related Work

Studies on DSLs and Domain-specific Modeling Languages (DSML) for agents have recently emerged, and these very few studies are still at their preliminary stages. For instance, a DSL called Agent-DSL is introduced in [28]. Agent-DSL is used to specify those agency properties that an agent should have in order to accomplish its tasks. However, the proposed DSL is only presented with its metamodel and just provides visual modeling of the agent systems according to agent features, such as knowledge, interaction, adaptation, autonomy, and collaboration. Likewise in [42], the authors introduced two dedicated modeling languages and call these languages

DSMLs. These languages are described by metamodels which can be seen as representations of the main concepts and the relations identified for each of the particular domains, again introduced in [42]. However, this study obviously included just the abstract syntax of the related DSMLs and does not give the concrete syntax or semantics of the DSMLs. In fact, the study only defines the generic agent metamodels for the MDD of MASs.

In [17], the author introduces a DSML for MAS. The abstract syntax of the DSML is derived from a platform independent metamodel, which is structured into several aspects each focusing on a specific viewpoint of a MAS. This approach is similar to our study. In order to provide the concrete syntax, the appropriate notations for the concepts and relations are defined in [49]. The semantics of the language is also given in [18]. These studies are noteworthy because they seem to provide the first complete DSML for agents, with all of its specifications. However it supports neither the agents on the Semantic Web nor the interaction of Semantic Web enabled agents with other environment members, such as semantic web services. Our study contributes to the aforementioned efforts by also specializing in the Semantic Web support of the MASs.

In [19], the authors introduce their approach on integrating agents with Semantic Web Services (SWSs) on a platform independent level. In addition to the MAS metamodel described in [17], a new platform independent metamodel is proposed for SWS. A relation between these two metamodels is established in a way that the MAS metamodel is extended with new meta-entities in order to support SWS interoperability and it also inherits some meta-entities from the metamodel proposed for SWS. Instead of using two separate metamodels, SEA_L has a built-in support for the modeling of agent and SWS interactions by including a special viewpoint. Moreover, semantic knowledge-base and agent internals can also be modeled in SEA_L.

Likewise, a new DSML is provided for MASs in [16]. The abstract syntax is presented using Meta-object Facility (MOF) [33] architecture. The concrete syntax and its tool are provided within a Graphical Modeling Framework (GMF) [11], and finally the code generation for the JACK agent platform [21] is realized by model transformations using JET [13]. However, the developed modeling language is not generic since it is only based on the metamodel of one of the specific MAS methodologies called Prometheus [38]. A similar study has been realized in [15] which proposes a technique for the definition of agent-oriented engineering process models and can be used for defining processes for creating both hardware and software agents. This study also offers the related MDD tool using Software & System Process Metamodel (SPEM) [34] and based on INGENIAS methodology [39] for MAS development. Nevertheless, similar to the DSML introduced in [17], neither [16] nor [15] cover software agents within the Semantic Web.

By considering our previous studies, in [25], we show how domain specific engineering can provide easy and rapid construction of Semantic Web enabled MASs. Ideas have been discussed for abstract syntax, concrete syntax, and formal semantics. Furthermore, a metamodel, which in fact constitutes the preliminary version of the abstract syntax of SEA_L, is introduced in [4]. Based on these building blocks, in this paper we have discussed SEA_L by including its syntax and semantics definitions, and shown how the language and its tools can be used during the development of real MASs.

5. Conclusion

This paper discussed the textual concrete syntax of a new DSL, called SEA_L, for Semantic Web enabled MASs. Additionally, we showed how the specifications of SEA_L can be used during the development of real MASs. Hence, agent software developers can first design their MASs by only taking care of the MAS domain specifications and abstracting from the target platform constraints. Following this domain specific design, the automatic application of predefined transformations enables developers to achieve executable code for the agent system that is intended for implementation in target platforms such as JADEX. Apart from its unique support for the Semantic Web, the use of SEA_L also brings an easier way of MAS development compared to merely programming with JADEX or any other specific MAS development framework.

For the concrete syntax, meta-elements are mapped to textual notations in Xtext, textual constraints are provided, and verification of these constraints was shown within the instance models. In this way, we have provided an interpreter mechanism and created an automatic code generation for users of the domain using Xtext and Xpand tools. Transformations from SEA_L to the other MAS platforms, e.g. JADE and JACK, are aimed in the next step. Hence, our Xpand-based interpreter for SEA_L presented in this paper can also be used for the implementation of SEA_L instances in other MAS platforms in addition to the JADEX.

As future work, we aim to evaluate SEA_L by providing two groups of MAS programmers with the same programming ability and then give them a real problem which can be solved by agents working within a semantic web environment. The first group would apply the classical approach of agent programming within the JADEX platform and semantic web programming in OWL-S. The second group would use SEA_L language to develop the solution and later they would add a complementary code (in JADEX and OWL-S) to the generated code by SEA_L. Based on their results, we would compare the development time, the amount of errors occurring for both groups, and the quality of the final code, again for both groups. In addition, we would compare the ratio of generated code with the full final code for the performance evaluation of SEA_L.

Acknowledgment. This study is funded as a bilateral project by the Scientific and Technological Research Council of Turkey (TUBITAK) under grant 109E125, and the Slovenian Research Agency (ARRS) under grant BI-TR/10-12-004. Also, we gratefully acknowledge the helpful comments from anonymous referees.

References

1. Badica, C., Budimac, Z., Burkhard, H. D., and Ivanovic, M.: Software agents: Languages, tools, platforms. Computer Science and Information Systems, Vol. 8, No. 2, 255-298. (2011)
2. Bradshaw, J. M.: Software Agents. MIT Press Cambridge, MA, USA. (1997)
3. Bratman, M. E.: Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, Massachusetts. (1987)
4. Challenger, M., Getir, S., Demirkol, S., and Kardas, G.: A Domain Specific Metamodel for Semantic Web enabled Multi-agent Systems. Lecture Notes in Business Information Processing, Vol. 83, 177-186. (2011)
5. Czarnecki, K., and Helsen, S.: Feature-Based Survey of Model Transformation Approaches. IBM Systems Journal - Model-driven software development, Vol. 45, Issue 3, 621-645. (2006)
6. Demirkol S., Challenger M., Getir S., Kosar, T., Kardas G. and Mernik, M.: SEA_L: A Domain-specific Language for Semantic Web enabled Multi-agent Systems. Second Workshop on Model Driven Approaches in System Development (MDASD 2012), held at Federated Conference on Computer Science and Information Systems (FedCSIS 2012), Wrocław-Poland, 9-12 September, 1373-1380. (2012)
7. Demirkol, S., Getir, S., Challenger M., and Kardas, G.: Development of an Agent based E-barter System. In International Symposium on Innovations in Intelligent Systems and Applications (INISTA), IEEE Computer Society, 193-198. (2011)
8. van Deursen, A., Klint, P., and Visser, J.: Domain-specific Languages: an annotated bibliography. ACM SIGPLAN Notices, Vol. 35, No. 6, 26-36. (2000)
9. Duddy, K., Gerber A., Lawley, M., Raymond, K. and Steel, J.: Model Transformation: A Declarative, Reusable Patterns Approach. In Azada, D. (Ed.) proceedings of Seventh IEEE International Enterprise Distributed Object Computing Conference (EDOC'03), IEEE Computer Society, Brisbane, Australia, 174-185. (2003)
10. Eclipse EMF: [Online] Available: http://www.eclipse.org/modeling/emf (Last access: March 2013)
11. Eclipse GMF: [Online] Available: http:// www.eclipse.org/modeling/gmp/ (Last access: March 2013)
12. Eclipse Help for Xtext: [Online] Available: http://help.eclipse.org/helios/index.jsp (Last access: March 2013)
13. Eclipse JET: [Online] Available: http://www.eclipse.org/modeling/m2t/?project=jet (Last access: March 2013)
14. Fowler, M.: Domain-specific Languages. Addison Wesley. (2011)
15. Fuentes-Fernandez, R., Garcia-Magarino, I., Gomez-Rodriguez, A. M., and Gonzalez-Moreno, J. C.: A Technique for Defining Agent-Oriented Engineering Processes with Tool Support. Engineering Applications of Artificial Intelligence, Vol. 23, Issue 3, 432–444. (2010)
16. Gascuena J. M., Navarro, E., and Caballero, A. F.: Model-Driven Engineering Techniques for the Development of Multi-agent Systems. Engineering Applications of Artificial Intelligence, Vol. 25, 159–173. (2012)
17. Hahn, C.: A Domain Specific Modeling Language for Multi-agent Systems. In Seventh International Conference on Autonomous Agents and Multi-agent Systems (AAMAS'08), ACM Press, 233-240. (2008)

18. Hahn C., and Fischer K.: The Formal Semantics of the Domain Specific Modeling Language for Multi-agent Systems. Lecture Notes in Computer Science, Vol. 5386, 145-158. (2009)
19. Hahn, C., Nesbigall, S., Warwas, S., Zinnikus, I., Fischer, K., and Klusch, M.: Integration of Multi-agent Systems and Semantic Web Services on a Platform Independent Level. In IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 200-206. (2008)
20. ISO/IEC 14977:1996 Standard, Information technology, Syntactic meta language - Extended BNF.
21. JACK: [Online] Available: http://aosgrp.com/products/jack/ (Last access: March 2013)
22. JADE: Java Agent DEvelopment Framework. [Online] Available: http://jade.tilab.com/ (Last access: March 2013)
23. JADEX: [Online] Available: http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview (Last access: March 2013)
24. Kardas G., Challenger M., Yildirim S., and Yamuc A.: Design and Implementation of a Multi-agent Stock Trading System. Software: Practice and Experience, Vol. 42, Issue 10, 1247–1273. (2012)
25. Kardas, G., Demirezen, Z., and Challenger, M.: Towards a DSML for Semantic Web enabled Multi-agent Systems. In International Workshop on Formalization of Modeling Languages, held in conjunction with the Twenty fourth European Conference on Object-Oriented Programming (ECOOP2010), ACM Press, 1-5. (2010)
26. Kos, T., Kosar, T., Knez J., and Mernik, M.: From DCOM interfaces to domain-specific modeling language: A case study on the Sequencer. Computer Science and Information Systems, Vol. 8, No. 2, 361-378. (2011)
27. Koster, V.: Implementation and Integration of a Domain Specific Language with oAW and Xtext. Technical Report. (2007)
28. Kulesza, U., Garcia, A., Lucena C., and Alencar, P.: A Generative Approach for Multi-agent System Development. Lecture Notes in Computer Science, Vol. 3390, 52-69. (2005)
29. Liu, S-H., Cardenas, A., Mernik, M., Bryant, B. R., Gray, J., and Xiong, X.: Introducing Domain-specific Language Implementation Using Web-Service Oriented Technologies. Multiagent and Grid Systems, Vol. 8, 19-44. (2012)
30. Macikenas, E., and Makunaite, R.: Applying Agent in Business Evaluation Systems. Information Technology and Control, Vol. 37, No. 2, 101 – 105. (2008)
31. Mens, T., and Van Grop, P.: A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Science, Vol. 152, 125-142. (2005)
32. Mernik, M., Heering, J., and Sloane, A.: When and how to develop domain-specific languages. ACM Computing Surveys, Vol. 37, No. 4, 316-344. (2005)
33. Object Management Group, Meta Object Facility (MOF) 2.0 Core Specification. [Online] Available: www.omg.org/spec/MOF/2.0/ (Last access: March 2013)
34. Object Management Group, Software & System Process Engineering Metamodel Specification Version 2.0, formal/2008-04-01, 2008. [Online] available at: http://www.omg.org/spec/SPEM/2.0/ (Last access: March 2013)
35. OMG ODM: [Online] Available: http://www.omg.org/spec/ODM/1.0/ (Last access: March 2013)
36. OWL: [Online] Available: http://www.w3.org/TR/owl-features (Last access: March 2013)
37. OWL-S: [Online] Available: http://www.w3.org/Submission/OWL-S/ (Last access: March 2013)

38. Padgham, L., and Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley & Sons, Ltd Publications. (2004)
39. Pavon, J., Gomez-Sanz, J. J., and Fuentes, R.: The INGENIAS Methodology and Tools. In Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-Oriented Methodologies, Article IX. Idea Group Publishing, 236–276. (2005)
40. Pešović D., Vidaković M., Ivanović M., Budimac Z. and Vidaković J.: Usage of Agents in Document Management. Computer Science and Information Systems, Vol. 8, No. 1, 193-210. (2011)
41. Rao, A., and Georgeff, M.: BDI Agents: From Theory to Practice. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 312-319, San Francisco. (1995)
42. Rougemaille, S., Migeon, F., Maurel, C., and Gleizes, M-P.: Model Driven Engineering for Designing Adaptive Multi-agent Systems. Lecture Notes in Artificial Intelligence, Vol. 4995, 318-33. (2007)
43. Schmidt, D. C.: Guest Editor's Introduction: Model-Driven Engineering. IEEE Computer, Vol. 39, No. 2, 25-31. (2006)
44. Sendall, S., and Kozaczynski, W.: Model Transformation: the Heart and Soul of Model-Driven Software Development. IEEE Software, Vol. 20, Issue 5, 42-45. (2003)
45. Shadbolt, N., Hall, W., and Berners-Lee, T.: The Semantic Web Revisited. IEEE IEEE Intelligent, Vol. 21, Issue. 3, 96-101. (2006)
46. Sycara, K.: Multi-agent Systems. AI Magazine, Vol. 19, 79-92. (1998)
47. Vallecillo, A.: A Journey through the Secret Life of Models. In Perspectives Workshop: Model Engineering of Complex Systems (MECS), 08331 in Dagstuhl Seminar Proceedings, Germany. (2008)
48. Varanda-Pereira, M. J., Mernik, M., Da Cruz, D., and Henriques, P. R.: Program Comprehension for Domain-specific Languages. Computer Science and Information Systems, Vol. 5, No. 2, 1-17, (2008)
49. Warwas S., and Hahn, C.: The Concrete Syntax of the Platform Independent Modeling Language for Multi-agent Systems. In Agent-based technologies and applications for enterprise interoperability, held in conjunction with the Seventh International Conference on Autonomous Agents and MASs, AAMAS. (2008)
50. Xpand: [Online] Available: http://wiki.eclipse.org/Xpand (Last access: March 2013)
51. Xpand documentation: [Online] Available: http://ditec.um.es/ssdd/xpand_reference.pdf (Last access: March 2013)
52. Xtext: [Online] Available: http://www.eclipse.org/Xtext/ (Last access: March 2013)

Sebla Demirkol received her B.Sc in Mathematics (Computer Science division) and M.Sc in Information Technologies from Ege University in 2009 and 2012 respectively. She is currently working as an Assistant Project Manager in Veripark Software Company. Her main research interests are model-driven development, multi agent systems and domain-specific languages.

Moharram Challenger received his B.Sc., and M.Sc. degrees in computer engineering from IAU-Shabestar and IAU-Arak Universities (Iran) in 2001 and 2005 respectively. Since 2006, he has been a tenure-track faculty member, as a lecturer, at computer engineering department, IAU-Shabestar University.

He is currently a Ph.D. candidate at Ege University, International Computer Institute with expected graduation of 2013. His research interests include domain-specific (modeling) languages, multi-agent and distributed systems with a current focus on the semantics of DSMLs. Moharram is also a student member of the IEEE and ACM.

Sinem Getir received her B.Sc in Mathematics (Computer Science division) and M.Sc in Information Technologies from Ege University in 2009 and 2012 respectively. She is currently a research assistant and a Ph.D. candidate in the University of Stuttgart. Her main research interests are model-driven development, formal semantics, model checking, and run-time verification. Other research interests are multi-agent systems and self-adaptive systems.

Tomaž Kosar received the Ph.D. degree in computer science at the University of Maribor, Slovenia in 2007. His research is mainly concerned with design and implementation of domain-specific languages. Other research interest in computer science include also domain-specific modelling languages, empirical software engineering, software security, generative programming, compiler construction, object oriented programming, object-oriented design, refactoring, and unit testing. He is currently a teaching assistant at the University of Maribor, Faculty of Electrical Engineering and Computer Science.

Geylani Kardas received his B.Sc. in computer engineering and both M.Sc., and Ph.D. degrees in information technologies from Ege University in 2001, 2003 and 2008 respectively. He is currently an assistant professor at Ege University, International Computer Institute. His research interests include model-driven software development, domain-specific (modeling) languages, agent-oriented software engineering and the Semantic Web. He is a member of the ACM.

Marjan Mernik received his M.Sc., and Ph.D. degrees in computer science from the University of Maribor in 1994 and 1998 respectively. He is currently a professor at the University of Maribor, Faculty of Electrical Engineering and Computer Science. He is also a visiting professor at the University of Alabama in Birmingham, Department of Computer and Information Sciences, and at the University of Novi Sad, Faculty of Technical Sciences. His research interests include programming languages, compilers, domain-specific (modeling) languages, grammar-based systems, grammatical inference, and evolutionary computations. He is a member of the IEEE, ACM and EAPLS.

Received: November 5, 2012; Accepted: April 29, 2013

DOI: 10.2298/CSIS121218068R

Using Reverse Engineering to Construct the Platform Independent Model of a Web Application for Student Information Systems

Igor Rožanc and Boštjan Slivnik

University of Ljubljana
Faculty of Computer and Information Science
Tržaška cesta 25, 1000 Ljubljana, Slovenia
{igor.rozanc,bostjan.slivnik}@fri.uni-lj.si

Abstract. A methodology for extracting the domain knowledge from an existing three-tier web application and subsequent formulation of the platform independent model (PIM) is described. As it was devised during a reverse engineering process of an existing web application which needed to be reimplemented on a new platform using new technology, it focuses on the domain knowledge and business functions. It produces the business model and the hypertext model leaving the presentation model aside. The methodology is semi-automated — the generation of the activity diagrams and parts of the hypertext model must be in part performed by an analyst, preferably the one with some domain knowledge. As the paper is primarily aimed at practitioners, a case study illustrating the application of the presented method is included.

Keywords: reverse engineering, web application, platform independent model, PL/SQL

1. Introduction

Reverse engineering of an existing software application can be aimed at different goals. One is to gain the insight into a competitors' product to learn how to replicate its design, the other is to discover possible patent infringements. Often it is performed to produce various kinds of documentation [5] as the documentation might be either outdated or even nonexistent. The final result of reverse engineering is a description of the application at a higher level of abstraction, but this may be understood differently by different people. If only the documentation has to be obtained, reverse engineering can result in a formal text description. However, if the result is to be used further on, i.e., for upgrading, modification or even reimplementation of the existing application, diverse design models are needed.

Although a software technical specification can be either missing or outdated simply because of a professional misconduct, the deficient software specification can result from a particular software development model used to produce the application. For instance, if the agile software development methodology is used [29], it is quite possible that no detailed software specification will ever be produced since one of the main principles of the "Manifesto for Agile Software Development" explicitly values *working software over comprehensive documentation* [3]. Furthermore, the agile methodology concentrates more on management rather than on technical aspect and documentation [25].

Later on the agile approach to software development can become an obstacle for the maintenance of the application for many reasons. First, porting an application from the existing platform to a new one might be difficult (even if the new platform is only a major new version of the existing platform). Second, business processes might change and since they are hard coded, a significant amount of the code must be changed. Third, in time people initially working on the application get replaced by new people with less insight into the application.

Hence, at certain point in the application's life cycle the existence of a model at a higher level of abstraction is an advantage for both managers and programmers [37]. Furthermore, it can also reduce maintenance costs [18]. The appropriate model can be produced even if no domain knowledge nor the application's architecture is known but even a limited amount of domain knowledge and insight into the application's architecture proves to be highly beneficial.

As the model-driven development (MDD) promotes a definition of the software development through a hierarchy of defined models at different levels of abstraction, it seems a natural choice for the formulation of the results obtained by reverse engineering [20, 30, 42]. These models are defined by the model-driven architecture (MDA) which implements the MDD. An important characteristic of MDA is promotion of the automatic generation of the lower level description models - the application code in the selected technology. Additionally, MDD promotes the use UML notation which became a standard modeling approach supported by various efficient tools. Thus by selecting the platform independent model (PIM) as defined in MDA for the final result of reverse engineering we gain a clear definition and an important advantage for the subsequent reimplementation of the application.

This paper describes a methodology of reverse engineering for producing the platform independent model (PIM) in order to modernize the application. In

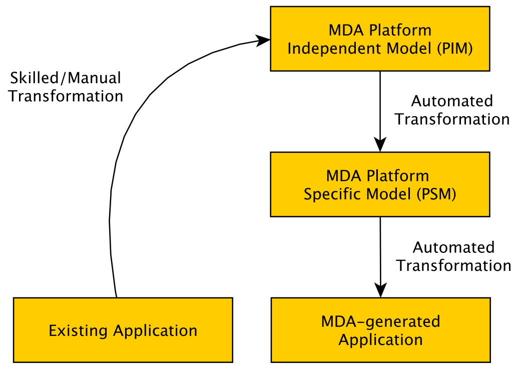

MDA Platform Independent Model (PIM)
Skilled/Manual Transformation
Automated Transformation
MDA Platform Specific Model (PSM)
Automated Transformation
Existing Application
MDA–generated Application

Fig. 1. The idea of reimplementation of an existing application using model-driven approach.

the ideal situation it would be used by the Architecture-Driven Modernization approach [41] as shown in Fig. 1. The methodology is especially targeted for situations where the team producing the PIM includes at least some members of the development team. Unlike some other papers [35], the paper includes an in-depth case study on how the methodology has been applied to a real world application. Initially the methodology advocated the generation of business model only [36]. As described here, it has been extended to include the generation of the hypertext model as well (leaving the presentation model aside as it is not needed if the application is reengineered). The generated class diagrams are now fully object oriented and by far the most critical part, i.e., the generation of the activity diagrams, has been improved significantly with (a) a new step of dead code elimination and (b) semi-automatic code simplification. Furthermore, the latest experiences with the described method are included as well.

Nowadays a large number of different languages, technologies and tools for the development of web applications is available. Hence, a completely general description of reverse engineering is almost impossible to formulate: either it is too general to be valuable as no concrete actions and procedures can be described or in trying to be comprehensive it becomes too large and imprecise. To ensure the paper has a practical value, the approach is given for the Oracle DB and Oracle Portal[1] as the selected case study is based on this technology. It is believed the Oracle technology is a good choice for presenting the new approach as it is (1) wide spread, (2) suitable for implementing the most complex web applications, and (3) a market leader and model for others. However, as PL/SQL is not object oriented (OO), reverse engineering of an existing PL/SQL application and generation of the PIM involve the shift to the OO-design.

The rest of the paper starts with Section 2 which contains an overview of the application used as a case study. Section 3 gives a short introduction into what the PIM of a web application should consist of. Section 4 describes elimination of unused parts of the application. Sections 5 and 6, the core of the paper, describe how the PIM can be extracted from an existing web application. The next two sections present the practical experience gained while producing the PIM for the actual real-world web application, and the discussion. The paper is concluded with a section on related work and conclusion. For the purpose of this paper, figures obtained from the generated diagrams have been manually translated to English.

2. A student information system as a case study

As the procedure presented in this paper sprang from practice, a case-study based on a web student information system named e-Študent (developed and initially deployed at the Faculty of Computer and Information Science of the University of Ljubljana, Slovenia) is to be introduced first [26, 27].

[1] The company, product and service names used in this paper are for identification purposes only — all trademarks and registered trademarks are the property of their respective holders.

e-Študent is a student information system developed and used at the University of Ljubljana, Slovenia. It is a three-tier web application built using the Oracle DB and Oracle Portal technology and written primarily in PL/SQL with some JavaScript. It consists of almost 800 different programming objects (dynamic pages, stored procedures and functions) with over 220.000 lines of code in total; its database contains almost 120 tables. The developer team consisted of people who were themselves developers and users at the same time, and hence the agile methodology seemed a natural choice [3, 29].

The development of e-Študent started in 2001 and by 2003 the initial release has been used by most faculties of the University of Ljubljana. Its main functions are providing electronic support for student enrollment, management of examination records and grades, and keeping the alumni records. All together it has been used by approximately 20.000 users (students, professors and staff). In 2008 the University of Ljubljana decided to replace e-Študent with at that time yet nonexistent successor. The reasons were many. First, the existing e-Študent was designed to be used by a single faculty and therefore different faculties were running their own instances instead of a single inter-faculty instance desired by the University. Second, as the number of elective courses increased dramatically by the introduction of the imminent new Bologna study programs (compared with the old pre-Bologna programs), the structure of the study programs was modified significantly and, sometimes even within the same faculty, in different directions.

The new system built after 2008 did not meet the expectations as it was focused more on implementing additional functionalities and less on suitable implementation of the existing, i.e., essential, ones. After the new system was made and evaluated it has been realized that during the development and maintenance of the original e-Študent a huge amount of the domain knowledge had been accumulated. In fact, due to the turbulent times of the Bologna reform, the source code of e-Študent is most likely the most comprehensive and the most formal specification of the student examination process. It is precisely this domain knowledge that should be extracted during reverse engineering, and it should be extracted as a model suitable for the development of e-Študent's successor after the first attempt, i.e., without eŠtudent domain knowledge, failed.

3. Platform independent model of a web application

Model-driven development is based on a notion of automatic transformations between different models describing an application on different levels. In the ideal situation, a developer would produce a platform independent model (PIM), add some platform specifications to reach the platform specific model (PSM), and finally generate the application [24]. Apart from this, the PIM provides a standard way to model an application in a technology independent way, i.e., by using UML diagrams. This approach is supported by a number of tools [42].

For web applications it has been advised [32] that the PIM should consist of

– *a business model*,

- *a hypertext model*, and
- *a presentation model*.

Apart from the (usual) business model describing the business processes, the hypertext model describes how web pages are built and linked while the presentation model contains details of the graphic appearance of a web application. The three-model web application description is a prevalent approach supported by most methods, however other model names, i.e., content, navigation and behavior model, may be used instead [7, 42].

It has been shown that by using the appropriate MD methodology, namely URDAD [39], out of 13 different kinds of UML diagrams only the following 4 kinds of diagrams are sufficient to produce the business model:

- *class diagram*,
- *use case diagram*,
- *activity diagram*, and
- *sequence diagram*.

The first three types are usually produced during the analysis phase (which is not an issue in reverse engineering) while all four types are needed during the design phase. Class diagrams are used for the service contract and for the collaboration context, use case diagrams are used for the responsibility identification and allocation, activity diagrams specify the full business process while sequence diagrams denote the user work flow and the success scenario [39].

The hypertext model is an abstract description of the composition and navigation between web application pages, page elements, and fragments of page elements [32]. This model is especially important in case of dynamic web applications with their distributed integration, user directed flow of execution and dynamic creation of HTML forms [33].

In most cases the PIM of an existing application is needed when the application must be reimplemented using the new platform. Hence, the focus is on the business model and the hypertext model; the presentation model is less important as it is to be replaced by a new one suitable for the new platform.

4. Elimination of the dead code and the dead data

During the development and especially during the maintenance developers produce a number of redundant code and data objects which are not used by the application anymore. Some of these objects might contain older or alternate solutions, some are temporary data collections, etc. Most of them are obsolete or even invalid.

The technique for dead code and dead data elimination is pretty straightforward: all objects not identified as live are considered dead. Live objects are identified using the following two steps:

1. Determining live root objects: In general, this step depends highly on the technology the application is made in. For the application that has been designed using the Oracle Portal and Oracle DB, the application's start page

	User pages	Dyn. pages	Procedures	Functions	Tables	Triggers
All objects	288	253	523	291	382	65
Live objects	236	186	388	207	117	58
Percentage	81.94	73.52	74.19	71.13	30.63	89.23

Table 1. The summary of dead code and dead data elimination.

and start pages of Portal user groups are retrieved from the Portal's Data Catalog.

2. Computing all live objects: Once the root objects are known, all objects reachable from the root objects must be identified. In other words, a transitive closure of the set containing the root objects must be computed. The connections among objects can be obtained by (a combination of) the following two methods:

   – by inspecting the metadata contained in the Portal's Data Catalog;
   – by parsing every object and extracting all links to other objects.

   Inspecting the metadata can most often produce perfectly adequate results. However, if nonstandard techniques have been used, the second method must be used also — it requires more effort since a parser for every programming language used within the application must be produced.

Note that the dead code elimination works on entire objects: if an object, i.e., a dynamic page, stored procedure or function, is found to be used, its entire contents is considered as live — dead code elimination within each particular code object is performed during the construction of activity diagrams.

Case study: e-Študent is an application made in Oracle Portal using Oracle DB. By inspecting the application's metadata in Oracle Portal the application's start page, five user groups and start pages for each user group were identified.

The core of the entry point for each user group is designed as a menu implemented in JavaScript (see also the description of use case diagrams below). By parsing JavaScript implementations of menus for all user groups the initial list of live user pages was obtained. Using Portal's Data Catalog all other live objects of the application were determined; apart from the user pages the list of live objects includes dynamic pages, stored procedures and functions, database tables and triggers.

The results of the dead code and dead data elimination are presented in Table 1. Note that more than 25 % of all code implementing the business logic, i.e., dynamic pages, stored procedures and functions, are not used. This is mainly due to the changes introduced during the Bologna reform. Likewise, almost 70 % of all tables are not used: many tables were introduced for live testing but later not removed for safety reasons.

5. Producing the business model

5.1. Class diagrams

Class diagrams represent the static structure of software systems in a graphical way [31]. Hence, it describes the structure of data and the structure of business processes.

In reverse engineering of a non-OO application based on the relational database the structure of data is obtained from the entity relationship model (ERM). The ERM might not exist or might be outdated, but it can be generated by inspecting the database using standard database tools. Once the relevant ERM is obtained, it is first transformed automatically into the conceptual class diagram, i.e., a class diagram without methods. This transformation encompasses the following rules [40]:

– Class definition: each entity is transformed into a separate class without any methods.
– Attribute definition: all table fields are transformed into attributes of appropriate classes.
– Associations definition: relations are transformed into associations and aggregations.

The names of classes and attributes are the same as the names of the corresponding entities and fields; classes corresponding to composite types get synthetic names.

The list of different tools that can carry out this transformation (at least to some degree if not entirely) includes tools like 'PowerDesigner' by SAP Sybase [4], 'UML Modeler for SQL' by Entrionics [2], and 'Altova UMODEL 2012' by Altova [1].

Although conceptional class diagram can be considered adequate [40], the proper way is to augment its classes with methods so that the resulting class diagram includes the description of behavior. During reverse engineering, the behavior is extracted from the stored procedures and functions of an existing application. To enhance the class diagram with methods, each stored procedure and function should be mapped into a method of a certain class. In cases where a stored procedure or function is associated with one table only, it is automatically transformed into a method of a class representing that table. Otherwise, a skilled analyst should manually determine the appropriate class.

Case study: The class diagram of e-Študent was made in two steps. In the first step, the ERM of e-Študent was automatically generated and transformed into the conceptional class diagram using PowerDesigner. The entire conceptional class diagram of e-Študent is shown in Fig. 2 which illustrates the overall structure of the original ER diagram and of the resulting (conceptual) class diagram at the same time.

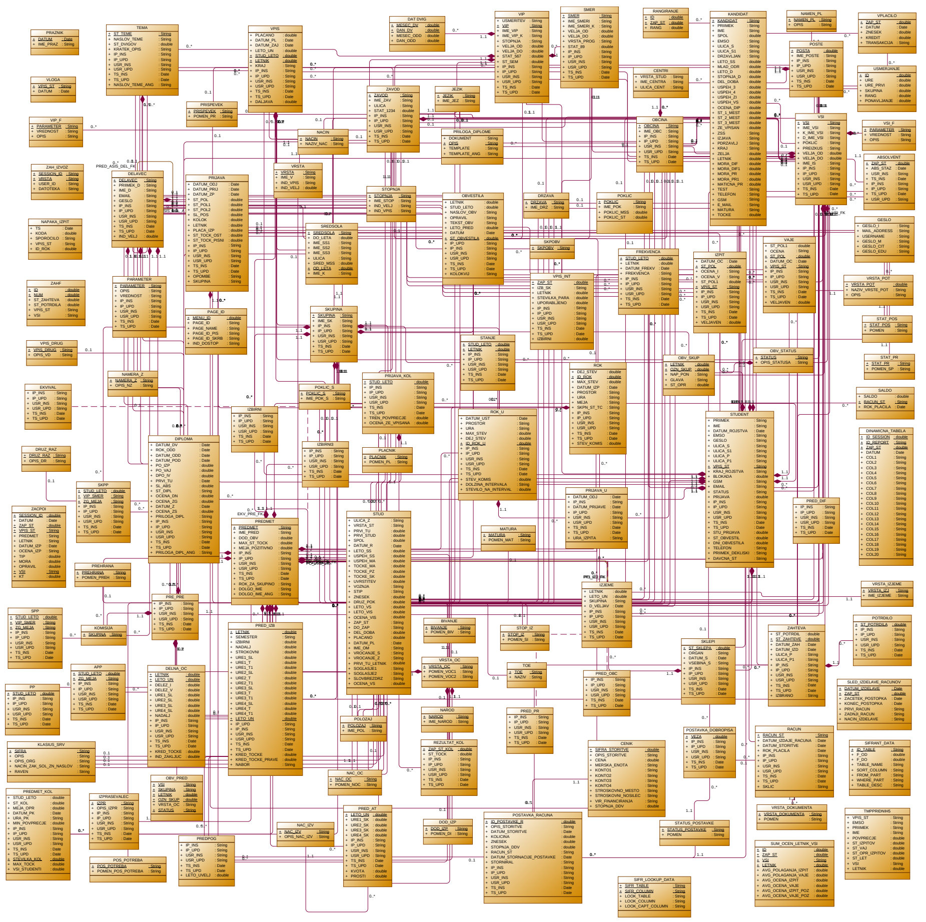


Fig. 2. The conceptual class diagram of e-Študent.

In the second step, the conceptual class diagram was transformed into the proper OO class diagram. The associations, i.e., the mapping of stored procedures and functions to tables, were first determined using the information available in Oracle DB Data Catalog and by subsequent parsing of stored procedures and functions source code[2]. In cases where a single stored procedure or function is associated with multiple tables, a heuristic was used to determine the list of most probable tables a stored procedure or function belongs to: tables with a higher number of inter table associations are placed higher on the candidate list. For each such stored procedure and function its sorted list of table candidates was presented to the analyst. He first selected the right

[2] The parser was made using the ANTLR v3.5-based PL/SQL parser by Patrick Higgins, `http://www.antlr3.org/grammar/list.html`.

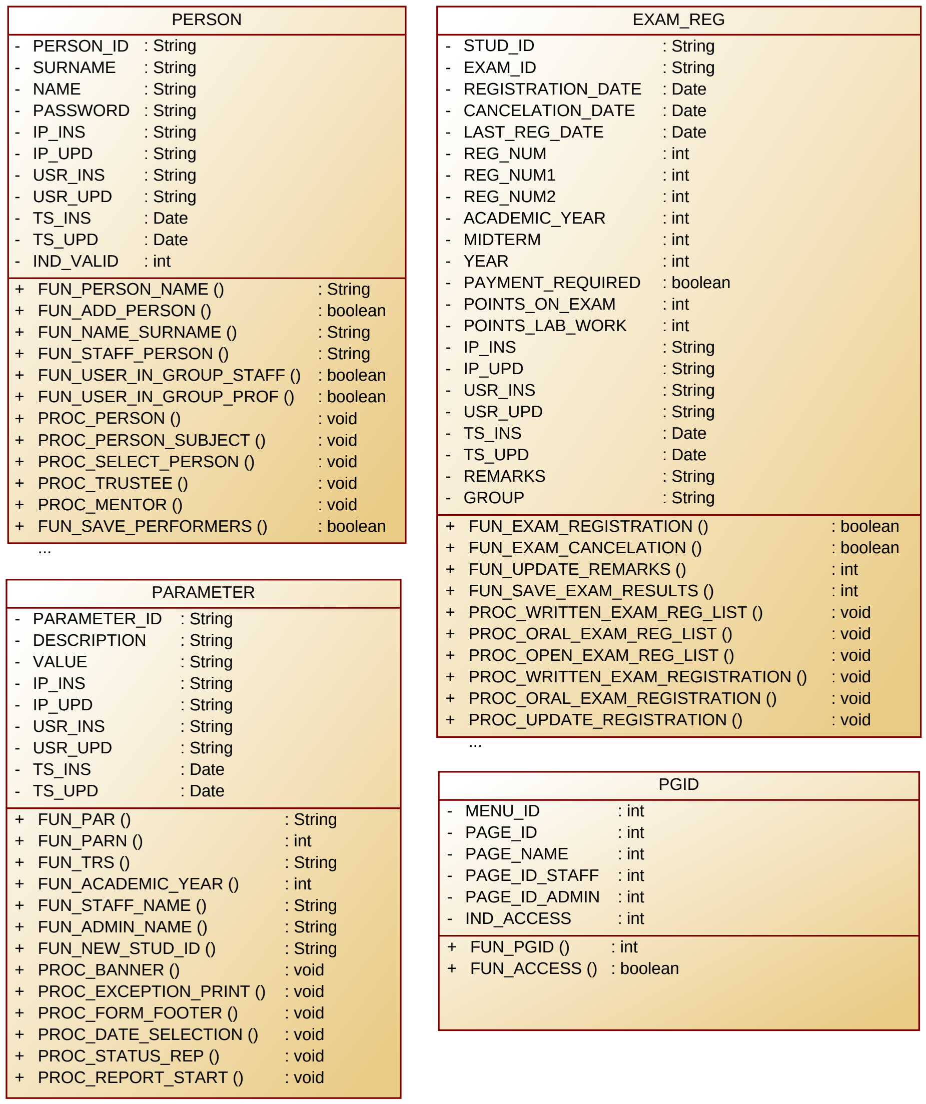

PERSON
- PERSON_ID : String
- SURNAME : String
- NAME : String
- PASSWORD : String
- IP_INS : String
- IP_UPD : String
- USR_INS : String
- USR_UPD : String
- TS_INS : Date
- TS_UPD : Date
- IND_VALID : int
+ FUN_PERSON_NAME () : String
+ FUN_ADD_PERSON () : boolean
+ FUN_NAME_SURNAME () : String
+ FUN_STAFF_PERSON () : String
+ FUN_USER_IN_GROUP_STAFF () : boolean
+ FUN_USER_IN_GROUP_PROF () : boolean
+ PROC_PERSON () : void
+ PROC_PERSON_SUBJECT () : void
+ PROC_SELECT_PERSON () : void
+ PROC_TRUSTEE () : void
+ PROC_MENTOR () : void
+ FUN_SAVE_PERFORMERS () : boolean
...
EXAM_REG
- STUD_ID : String
- EXAM_ID : String
- REGISTRATION_DATE : Date
- CANCELATION_DATE : Date
- LAST_REG_DATE : Date
- REG_NUM : int
- REG_NUM1 : int
- REG_NUM2 : int
- ACADEMIC_YEAR : int
- MIDTERM : int
- YEAR : int
- PAYMENT_REQUIRED : boolean
- POINTS_ON_EXAM : int
- POINTS_LAB_WORK : int
- IP_INS : String
- IP_UPD : String
- USR_INS : String
- USR_UPD : String
- TS_INS : Date
- TS_UPD : Date
- REMARKS : String
- GROUP : String
+ FUN_EXAM_REGISTRATION () : boolean
+ FUN_EXAM_CANCELATION () : boolean
+ FUN_UPDATE_REMARKS () : int
+ FUN_SAVE_EXAM_RESULTS () : int
+ PROC_WRITTEN_EXAM_REG_LIST () : void
+ PROC_ORAL_EXAM_REG_LIST () : void
+ PROC_OPEN_EXAM_REG_LIST () : void
+ PROC_WRITTEN_EXAM_REGISTRATION () : void
+ PROC_ORAL_EXAM_REGISTRATION () : void
+ PROC_UPDATE_REGISTRATION () : void
...
PARAMETER
- PARAMETER_ID : String
- DESCRIPTION : String
- VALUE : String
- IP_INS : String
- IP_UPD : String
- USR_INS : String
- USR_UPD : String
- TS_INS : Date
- TS_UPD : Date
+ FUN_PAR () : String
+ FUN_PARN () : int
+ FUN_TRS () : String
+ FUN_ACADEMIC_YEAR () : int
+ FUN_STAFF_NAME () : String
+ FUN_ADMIN_NAME () : String
+ FUN_NEW_STUD_ID () : String
+ PROC_BANNER () : void
+ PROC_EXCEPTION_PRINT () : void
+ PROC_FORM_FOOTER () : void
+ PROC_DATE_SELECTION () : void
+ PROC_STATUS_REP () : void
+ PROC_REPORT_START () : void
PGID
- MENU_ID : int
- PAGE_ID : int
- PAGE_NAME : int
- PAGE_ID_STAFF : int
- PAGE_ID_ADMIN : int
- IND_ACCESS : int
+ FUN_PGID () : int
+ FUN_ACCESS () : boolean

Fig. 3. Four classes of e-Študent class diagram.

table (usually the first table from the candidate list), and then, using Power Designer, declared the method (corresponding to the stored procedure or function in question) in the class based on the selected table. Four classes of the final class diagram are shown in Fig. 3. Unfortunately, the entire class diagram with all details (some methods have 20 parameters!) exceeds the available space. Method names are the same as names of the stored procedures and functions the methods are based (prefixes `PROC_` and `FUN_` are inherited from the existing eŠtudent code, not added). Keeping the same names simplifies the analysts transition from the old application to the new model.

5.2. Use case diagrams

A use case diagram represents system functionality by exhibiting the interactions between system's users and the transactions that provide value to users. They display relationship between actors and use cases [31]. In reverse engineering they are important as they offer unrivaled top-down insight of the system's functionality.

In most web applications, a use case diagram specifies a set of actions that can be performed by a certain user. In fact, as each user may play different roles, each use case diagram specifies main operations that can be performed by a particular user group. Hence a list of user groups must be retrieved from the system using one of the following two methods:

- by checking the list of user groups stored in the Portal;
- by inspecting the system from the user's point-of-view (in most cases, user roles can be determined by inspecting how each user logs into the application).

Once the list of user roles is established, one use case diagram per each user group should be produced. The generation of the use case diagram depends heavily on the technology and tools the web application is made with, and therefore no list of procedures applicable to all and every tool can be given here — for the Oracle DB/Portal case, see the case study below.

Users performing different user roles might be allowed to perform operations common to many different roles. Although a clear sign of an incautious design, two situations can nevertheless arise in practice:

- One particular function is found in different use case diagrams, either under the same name or under different names.
- Two functions found in two different use case diagrams share the same name but denote two substantially different actions.

Using the static analysis of the code it is possible to check whether two functions are carried out by the same code. However, if they are not, the resolution must be made manually by a developer/analyst.

Case study: Initially, the five user roles of e-Študent have been determined using the domain knowledge but the list of user roles kept by the Oracle Portal has been checked for verification.

The use case diagrams for e-Študent were obtained from the e-Študent menu files. The menu files were generated by HVMenu 5.41[3], a public-domain

[3] `http://www.dynamicdrive.com/dynamicindex1/hvmenu`; the latest version of HVMenu, i.e., 5.5, dates back in 2003.

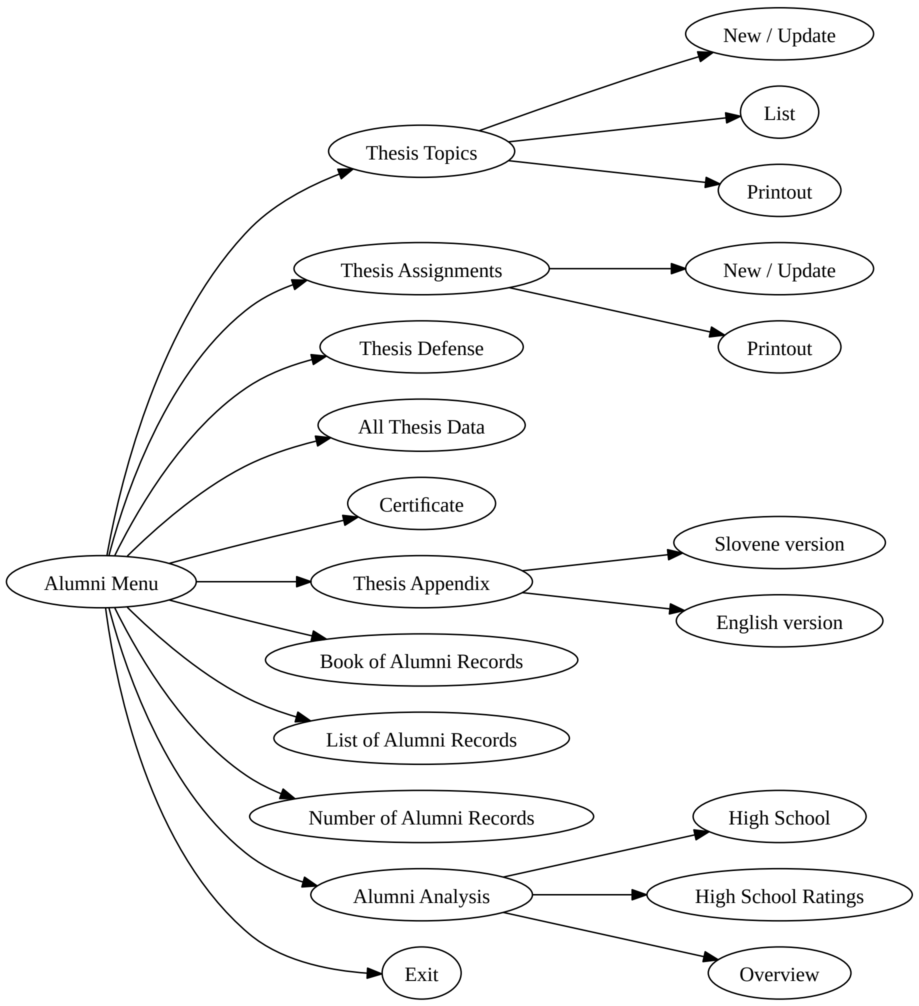


Fig. 4. The use case diagram for theses and alumni records: internal nodes represent (sub)-menus, leaves represent the main operations.

tool for producing Javascript code implementing menus to be used within web pages. Since generated, these files have a very indicative structure (one call of `Array` constructor per menu option) that allowed the entire structure of menus, submenus and options to be obtained by simple parsing. Hence, under the assumption (supported by the domain knowledge) that a single menu option reflect a single main operation a user can perform, the generation of use case diagrams was thus reduced to parsing Javascript menu files and producing the diagrams using the `dot` tool of GraphViz package by simply ranking graph nodes from left to right (`rankdir=LR`). Due to their size, only one of the resulting seven diagrams is shown; see Fig. 4. There are seven diagrams instead of five because the menu of one particular user group is split into three menus (one main menu and two submenus).

5.3. Activity diagrams

Activity diagrams are the most important artifacts in terms of the future reimplementation of the existing application as they denote the operational semantics of business processes [31].

At first glance, the activity diagrams can be produced automatically from the existing code. Certain tools are available, each specialized for a particular platform. However, no tool seems to be capable of generating activity diagrams for a real world application that would be suitable for reengineering the application. In most cases the generated activity diagrams are simply distilled code shown in a different form and thus they should not be used in the subsequent MDD for the following two reasons:

- The generated diagrams face granularity problem: understanding of the diagrams is obstructed as too many unnecessary details are included while sometimes some important details are systematically omitted [23, 43, 44].
- The generated diagrams may include some bad design elements which should not be propagated to the next versions of the application [23].

Hence, producing adequate activity diagrams cannot be fully automated but it can be machine supported. Namely, before the PL/SQL code is given to an analyst to retrieve the business logic, the code should be significantly simplified by automatically removing as much implementation details as possible. The heuristics used for code simplification is based on the classification of code fragments into the following categories:

- *object structure items*;
- *control structures* (`declare`, `begin`, `end`, `if`, `for`, etc.);
- *SQL blocks* (`cursor`, `select`, `update`, `fetch`, etc.);
- *data presentation* (`htp.` commands for data output);
- *data retrieval* (textbox, submit, button, checklist, etc.);
- *stored procedure and functions calls*;
- *other PL/SQL code*;
- *comments*;
- *other code*.

Each fragment containing an SQL block, data presentation or data retrieval is replaced by a single line. SQL block is replaced by the first SQL command augmented with the table used; the data presentation fragment and the data retrieval fragment are replaced by `print <data>` and `input <data>`, respectively, where `<data>` denotes the data either presented or read. The fragments of the last kind (other code) are removed.

The simplification introduces a risk some important details are removed. The analyst must be aware of this and when in doubt the original code of each simplified fragment must be checked.

The described method of code simplification is derived from the method used for producing workflows in IBM WebSphere Business Integration Workbench [43, 44], but it has been modified for the PL/SQL code in the Oracle platform — the list of categories has been compiled on the basis of authors' experience.

The code simplified in this way is raised to a higher level of abstraction and enhances the productivity of the analyst. Once the analyst performs the transformation of the PL/SQL code, the activity diagram can be generated from the simplified PL/SQL code automatically.

Case study: To actually produce the activity diagrams for e-Študent, the following sources needed to be reverse engineered:

– dynamic pages (HTML + PL/SQL),
– stored procedures and functions (PL/SQL),
– reports (SQL),
– triggers (PL/SQL), and
– JavaScript code.

A naive approach of using a tool to generate activity diagrams, e.g., 'UML Modeler for SQL' as

$$\begin{tikzcd} \text{PL/SQL code: DynPages,procedures,functions} \arrow[d, "\text{UML Modeler for SQL}"] \\ \text{Activity Diagrams,} \end{tikzcd}$$

failed exactly for the reasons outlined above. For a single stored procedure the number of elements within the activity diagram produced automatically using some tool is proportional to the number of lines of codes, and such a diagram is not readable even in case of moderate size procedures [36].

Illustrating the processing of a complete dynamic page exceeds the available space and thus only an excerpt is shown in Fig. 5. The first step consists of automatic PL/SQL code simplification. Classification of source code constructs and their subsequent removal or transformation in accordance with the rules specified above was implemented atop of another PL/SQL parser (also based on Patrick Higgins's ANTLR v3-based PL/SQL parser, see Subsection 5.1). After manual inspection and further simplification the activity diagram was generated using the reverse engineering options of 'UML Modeler for SQL'.

The entire set of activity diagrams includes 781 diagrams — one for every live dynamic page (186), stored procedure (388) and function (207). Manually

```
         ...
IF v_assistant = a THEN
  htp.p('<tr><td width="30%" align="right">');
  PROC_SELECT_PERSON(0);
  htp.p('</td><td width="70%">&nbsp;');
  PROC_SELECT_PERSON(1);
  htp.p('</td></tr>');
  t_person_id := FUN_STAFF_PERSON;
ELSE
  t_person_id := portal30.wwctx_api.get_user;
END IF;
htp.p('<td width="12%" align="right" height="30"><font class="label_star">*</font><font class=
  "label_text">Date from:</font></td><td width="88%" valign="middle">&nbsp;');
htp.formText(cname => 'f_from', cvalue => v_from_aux, csize => '10', cmaxlength => '10', cattributes =>
  'class="input_field" onChange="checkNull("f_from","f_from");isValidDate("f_from","f_from")"');
PROC_DATE_SELECTION('OpenExamRegs_form', 'f_from');
htp.p('<font class="help_text">(dd.mm.llll)</font></td>');
htp.p('</tr><tr>');
htp.p('<td width="30%" align="right" height="30"><font class="label_star">*</font><font class=
  "label_text">Date from:</font></td>');
htp.p('<td width="70%" valign="middle">&nbsp;');
htp.formText(cname => 'f_to', cvalue => v_to_aux, csize => '10', cmaxlength => '10', cattributes =>
  'class="input_field" onChange="checkNull("f_to","f_to");isValidDate("f_to","f_to")"');
PROC_DATE_SELECTION('OpenExamRegs_form', 'f_to');
         ...
```

⇓ PL/SQL simplification parser

```
         ...
IF v_assistant = 1 THEN
  PROC_SELECT_PERSON(0);
  PROC_SELECT_PERSON(1);
  t_person_id := FUN_STAFF_PERSON;
ELSE
  t_person_id := portal30.wwctx_api.get_user;
END IF;
INPUT v_from_aux;
PROC_DATE_SELECTION('OpenExamRegs_form', 'f_from');
INPUT v_do_pom;
PROC_DATE_SELECTION('OpenExamRegs_form', 'f_to');
         ...
```

⇓ manually

```
         ...
IF USER=ASSISTANT THEN
  PROFESSOR_SELECTION;
ELSE
  SET_USER_STAFF;
END IF;
INPUT_DATE1;
CHECK_DATE1;
INPUT_DATE2;
CHECK_DATE2;
         ...
```

UML Modeler for SQL ⟹

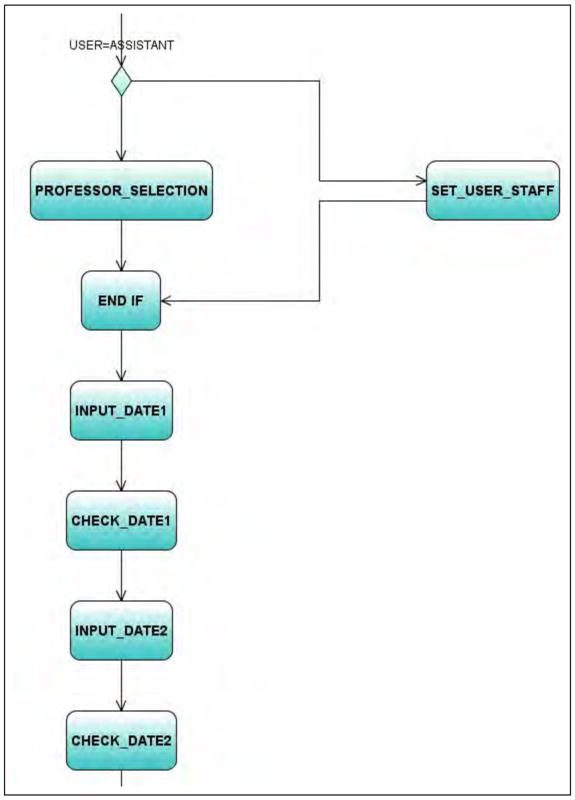


Fig. 5. Generating an activity diagram (due to its size only an excerpt is shown): the source code (top) is simplified twice (first automatically, then by an analyst) to be later transformed into to the activity diagram.

processing all this code objects is a tremendous task, but it is significantly reduced using the automatic code simplification and automatic generation of activity diagrams using UML Modeler for SQL. In many cases, e.q., maintenance or consulting, not all activity diagrams should be produced.

In e-Študent, the complexity of reports (generated using Oracle Report Builder), triggers and Javascript sources is not an issue. Therefore, they are considered as an appendix to the business model.

5.4. Sequence diagrams

Sequence diagrams express interactions and data flow between different objects within an application and thus describe a dynamic component of business processes. They may be used at different levels of abstraction to present different views of the application: usage scenarios (a description of a potential way the application is used), the logic of methods (explore the logic of a complex operation, function, or procedure) or the logic of services (a high-level method, often invoked by clients).

To generate sequence diagrams, a sequence of calls performed by every programming object must be obtained first. In PL/SQL code the programming object are dynamic pages (representing user's main operations) and stored procedures and functions (transformed to class methods).

Next, for each object a direct call tree is constructed: the root node contains the artifact itself and the leaves contain, from left to right, the artifacts called. Thus, the direct call tree is an ordered tree of height 1 if there are some calls from the programming object in the root node or 0 otherwise.

A direct call tree is transformed into a complete call tree by repetitive replacement of leaves with their direct call trees: a leaf is replaced if its direct call tree is of height 1 and no internal node on the path from the root to the leaf does not contain the programming object in the leaf. The transformation is completed once no leaf can be replaced any more.

Finally, the generated sequence diagram contains user's main operations (found in the use case diagrams) and classes (found in the class diagram) in the head sections while transitions are obtained by a preorder traversal of the complete call tree.

In this manner, a complete sequence diagram is obtained statically (disregarding PL/SQL control structures). There are at least two ways of producing the sequence diagram dynamically during reverse engineering. The first one is based on the reconstruction of clickstreams from the data obtained in the application's log files accumulated over many years [34]. The other way is to construct and apply a comprehensive set of test cases [9]. Using the dynamic approach, a number of sequence diagrams are obtained for each user's main operations while using the static method outlined above one complete sequence diagram per user's main operation, although bigger, is generated.

Case study: For all dynamic pages, stored procedures and functions of eŠtudent, sequences of inter-method calls were retrieved by a simple scanner of the source code additionally relying on information stored in Portal's Data Catalog. By another custom tool these sequences were transformed using the method described above into a GraphViz format and finally produced by `dot`.

An example of a sequence diagram is shown in Fig. 6. Due to the size of direct/complete call trees and the size of sequence diagrams, only a small excerpt of a single sequence diagram can be presented.

6. Producing the hypertext model

A suitable model is needed to adequately present the complex navigation of a dynamic web application. Several formal descriptions like FSM [10] and WebML [15] more or less meet this criterion, but the Atomic Section Model (ASM) [33] was chosen as it is capable of representing complex web applications with simple graphs. Furthermore, it can be produced using the static analysis.

ASM is a well known model primarily developed for testing web applications. Each HTML page is represented by a Component Interaction Model (CIM); CIMs of all HTML pages are combined together into an Application Transition Graph (ATG) representing ASM. Thus, the ATG of a web application is the formal representation of the hypertext model.

The CIM of a single HTML page is a quadruple $\mathrm{CIM} = \langle \mathrm{S}, \mathrm{A}, \mathrm{CE}, \mathrm{T} \rangle$ with

- a set of start pages S from which the page is referenced,
- a set of atomic sections A the page is made of,
- a component expression CE describing the page structure, and
- a set of transitions T pointing from and to (other) HTML pages.

An atomic section (AS) is a basic block of PL/SQL code producing the HTML page contents send to a client. The component expression is a regular expression denoting all possible sequences, selections and iterations of diverse ASs when dynamically constructing HTML page. All sets are fixed; in reverse engineering they can be retrieved by parsing HTML code or objects which dynamically create HTML pages, and by some manual processing by an analyst.

Originally the first component of a CIM is a single start page [33]. However, to produce the adequate hypertext model the first component of a CIM must be a set of all pages pointing to a page the CIM is made for. This is important for the construction of the ATG.

The ATG of an application is a quadruple $\mathrm{ATG} = \langle \Gamma, \Theta, \Sigma, \alpha \rangle$ with

- a set Γ of software components (CIMs),
- a set Θ containing all transitions of all CIMs,
- a set Σ of variables defining possible states of the presentation layer, and
- a set α of of all diverse starting pages (usually one).

The ATG is usually presented as a directed graph with a set of vertices Γ and a set of edges Θ.

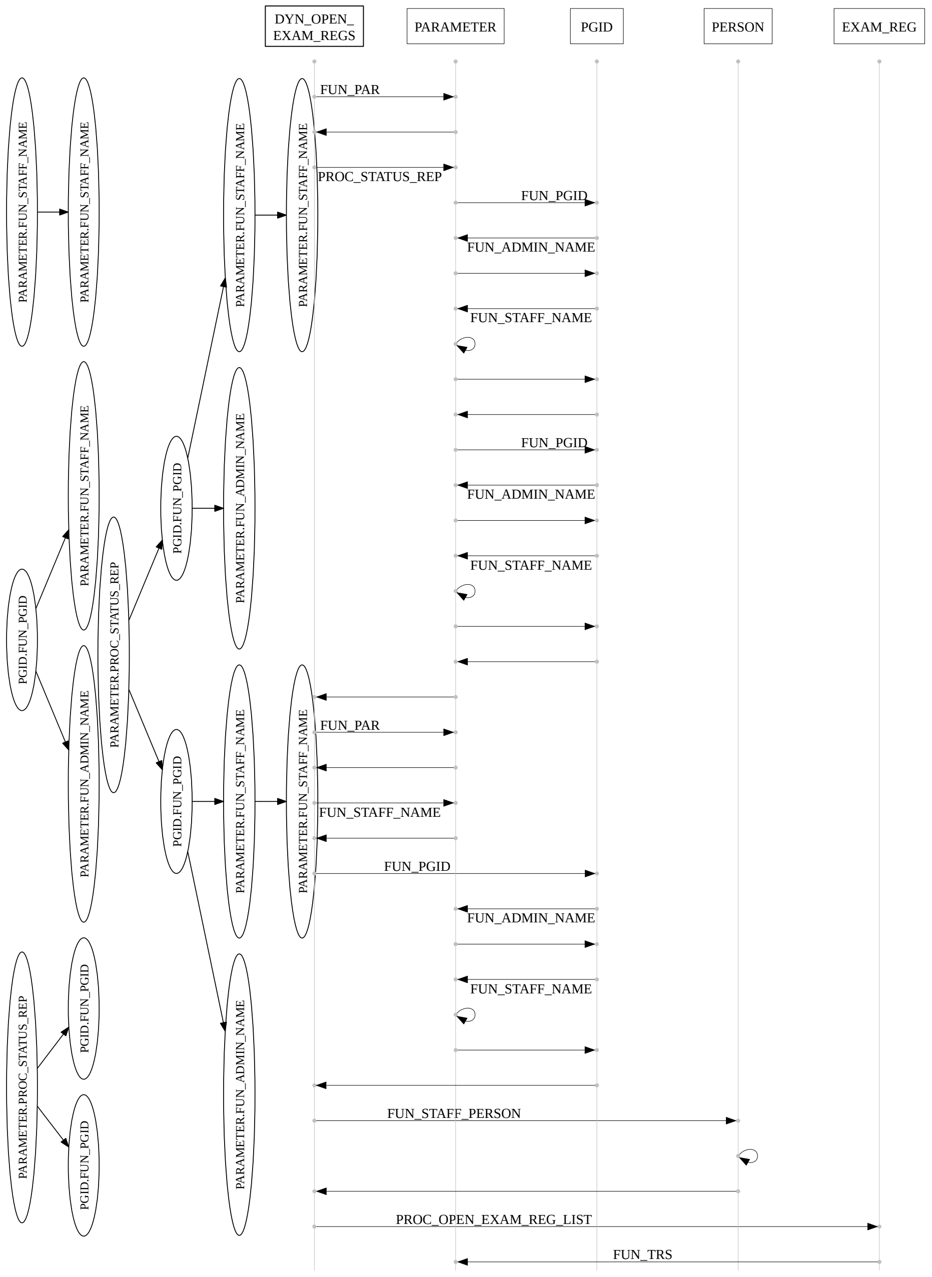


Fig. 6. An example of a sequence diagram generation (only cca 15 % is shown). The transitions corresponding to the call of PARAMETER.PROC_STATUS_REP from dynamic page DYN_OPEN_EXAM_REGS is based on the complete call graph (center) produced from the three direct call graphs (left).

A CIM is extracted from a dynamic page or a stored procedure using a method similar to extracting an activity diagram: it consists of (1) code simplification, (2) manual extraction of CIM structure and generation of graphical representation of CIM.

Code simplification is performed by the parser used for generation of activity diagrams except that the code fragments are classified into the following categories:

- *presentation elements* (code fragments which are copied to HTML page or produce the contents that is copied to HTML page);
- *links* (links to other HTML pages, SUBMIT parameter definitions, etc.);
- *control structures*;
- *procedure and function calls*;
- *other code* (SQL blocks, declarations, etc.).

All fragments classified as 'other code' are eliminated. Note that this code simplification concentrates on code fragments that were mostly left out by the code simplification used for producing activity diagrams.

Once the code is simplified, the CIM is produced manually by an analyst defining atomic sections and the control flow among them, and the transitions to and from other dynamic pages. Each atomic section represent a description of a group of presentation elements, stored procedure and function calls at a higher level of abstraction. The ATG is constructed simply by connecting CIMs using the transitions among them.

Case study: For instance, the CIM of a dynamic page for open exam registrations is shown in Fig. 7. It contains 15 atomic sections (labeled $\mathrm{P}_1 \ldots \mathrm{P}_{15}$) connected as described by the regular expression

$$\mathrm{P}_1((\mathrm{P}_2|\varepsilon)\mathrm{P}_3((\mathrm{P}_4(\mathrm{P}_5|\mathrm{P}_6)(\mathrm{P}_7|\varepsilon)\mathrm{P}_8)|(\mathrm{P}_9(\mathrm{P}_{10}|\varepsilon)\mathrm{P}_{11}(P12|\varepsilon)\mathrm{P}_{13})|\varepsilon)\mathrm{P}_{15})|\mathrm{P}_{14}$$

where P_1 is "HeadSectionAndTitle", P_2 is "StatusMissingURL", etc. Futhermore, it contains 4 transitions, namely

$$\begin{aligned}
&\mathrm{MenuProfessor} \longrightarrow \mathrm{DYN\_OPEN\_EXAM\_REGS}[\mathrm{POST}(\bot,\bot,\bot,\bot)]\\
&\mathrm{MenuStaff} \longrightarrow \mathrm{DYN\_OPEN\_EXAM\_REGS}[\mathrm{POST}(\bot,\bot,\bot,\bot)]\\
&\mathrm{DYN\_OPEN\_EXAM\_REGS.PrintReport} \longrightarrow \mathrm{GetReport}[\mathrm{POST}(\mathrm{id})]\\
&\mathrm{DYN\_OPEN\_EXAM\_REGS.EndOfBodySection} \longrightarrow\\
&\qquad \mathrm{DYN\_OPEN\_EXAM\_REGS}[\mathrm{POST}(\mathrm{f\_from},\mathrm{f\_to},\mathrm{f\_button},\mathrm{f\_print})]
\end{aligned}$$

The first two transitions specify where is this dynamic page reachable from while the third one specifies which dynamic page is used after the main operation performed by this dynamic page has been finished. The fourth transition specifies a return to the same dynamic page once the parameters have been refined using HTML form elements contained within the page.

To produce CIMs for eŠtudent, the source code of a dynamic page was simplified by the same parser as used for activity diagrams but using the code

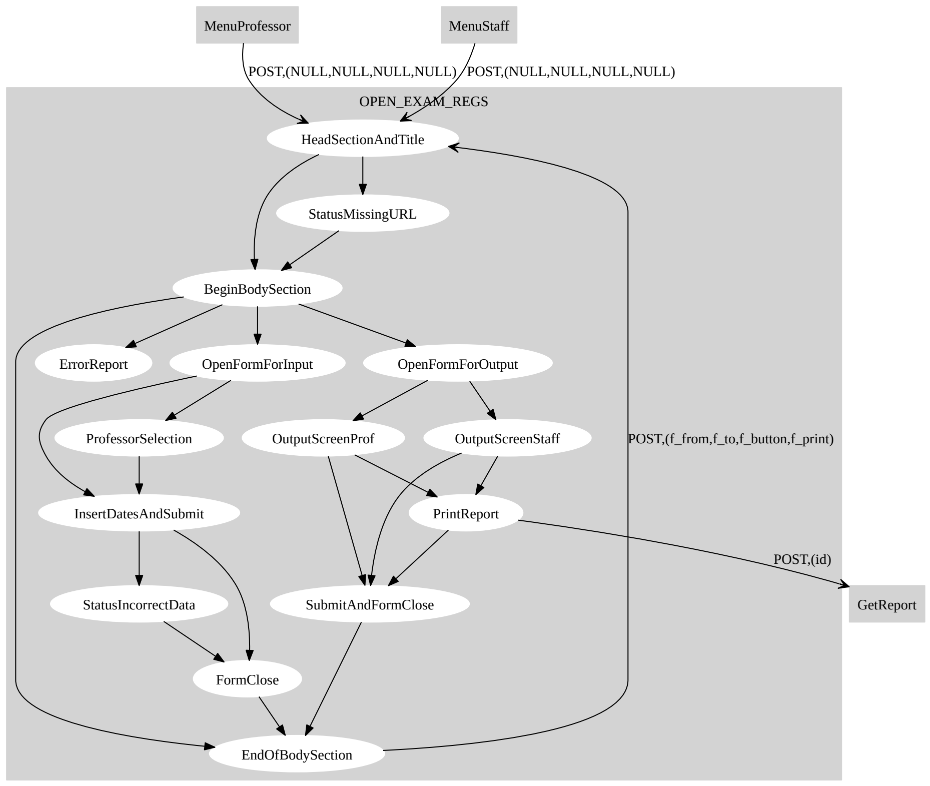


Fig. 7. The CIM for the dynamic page for processing open exam registrations.

classification as described above. Once the analyst produced the CIM description, the graphical presentation of CIMs was generated using the `dot` tool from the GraphViz package.

Apart from the four pages used during the login process, e-Študent contains five entry points, one for each user group. After a quick check it was established that one CIM must be produced for each of 186 dynamic pages 349 of 388 stored procedures and 65 of 207 stored functions. Thus, the hypertext model of e-Študent consists of almost 600 CIMs. To produce the ATG, each transition is augmented with a link to a `dot` file containing a CIM of a dynamic page reachable by the transition.

7. Lessons from the case study

Since the start of the reverse engineering of eŠtudent in 2012 [36], the focus of the reverse engineering has changed. Namely, in autumn 2012 the initial goal of producing the PIM suitable for fully automatic reengineering of eŠtudent has been replaced by producing the PIM suitable for maintenance and consulting.

The change of the goal was due to the decision of the new development team not to implement the new eŠtudent from the described models because of the complexity of the application, major modifications of the existing requirements and a number of new requirements.

The initial eŠtudent development team consisted of 9 developers but only one developer (the first author) actively maintains the application nowadays. The construction of the models proved to be a great advantage during maintenance for a number of reasons. First, it provided the maintainer a consistent top-down understanding of various aspects of the entire application. Second, it enabled the maintainer to understand the entire source code even though most of the past developers are no longer of any help. It turned out that code modifications (due to an urgent requirement changes) or bug fixes were much easier to perform once the corresponding activity diagrams were constructed. Before the diagrams were produced, finding and fixing the part of the code to be changed and testing the fix demanded significantly more work.

Once the new development team started on the next generation of eŠtudent in autumn 2012, the models were used intensively for the domain knowledge transfer during the design phase of the new application. The class diagram was transferred into the new application and later modified according to new requirements, and the use case diagrams were user for establishment of the compulsory tasks.

The consultant (the first author) found the activity diagrams made at a higher level of abstraction as described in this paper invaluable since the new development team needed information on operations at a higher level of granularity. More precisely, once the main elements of the PIM were available, the consulting became far more comprehensive and compact. The need for consulting services decreased sharply after approximately 3 months during which the consultant-analyst was producing and conveying the model (especially the activity diagrams). Hence, the method proved its importance in practice. Whenever low-level details were required, the original source code was preferred to the automatically generated activity diagrams, i.e., as described in [36].

As the construction of models requires a considerable amount of analyst's work, the models were constructed to the extent requested by maintenance needs and consulting work only. Additionally, the construction of the models was interleaved with the development of the necessary custom tools and thus no clear separation of time used for each of these two tasks is available.

The construction of class diagram and use case diagrams took 2 man-weeks to complete. The most of this time was spent for assigning methods based on stored procedures and functions to classes of the conceptual class diagram.

As expected, the most time consuming part of the reverse engineering proved to be the construction of activity diagrams. After the initial attempt described in [36], the dead code elimination and source code simplification were introduced. Now, an analyst roughly familiar with eŠtudent needs approximately 1 hour to produce the activity diagram for a relatively simple dynamic page shown in Fig. 5. Based on all activity diagrams constructed (51 in total so far) during maintenance and consulting, we estimate that 2 hours per activity diagram are needed on average and thus for the entire set of 781 activity diagrams 40 man-weeks suffices.

As the generation of sequence diagrams is fully automated, the time needed for the generation is not an issue. The CIMs are usually produced by the same analyst as activity diagrams and thus approximately 60 % less time is needed as for the activity diagrams (but since less CIMs than activity diagrams has been produced so far, this estimation is less reliable).

Many papers agree that a large part of the reverse engineering must be performed manually, but the paper by Di Lucca et al. is one of the few that provide at least some metrics on actual reverse engineering of a web application. Their conclusions are the same as ours: “the most expensive steps are those requiring human intervention” [17, p. 96] (even though there are some inconsistencies regarding Table V on the same page).

8. Discussion

An important issue in reverse engineering is to properly define the focus:

1. If the produced model can be used by the MDD tools to generate the new version of the application, a lot is gained since the automated code generation is less error prone and can be done quickly for different platforms.
   However, to produce the adequate model for automated MDD, the proper understanding of the existing web application must be gained first. This understanding usually requires a shift to a higher level of abstraction which, as all authors agree, cannot be fully automated. Only after the model on the higher level of abstraction is obtained, the usual forward engineering involving “understanding $\rightarrow$ model $\rightarrow$ code” can be applied [32].
2. Otherwise, the produced model can serve as a well formalized documentation and therefore it enables the switch from agile software development to other software development methodologies.
   Furthermore, a proper formal documentation provides a foundation for efficient maintenance and consulting. Like above, the models must be shifted on the higher level of abstraction in order to yield an insight rather than simply machine readable description.

In either case, the shift to a higher level of abstraction involves primarily processing of the existing source code to produce proper activity diagrams since these diagrams represent the main formalization of business logic. The next in line are CIMs since they specify the navigation aspect. Generation of both kinds of diagrams requires analyst support.

PL/SQL proved to be far too low-level formalism any automatic transformation into the PIM could be successful [13]. Even if activity diagrams are generated automatically, they are too detailed and include too many past design elements (including bad solutions that should not be propagated to newer versions).

There are certain tools available that can ease reverse engineering significantly (Visual Paradigm, UModel, PowerDesigner, . . .), but for two reasons the task cannot be fully automated using any of them. First, even if they produce a formally correct model or a part of it, it is usually too large and too detailed, and thus inadequate for later use as shown in [36]. Second, obviously no tool can gain a proper insight into the logic behind the application. This proves to be the major obstacle in reverse engineering of an application based on the entity relationship diagrams and PL/SQL code as producing the PIM involves a shift to the OO design. Hence, a number of custom tools must be made during reverse engineering to support but not replace the manual construction of the PIM.

As it turns out reverse engineering of PL/SQL code must be done by someone who is at least to some degree familiar with the design of the system being reverse engineered. There are at least three operations that must be human assisted: transformation of the entity relationship diagrams to class diagrams, transformation of the PL/SQL code to activity diagrams, and transformation of the presentation elements of the PL/SQL code into the hypertext model.

Furthermore, a lesson learnt the hard way is that if a reverse engineering of a PL/SQL-based web application that was produced by agile development is to be economically viable, reverse engineering must be performed by at least some members with the domain knowledge who participated in the development of the application.

Finally, even though PL/SQL is not a good starting point for a reverse engineering towards the PIM, sometimes it simply must be done. And although it might encompass a lot of manual processing, the resulting model is worth the effort [18].

9. Related work

Recently, a number of authors reported a different approaches to reverse engineering of dynamic web or similar applications. Favre described a MDA-based framework of platform-independent and platform-dependent models that are to be produced by reverse engineering of object-oriented code [20]. It is "propose(d) to apply static and dynamic analysis to generate models" [20] but no actual procedures for the generation of these models are given and no test based on a real-world application is included.

Di Lucca et al. [17] presented a reverse engineering process for dynamic web application supported by the WARE tool. Both static and dynamic analysis were performed to describe business model using class diagrams, use-case diagrams and sequence diagrams. Although similar to our work, their approach does not include activity diagrams — but only activity diagrams enable a comprehensive understanding of business logic, i.e., not just what tasks are performed but also how they are actually performed.

Concentrating on the actual procedures of static reverse engineering, Zou et al. describe how a business model definition can be constructed after undocumented changes to the source code were made [44, 45]. Like ours their technique involves source code simplification using a heuristic based on categorizing programming language constructs. However, they only describe the generation of workflows which roughly correspond to activity diagrams while our method covers other models of PIM and the hypertext model as well.

Bellucci et al. described the MARIA tool to perform static reverse engineering of user interfaces of dynamic web applications [12]. The static transformation from the concrete language (HTML, CSS, Ajax, JavaScript) to the abstract (platform independent) one is described — again it is based on categorizing source code constructs. The tool produces the description of the user interface at two different abstract levels but it cannot be used to generate a complete business and hypertext model.

The dynamic approach to reverse engineering has also been considered. Like Zou et al. [44, 45] Di Francescomarino et al. presented a reverse engineering process leading to the web application's business model only using the dynamic analysis of GUI-forms [16]. Amalfitano et al. focused on dynamic analysis of Rich Internet Applications (RIA) user interface to produce a proper description using Finite State Machines (FSMs) [8]. Similarly, Marcheto et al. presented a ReAjax tool to perform on Ajax web applications [28]. It is focused on Ajax specifics and produces GUI-based state model. Alalfi et al. [6] perform reverse engineering to obtain UML sequence diagrams for PHP web applications using instrumentation and analysis of execution traces. Likewise, Briand et al. described reverse engineering of distributed Java applications to produce sequential diagrams as well [14].

To summarize, certain papers include no or very little concrete procedures for reverse engineering [19, 20, 35] while we include the generation of these models as well. Other papers focus on a single aspect, i.e, workflow based business model [44, 45], reverse engineering of UIs [12, 16, 28], sequence diagrams [6, 14], or leave out some important aspect [17]. However, our approach tends to come as close as possible to the PIM of a web application which should consist of a business model, a presentation model and a hypertext model as many authors working on a standard (forward) modeling agree, i.e., UWE [21, 22], WebML [15], OOHDM [38], Netsilon [32], W2000 [11].

10. Conclusion

Instead of yet another paper describing a methodology of a reverse engineering for producing different models, we concentrated on one particular kind of web applications, namely those written primarily in PL/SQL and based on Oracle Portal/DB. Our methodology produces the business and the hypertext model, both at the level of abstraction suitable for human insight into the application. In presenting it, we focused on procedures rather than on a tool that might implement them.

We tried to avoid the approach of (1) some sources [35] which describe what models are to be made without giving any hints about how this models could be made, (2) of some sources [44, 45] which do not provide the entire PIM, or (3) of some sources [17] that leave out the most important components, i.e., the activity diagrams. We find these approaches inadequate as the problems are not in the selection of the appropriate models but in gaining the insight into the existing application and the subsequent applicability of the produced models.

The methodology has been tested on the real-world application intensively used in practice, i.e., a student information system eŠtudent. The models retrieved by reverse engineering have been used successfully for maintenance and consulting. We believe that the produced models represent a viable starting point for the design of a model suitable for automated MDA.

References

1. Altova UModel 2012, http://www.altova.com/umodel.html (retrieved June 4th, 2013)
2. Entrionics UML modelling for SQL, http://www.entrionics.com (retrieved June 4th, 2013)
3. Manifesto for agile software development, http://agilemanifesto.org (retrieved: June 4th, 2013)
4. SAP Sybase PowerDesigner, http://www.sybase.com/products/modelingdevelopment/powerdesigner (retrieved November 5th, 2012)
5. Akkiraju, R., Mitra, T., Thulasiram, U.: Reverse engineering platform independent models from business software applications. In: Telea, A.C. (ed.) Reverse Engineering - Recent Advances and Applications, chap. 4, pp. 83–94. InTech Press (2012)
6. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Automated reverse engineering of UML sequence diagrams for dynamic web applications. In: Proceedings of the 2nd International Conference on Software Testing, Verification and Validation (ICST'09). pp. 287–294. IEEE Computer Society Press, Denver, CO, USA (2009)
7. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Modelling methods for web application verification and testing: state of the art. Software Testing, Verification and Reliability 19, 265–296 (2009)
8. Amalfitano, D., Fasolino, R., Tramontana, P.: Experimenting a reverse engineering technique for modelling the behaviour of rich internet applications. In: Proceddings of the IEEE International Conference on Software Maintenance (ICSM'09). pp. 571–574. IEEE Computer Society Press, Edmonton, AL, Canada (2009)
9. Ammann, P., Offut, J.: Introduction to Software Testing. Cambridge University Press, New York, NY, USA (2008)
10. Andrews, A.A., Offut, J., Alexander, R.T.: Testing web applications by modeling with fsms. Software & Systems Modeling 4(3), 326–345 (2005)
11. Baresi, L., Garzotto, F., Paolini, P.: Extending UML for modeling web applications. In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences. pp. 1–10. Honolulu, HI, USA (2001)
12. Bellucci, F., Ghiani, G., Paternò, F., Porta, C.: Automatic reverse engineering of interactive dynamic web applications to support adaptation across platforms. In: Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces (IUI'12). pp. 217–226. Lisbon, Portugal (2012)
13. Billig, A., Busse, S., Leicher, A., Süss, J.G.: Platform independent model transformation based on triple. In: Proceddings of the 5th ACM/IFIP/USENIX International Conference on Middleware (Middleware'04). pp. 493–511. Springer-Verlag, New York, NY, USA (2004)

14. Briand, L.C., Labiche, Y., Leduc, J.: Toward the reverse engineering of UML sequence diagrams for distributed Java software. IEEE Transactions on Software Engineering 32(9), 642–663 (2006)
15. Ceri, S., Piero, F., Bongio, A.: Web modeling language (WebML): a modeling language for designing Web sites. Computer Networks 33(1–6), 137–157 (2000)
16. Di Francescomarino, C., Marchetto, A., Tonella, P.: Reverse engineering of business processes exposed as web applications. In: Proceedings of the 13th European Conference on Software Maintenance and Reengineering. pp. 139–148. IEEE Computer Society Press, Kaiserlautern, Germany (2009)
17. Di Lucca, G.A., Fasolino, A.R., Tramontana, P.: Reverse engineering web applications: the WARE approach. Journal of Software Maintenance and Evolution: Research and Practice 16(1-2), 71–101 (2004)
18. Dzidek, W.J., Arisholm, E., Briand, L.C.: A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Transactions on Software Engineering 34(3), 407–432 (2008)
19. Escalona, M.J., Gutiérrez, J.J., Rodríguez-Catalán, L., Guevara, A.: Model-driven in reverse: the practical experience of the AQUA project. In: Proceedings of the 2009 Euro American Conference on Telematics and Information Systems: New Opportunities to increase Digital Citizenship (EATIS'09). pp. 17:1–17:6. ACM, Prague, Czech Republic (2009)
20. Favre, L.: Formalizing MDA-based reverse engineering processes. In: Proceedings of the 6th International Conference on Software Engineering Research, Management and Applications (SERA'08). pp. 153–160. IEEE Computer Society Press, Prague, Czech Republic (2008)
21. Knapp, A., Koch, N., Zhang, G.: ArgoUWE: A CASE tool for web applications. In: Proceedings of the 1st International Workshop on Engineering Methods to Support Information Systems Evolution. Geneva, Switzerland (2003)
22. Koch, N., Kraus, A.: The expressive power of UML-based web engineering. In: Proceedings of the 2nd International Workshop on Web-Oriented Software Technology (IWWOST'02). pp. 105–119. Malaga, Spain (2002)
23. Liebarman, B.: UML activity diagrams: Versatile roadmaps for understanding system behavior. The Rational Edge p. 12 (2001)
24. Luković, I., Varanda Pereira, M.J., Oliveira, N., da Cruz, D., Henriques, P.R.: A DSL for PIM specifications: Design and attribute grammar based implementation. Computer Science and Information Systems 8(2), 379–403 (2011)
25. Mahnič, V.: A case study on agile estimating and planning using scrum. Electronics and Electrical Engineering 5(111), 123–128 (2011)
26. Mahnič, V., Drnovšček, S.: Introducing agile methods in the development of university information systems. In: Proceedings of the 12th International Conference on European University Information Systems (EUNIS 2006). pp. 61–68. Tartu, Estonia (2006)
27. Mahnič, V., Rožanc, I., Poženel, M.: Using e-business technology in a student records information system. In: Proceedings of the 7th WSEAS International Conference on E-Activities (E-Activities'08). pp. 589–594. Cairo, Egipt (2008)
28. Marchetto, A., Tonello, P., Ricca, F.: Reajax: a reverse engineering tool for ajax web applications. IET Software 6(1), 33–49 (2012)
29. Martin, R.C.: Agile Software Development, Principles, Patterns, and Practices. Prentice Hall, Englewood Cliffs, NJ, USA (2002)
30. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA distilled - principles of model-driven architecture. Addison-Wesley, Boston, MA, USA (2004)

31. Miles, R., Hamilton, K.: Learning UML 2.0. O'Reilly Media, Sebastopol, CA, USA (2006)
32. Muller, P.A., Studer, P., Fondement, F., Bezivin, J.: Platform independent Web application modeling and development with Netsilon. Software & System Modeling 4(4), 424–442 (2005)
33. Offutt, J., Wu, Y.: Modeling presentation layers of web applications for testing. Software & Systems Modeling 9(2), 257–280 (2010)
34. Poženel, M., Mahnič, V., Kukar, M.: Separation of interleaved web sessions with heuristic search. In: Proceedings of the IEEE International Conference on Data Mining (ICDM). pp. 411–420. IEEE Computer Society Press (2010)
35. Raghupathi, W., Umar, A.: Exploring a model-driven architecture (MDA) approach to health care information systems development. International Journal of Medical Informatics 77(5), 305–314 (2008)
36. Rožanc, I., Slivnik, B.: Producing the platform independent model of an existing web application. In: Proceedings of the Federated Conference on Computer Science and Information Systems. pp. 1385–1392. IEEE Computer Society Press, Wroclaw, Poland (2012)
37. Rugaber, S., Stirewalt, K.: Model-driven reverse engineering. IEEE Software 21(4), 45–53 (2004)
38. Schwabe, D., Rossi, G., Barbosa, S.D.J.: Systematic hypermedia application design with OOHDM. In: Proceedings of the 7th ACM Conference on Hypertext. pp. 116–128. ACM, Bethesda, Maryland, USA (1996)
39. Solms, F., Loubser, D.: Generating MDA's platform independent model using URDAD. Journal of Knowledge-Based Systems 22(3), 174–185 (2009)
40. Sparks, G.: Database modelling in UML. Methods & Tools 9(1), 10–23 (2001)
41. Ulrich, W.M., Newcomb, P.: Information Systems Transformation: Architecture-Driven Modernization Case Studies. Morgan Kaufmann, Burlington, Mass., USA (2010)
42. Valderas, P., Pelechano, V.: A survey of requirements specification in model-driven development of web applications. ACM Transactions on the Web 5(2), article(10) (2011)
43. Zou, Y., Guo, J., Foo, K.C., Hung, M.: Recovering business processes from business applications. Journal of Software Maintenance and Evolution: Research and Practice 21(5), 315–348 (2009)
44. Zou, Y., Lau, T.C., Kontogiannis, K., Tong, T., McKegney, R.: Model-driven business process recovery. In: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04). pp. 224–233. IEEE Computer Society Press, Delft, The Netherlands (2004)
45. Zou, Y., Zhang, Q., Zhao, X.: Improving the usability of e-commerce applications using business processes. IEEE Transactions on Software Engineering 33(12), 837–855 (2007)

Igor Rožanc received the M.Sc. and Ph.D. degrees in computer science from the University of Ljubljana in 1995 and 2003, respectively. He is currently at the University of Ljubljana, Faculty of Computer and Information Science. Throughout his career he has been actively involved in the design and development of student information systems. His research interests include distance learning and programming methodologies.

Boštjan Slivnik is an Assistant Professor at the University of Ljubljana, Faculty of Computer and Information Science where he received the M.Sc. and Ph.D. degrees in computer science in 1996 and 2003, respectively. His research interests include parsing algorithms, compilers, formal languages, scheduling and distributed algorithms. He has been a member of the ACM since 1996.

Received: December 18, 2012; Accepted: August 23, 2013.

DOI:10.2298/CSIS121228059D

Model Execution: An Approach based on extending Domain-Specific Modeling with Action Reports

Verislav Djukić[1], Ivan Luković[2], Aleksandar Popović[3], and Vladimir Ivančević[2]

[1] Djukić – Software Solutions,
Gärtnerstrasse 17, 90408 Nürnberg, Germany
info@djukic-soft.com
[2] University of Novi Sad, Faculty of Technical Sciences,
Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia
{ivan,dragoman}@uns.ac.rs
[3] University of Montenegro, Faculty of Natural Sciences and Mathematics,
Džordža Vašingtona bb, 81000 Podgorica, Montenegro
aleksandarp@rc.pmf.ac.me

Abstract. In this paper, we present an approach to development and application of domain-specific modeling (DSM) tools in the model-based management of business processes. The level of Model-to-Text (M2T) transformations in the standard architecture for domain-specific modeling solutions is extended with action reports, which allow synchronization between models, generated code, and target interpreters. The basic idea behind the approach is to use M2T transformation languages to construct submodels, client application components, and operations on target interpreters. In this manner, M2T transformations may be employed to support not only generation of target platform code from domain-specific graphical language (DSGL) models but also straightforward use of models and appropriate DSM tools as client applications. The applicability of action reports is demonstrated by examples from document engineering, and measurement and control systems.

Keywords: domain-specific modeling, model-driven development, model transformations, modeling tools, document engineering

1. Introduction

Over the last few years, there have been increased efforts within the academic community to improve software engineering through application of software models [33]. In numerous works, there are remarks that the adoption of Model Driven Software Development (MDSD) and the Unified Modeling Language (UML) as its main language has only partially achieved the proclaimed goals related to development productivity and software quality [19], [21]. Some authors consider the unfitness of UML for domain specific problems to be the main reason for this failure. Expecting that an average software engineer uses or thinks in domain independent abstractions might have been unrealistic. Several approaches, including Domain Specific Modeling (DSM) and MDSD, still focus on software models, which are sufficiently formal but also understandable to both machines and humans. One of the important goals in the aforementioned approaches is that models should not only be part of the specification but also of the implementation of the corresponding systems.

Software industry experts are more pragmatic in regard to these issues and not determined to use general purpose modeling languages, such as UML, at all costs. They are more focused on developing modeling tools that satisfy requirements for highly specialized production and control systems. Although the quality and usability of these tools are not being questioned, the manufacturers are constantly faced with high costs of development and customization, even for very similar domains. Taking all into consideration, we expect that the software industry will base its highly specialized tools on the DSM architecture to a much greater extent. The following two improvements could be particularly important: (i) better support for the construction of modeling languages and their syntax, including abstract, concrete graphical, and concrete textual syntax; and (ii) better synchronization between meta-models, models, generated code, and target interpreters or “execution machines”. Our research is oriented toward the latter improvement, i.e., better synchronization between meta-models, models, generated code, and target interpreters. The aforementioned synchronization is closely linked to model debugging and execution.

The topic of our research presented herein is also present in other domains of application within the field of software engineering. One such domain is software development based on MDSD and Computer Aided Software Engineering (CASE) tools. The traditional CASE tools support the creation of platform independent model (PIM) software specifications, their automatic transformation into platform specific model (PSM) specifications, and ultimately the generation of program code. However, it cannot be actually expected that these tools support incremental interpretation of specifications and dynamic changes of the applied meta-models. These requirements may be gradually fulfilled in the evolution of CASE tools into MDSD tools by insisting on retaining the complete synchronization between the created PIM models and the generated program code. An example of one such MDSD tool, which is developed by the authors of this paper, is the Integrated Information Systems CASE Tool (IIS*Case) [26]. At present, this tool relies on the PIM model of an information system to generate: (i) implementation description of a database schema and (ii) prototypes of the applications supporting operations on that database. In the current version, any modification within the model requires a new generation of the implementation description of the database schema, as well as a new generation of the prototype applications. In this manner, in forward engineering, there is support for a one-way synchronization. One of the future research tasks includes implementing in IIS*Case the automatic two-way synchronization between the model and the system executing the applications. As opposed to the existing abovementioned approaches to the execution of models created using a DSM tool, our approach supports incremental interpretation of specifications. Each user operation on a model in the DSM tool is directly interpreted in real time, which may be utilized to verify the correctness of the specification. Simulation tools have supported this approach for quite some time, but they set restrictions on the semantics of simulation languages, i.e., meta-modeling is considerably limited. The execution of models whose semantics is not known in advance represents a significantly more complex problem with respect to both the theoretical and practical issues. The most difficult problems are the definition and automatic generation of a target interpreter that supports incremental verification of specifications. Moreover, the goal of our approach, to which we actively direct our efforts, is to support the two-way synchronization by allowing the direct execution of changes on a model. This may be achieved by using operations on the application that represents the result of the incremental specification. There should be support also for the direct extension of a meta-model in real time according to the operations executed on the previously created models.

Our initial application of MDSD, DSM, and model transformation principles is related to complex problems in document engineering, previously presented in [7], [11], [14], [22], [24], [26]. Positive experience with the construction and application of domain specific languages (DSLs), together with problems related to the development of client applications for measurement and control systems, indicated that the Model-to-Text (M2T) transformations in DSM may be significantly improved and utilized in model debugging and execution. By employing extended M2T transformations, namely "action reports", we intend to make possible the use of modeling tools as client applications. Notwithstanding the fact that current techniques for code generation from models have great capabilities, we demonstrate herein the practical value brought by: the introduction of the submodel concept and appropriate operations; the introduction of the transaction concept in the context of (sub)models; and the use of action reports (generators) as synchronization units during the testing of meta-models, models, client applications, and target interpreters. The practical value of introducing submodels, transactions, and action reports, is that M2T transformations, in addition to being employed for the generation of code in a target language, may also be used for expressing semantics of user actions on a PIM, i.e., on the graphical interface of a DSM tool.

In order to refer to the activities related to meta-modeling (Me), modeling (M), interpretation (I), and documenting (D) of model changes and execution flow, we introduce the term/acronym *MeMID activities*. Consequently, the approach to the modeling and development of software systems that includes all of the aforementioned activities is named *the MeMID approach*. When compared to the traditional approach to modeling, the MeMID approach includes interaction between all of the components in the DSM architecture, incremental specification, and visual representation of all changes within a real system being modeled. We took a pragmatic approach to the issue of model execution, with the goal of having solutions that may be sufficiently understood by a wide range of users and quickly applied in various business domains. The emphasis is placed neither on the definition of syntax of user semantic actions, nor on meta-modeling, but on the definition of action semantics, i.e., on the interpretation of user actions in a DSM tool during their execution and not solely afterwards, during code generation.

Besides the Introduction and Conclusion, the paper contains eight sections. In Section 2, we describe the state of the art and what is expected from DSM for model execution. The description of the concept of action reports and how they differ from code generators may be found in Section 3. In Section 4, we describe Model-to-Application (M2A), Application-to-Model (A2M), and Model-to-Document (M2D) transformations with respect to application generation. In Section 5, we describe usage of submodels and transactions in the testing of a DSL, model, and target framework or interpreter. This is illustrated with examples of using DSM tools for modeling documents, document templates, and modeling systems by documents. In Section 6, we describe how arbitrary user components may be integrated into DSM tools with the goal of visually representing abstract language concepts. In Section 7, we give examples of the synchronization between a client application and modeling tool. Section 8 describes usage of action reports for the purpose of implementing operations on DSM models, the target interpreter, and user applications. Chapter 9 contains a survey of related work, and overview of the current state of technology in the area of model execution.

2. State of the Art and MeMID Activities

There are certain differences between the roles of some elements in the architecture of DSM and UML tools. These roles originate from different perspectives on modeling in domain specific (DSM) and general purpose (UML) tools. On one hand, DSM tools promote unrestricted construction of domain-specific languages tailored to the needs of users in narrow business domains. On the other hand, UML tools promote construction and use of profiles that are tailored to a particular domain but retain basic elements of the UML syntax, as in the case of SysML [34]. Moreover, DSM tools allow rapid construction of any language belonging to the UML group, while UML tools feature a more suitable graphical interface. In DSM tools, a model is completely separated from the target language, i.e., models are fully platform independent. In UML tools, there is an early coupling between a model and the target language. In DSM tools, reverse engineering is regarded as a methodologically inappropriate procedure, while it is indispensable in UML tools for the purpose of synchronizing code and model. Nonetheless, these observations are fairly general since there are significant differences even between the tools of the same group.

Further evaluation of the state of the art in the area of model execution is done with respect to the aspects of traditional and advanced code generation and execution (Fig. 1). A modeling language is constructed using a dedicated editor, while models are created using the newly constructed language. In the DSM architecture, these steps correspond to meta-modeling and modeling activities. PIMs are transformed into source code in a general purpose programming language. Transformations are done using patterns or navigation languages [15], [30]. The generated source code in some language (e.g., IEC 611.31, C++, Java, and C#) is translated into binary code using a compiler so that it could be executed on the target platform. This DSM use case is marked as Traditional Flow in Fig. 1. In some cases, target platforms are operating systems themselves, but they may often be Run-Time Systems (RTSs) or Execution Machines, which feature a set of functions more suited for the concrete purpose when compared to operating systems. In our opinion, traditional use of DSM tools significantly improves productivity in the system development, but also has serious drawbacks.

The basic drawbacks of the traditional approach include: (i) weak synchronization between the generated code, model, and meta-model, which hinders incremental execution of models; and (ii) growth of specifications. As the specification is growing, the model should be executed accordingly, first, as empty, and later as more complex, while for each action on the model there should be a corresponding interpretation in the target RTS.

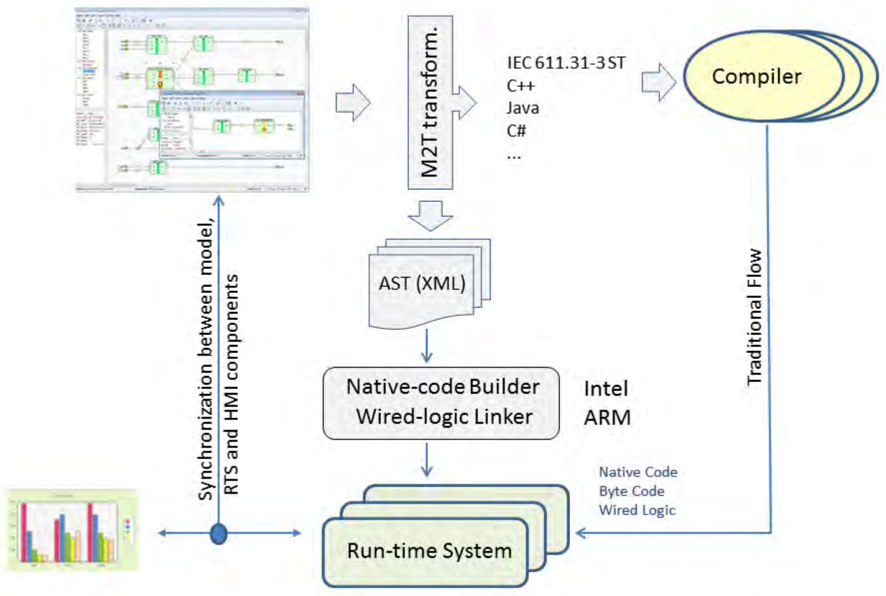


Fig. 1. Traditional and advanced usage of DSM tools

In the traditional approach, which is based on transformations into a general purpose language, the semantics expressed by a PIM may be significantly limited by a transformation to a target general purpose language (GPL). The approach that we propose, which is illustrated herein in Fig. 1 and with several examples tested in practice, includes:

– direct translation of PIM models to binary code tailored to the characteristics of the target RTS and hardware;
– dynamic linking of specifications being executed using increments, which are the result of changes in the model;
– use of action report interpreter within DSM tools, Human-machine interface (HMI) components, and the RTS for the purpose of their synchronization;
– application of arbitrary user components for the visualization of abstract DSL concepts; and
– run-time visualization of the interpretation of specifications within the DSM tool.

As indicated in Fig. 1, at the level of M2T transformations, an extended abstract syntax tree (AST) is generated. It is an Extensible Markup Language (XML) structure, from which it is possible to generate code in binary, assembly, or a general purpose programming language. Depending on the characteristics of the RTS and target hardware, various protocols for dynamic linking of binary code to the RTS are employed. These protocols specify how to exchange data on variables, arrays, user structures, external functions, and values of object instances. If the modeling language is sufficiently rich, there is no need for a host language, and, consequently, for a GPL compiler. We consider this approach especially suitable for target RTSs that support: incremental updating, dynamic linking of binary code, and execution of instructions used to communicate with wired logic controllers. The target system may also be a virtual machine, which executes byte code. We use the term byte code to denote a set of platform independent assembly instructions that are primarily intended to be interpreted by virtual machines. Due to their slow interpretation times, virtual machines are generally not suitable for systems that should have a prompt and time-determined response.

The tracking of model changes presents an important research topic of practical relevance to the Model-Driven Development (MDD) community. In [29], the authors introduce new features of the MetaEdit+ Workbench [30] and present various capabilities for visualizing language concepts of a DSL, including dynamic modification of appearance properties. The MetaEdit+ Workbench is a tool that provides support for various development phases including meta-modeling, modeling, code generation, and simulation of the modeled system. In our approach, we borrow two well-established ideas that are implemented in modern database management systems: transactions and views.

In [27], the authors report the lack of support for model debugging in DSL tools. While most GPL Integrated Development Environments (IDEs) support model debugging because language syntax and semantics are known in advance (and because there is a compiler), the situation concerning DSLs is substantially more complex. The standard debugging scenario is conceptually restricted by operating systems, target frameworks, and libraries. Therefore, any pragmatic approach featuring even minor improvements related to MeMID activities is going to represent a significant contribution to the testing of domain-specific models.

3. Action Report as an Extended M2T Transformation

An action report is a special M2T transformation formally defined using a language for specifying code generators that, in addition to the description of the model-to-text transformation, contains commands and rules for command invocations during model execution. DSM involves use of reports, also known as generators, to specify how to utilize information from abstract models and to generate code in accordance with a particular concrete syntax [3], [14], [20], [30]. A report is a program whose interpretation yields a textual representation of the semantics expressed in a model. Since transformation languages support model filtering by selection of objects and relations according to a criterion, they should be used to explicitly define a submodel or model view. The need to introduce submodels arises from the fact that, in practice, testing is most of the time focused on a single part of the system and not on the system as a whole.

The purpose of extending report languages and their interpreters is to improve synchronization between a modeling tool, target interpreter and client applications that are not generated by the modeling tool. Therefore, an action report is a report containing synchronization commands. Accordingly, an action report interpreter is an extended code generator that, in addition to reading, may change the state of a model, meta-model, client application and target interpreter. Put in simple terms, an action report features set and get operations for property values. In such role of action reports, it is assumed that every participant in the synchronization has an instance of the action report interpreter.

Relevant characteristics of action reports are divided into three groups: (i) those that are related to modeling tools; (ii) those that are related to target interpreters; and (iii) those that are related to user components for visualizing and documenting actions.

The first group includes the following characteristics: (i) action reports are defined in the context of a submodel; (ii) action reports allow frequent model view changes, i.e., frequent submodel redefinitions; (iii) action reports are executed inside an optimized transaction whose beginning and end are tied to valid model states; and (iv) action reports may execute operations (and be referenced) in the context of both concepts forming a meta-model (modeling language) and objects not part of the meta-model, i.e., any user control.

The second group includes the following characteristics: (i) there are target environments that support model interpretation during specification time, which introduces the need for an operation that would calculate specification increment between two model states; and (ii) when employing models to manage business processes, action reports may be used to synchronize business activities prior to a switch to a new management model, as well as to incrementally generate documentation and applications that precede the change of the business model.

The third group includes the following characteristics: (i) all the communication between modeling tools and external applications is in the form of textual commands specified in the syntax of a generator language; (ii) action reports are closely related to target interpreter environments, which may vary greatly; (iii) action reports may be called both synchronously and asynchronously, while calling rules define order, frequency, and/or logical conditions related to the call; and (iv) if the target interpreter does not support incremental update during interpretation time, the problem is reduced to the recompilation of the generated code and the use of appropriate debugging tools, which are often part of IDEs.

The role of action reports is illustrated in Fig. 2. They are primarily an interface between the modeling tool, user applications, and target interpreter or debugging environment for the generated code. The interpretation of action reports is performed by special components that are instances of action report interpreters, which are labeled *AR Int* within the little yellow rectangles featured in Fig. 2. The objective is to allow various user groups like meta-modelers, modelers, testers, etc., to use an existing DSM tool as a means of testing the generated code, target interpreter, model and DSL. Action reports are not intended to be used for the description of dynamic characteristics of a system. These characteristics may be completely formally specified through UML state diagrams or equivalent DSLs. Action reports are employed to allow direct use of the existing DSM graphical interface in debugging or testing of the generated code.

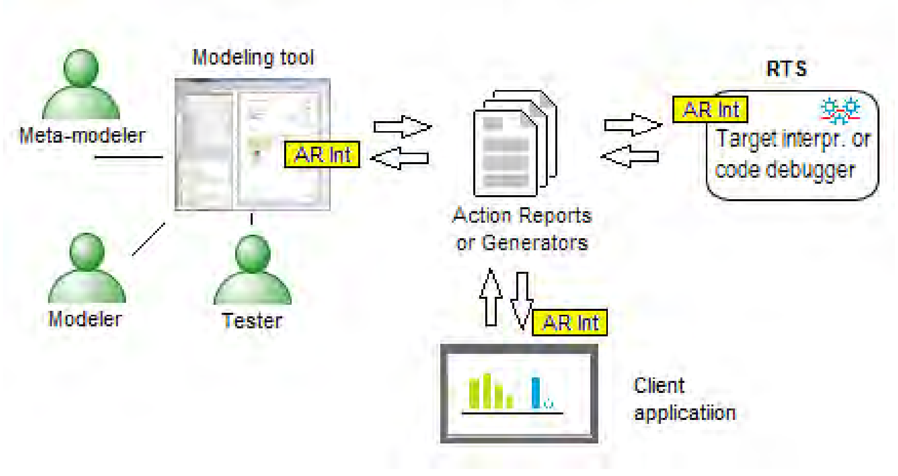


Fig. 2. Action reports and their interpreters

When the modeling language is not sufficiently semantically rich, generators may be temporarily used to describe semantics, i.e., surpass problems caused by the lack of DSL concepts. This scenario is typical particularly for the DSL construction phase.

We close the action reports introductory section with a remark that the importance of action reports as defined herein may significantly differ depending on the actual context. In some business domains, the feedback that action reports may provide to modeling tools has no relevance. However, when DSLs are used in specification of measurement and control processes, action reports are essential and their use brings numerous advantages [29]. A modeling tool may be used as an HMI by exploiting the feedback from the target interpreter. There may also be different visual representations of a single language concept.

4. M2A , A2M, and M2D Transformations

For the purpose of investigating and verifying practical usability of Model-to-Application, Application-to-Model, and Model-to-Document transformations, we implemented the DVRepLang language for specifying these transformations and a corresponding interpreter [8], [14]. They are part of DVDocIDE [10], a DSM tool for document modeling. M2A/A2M transformations are basically M2T/Text-to-Model (T2M) transformations whose purpose has been described in various papers [30], [34]. M2T transformations have been applied in numerous tools for code generation from models [2], [14], [15], [20]. The motivation for introducing M2A/A2M transformations in our research is differentiating in code generation between: (i) procedures that generate the code for the communication between modeling tools and a target interpreter and (ii) procedures that generate the code to be interpreted or executed on the target interpreter. The procedures that generate the code responsible for the communication are tailored to the characteristics of communication components, i.e., communication frameworks. On the other hand, the procedures that generate the code being interpreted are tailored to the characteristics of the framework and target system. The semantics expressed by the model is interpreted by this target system independently from the manner in which the communication is performed. For example, if both frameworks are inadequate, the communication procedures may generate TCP/IP commands, while the procedures responsible for expressing the semantics of the model may generate code in C++. In this context, the target interpreter is important as a component that verifies model and gives feedback for the potential refinement of both the model and DSL. The reason for introducing the notion of a M2D transformation is a need to extend M2T transformations with procedures for the generation of documentation about the MeMID activities.

The most important characteristics of M2A/A2M transformations include:

- target text is a code in a GPL, DSL, or any textual format interpretable by a modeling tool or a target interpreter;
- target text contains embedded semantic actions like property get and set operations;
- operations may be performed on models inside a repository or locally on visual representations of DSL concepts in the graphical interface of a modeling tool;
- these transformations may include operations on external elements of the presentation that are not part of the modeling tool (see Fig. 3);
- these transformations do not directly modify the meta-model, but are used for the semi-automatic inclusion of user controls that graphically represent language concepts; and
- when there is a discrepancy between the concepts directly supported by the interpreter and those of the DSL, these transformations provide an interface for the communication between the relatively incompatible units.

The most important characteristics of M2D transformations include:

- target text is a specification of document instances in a DSL;
- such specification contains identifiers of layout styles that are used for the document rendering;
- target interpreter, which features an instance of the action report interpreter, utilizes action report definitions as a basis for the identification of rules and conditions for initiating document rendering; and
- M2D transformations include rendering of well-designed documents in the PDF or HTML format in the form of external services.

By introducing these transformations, we satisfy some of the user requirements related to the more agile testing and documenting of DSLs, models, and target interpreters. The ideal environment for the application of these transformations within the MeMID activities is the one that supposes the existence of the “universal interpreter” and does not require interrupting the interpretation during the synchronization of model changes. These “hot” switches to a new version of the model are known as incremental updates. Universal interpreters that are independent of the application domain do not exist. Any generalization of the target interpreter necessarily leads to a greater separation of the language used to describe the problem from the language interpretable by the interpreter. In practice, there is a compromise to solve the widest possible class of problems by upgrading the interpreter so that it could internally translate DSL constructs that are at a high level of abstraction to an optimized set of elementary operations.

With respect to the connectedness of meta-models and models, modern tools vary greatly. Some tools support meta-modeling only through textual syntax and feature weak synchronization between meta-models and models [15]. Other tools consistently support abstract graphical models, graphical DSL constructions, and different visual representations for the same language concept, as well as full synchronization between the meta-models and models [30]. Different visual representations of a single language concept allow animations, i.e., visual presentations of model states during interpretation [29]. The debugging of DSM models cannot be equated with the debugging inside GPL IDEs. With the GPL-to-assembly transformations, there is a finite, predetermined set of source and target language concepts. On the other hand, in DSM neither the source nor the target language needs to be known in advance. The source language is constructed to meet the domain-specific needs and the target code may substantially depend on the existing libraries and frameworks. One of the approaches to the formation of a stronger logical relationship between debugging environments and modeling tools includes the use of patterns. In this manner, it is generally possible to relate the model to the target code. One disadvantage of the use of patterns is that they need to be created for each combination of a DSL and target platform. The critical issue is how efficient the debugging of the resulting code is when done through a GPL IDE that is logically separated from the meta-modeling tool. This problem is extensively debated and the proving of the language validity is a topic of numerous papers and books [21], [27].

Further discussion of MeMID activities is based upon an assumption that the debugging rules or steps should be defined inside the M2A, A2M and M2D transformations in order to provide the feedback from the target interpreter toward the model.

5. Using Submodels, Transactions, and Action Reports in MeMID activities

Modeling tools usually support the concept of model decomposition, which implies that an object, relation, or role may be linked to a submodel. This allows for a model to be described and expressed at different levels of granularity and sometimes even at different levels of abstraction. During testing, it is necessary to focus on just a subset of elements within the model. In DSM tools, this subset should be defined using a submodel, as a complex object with its own structure, operations, and constraints. Although default operations (insert, delete, connect, and disconnect) and constraints express fundamental dynamics of the system described by that model, they are not sufficient to express the rules for the translation of the model from one consistent state to another. For this reason, modeling tools should include support for the transaction concept. Transaction is defined as an operation that validates a sequence of actions on a model and updates the repository. Similar to the database transaction, it includes a validation of actions in the context of MeMID activities. Therefore, we expect that modeling tools explicitly support defining submodels, similarly to how it is supported in DVDocIDE [10].

The purpose of submodels and transactions is illustrated by an example presented in Fig. 3 The diagram in the left section of the figure features activities *A1-A4* that are part of the production of advertisements and related documents. The activity *A2* (Standard ad production) is composite and consists of several activities in the modeling of small advertisements. To model advertisements, we use a DSL named DVAdLang, [5], [11]. The subgraph of the object *A2*, marked with *M4*, is an advertisement model that features a logo, several phone numbers, and an email address. In the upper right section of the figure, there are three models (*M1*-*M3*) in three consistent states (*S1*-*S3*), all of them representing the same advertisement. These advertisements states, which are explicitly expressed by their models *M1*-*M3*, are evaluated in the context of the submodel *SM1*, which does not contain the advertisement title (the yellow rounded rectangle).

With respect to model execution, there are two levels of verification: (i) model verification during design time, done by the modeling tool and in accordance with the meta-model; and (ii) on-demand verification of the code generated from the model, whose form of invocation is explicitly expressed by transactions, i.e., action reports in a M2A transformation (in Fig. 3 marked by *T1* and *T2*). Successfully completed transactions change the advertisement states while giving a visual representation for each of these states, i.e., they document the changes in the advertisement states using well-designed PDF documents (see the lower section of Fig. 3). Partial verification of a model, herein illustrated by the example of the submodel *SM1*, which is represented by a shaded rectangle with rounded edges, is not directly supported in standard DSM tools. This fact hinders a wider use of DSM tools in certain domains, such as document engineering and incremental specification of measurement and control processes. In the presented example, we implemented this functionality using the incremental document generator DVDocGen [6] as the target interpreter. In this manner, we obtained advertisement images, which are shown in the lower section of Fig. 3. DVDocGen can detect, interpret, and update action reports. The DSM modeling tool needs to interpret only a property value set operation in order to visualize the model execution flow. As opposed to DVDocIDE [10], which is focused on the formal specification of documents, general purpose DSM tools mostly do not support such operations.

Examples 1 and 2 further refer to the contents of Fig. 3 and include: (i) specification of the action report *AR1*, which sets the text property *Font.Underline* in the objects in the modeling tool; and (ii) a generic form of a DSL script, which is an interpretable textual representation of a portion or whole semantics expressed by a model.

Example 1. The action report *AR1* is defined using DVRepLang [8], [38], a language similar to the MetaEdit+ Reporting Language (MERL) [30]. Both languages are navigation languages for M2T transformations of models into an arbitrary target text. *AR1*, which is presented in Listing 1, is applicable to all models that are of the same type as *M1*-*M4* from Fig. 3. It is used to generate, in accordance with the syntax of DVAdLang language, a DSL script from the advertisements models. Besides the code segments that are responsible for a standard M2T transformation, *AR1* also contains sections for embedded semantic actions.

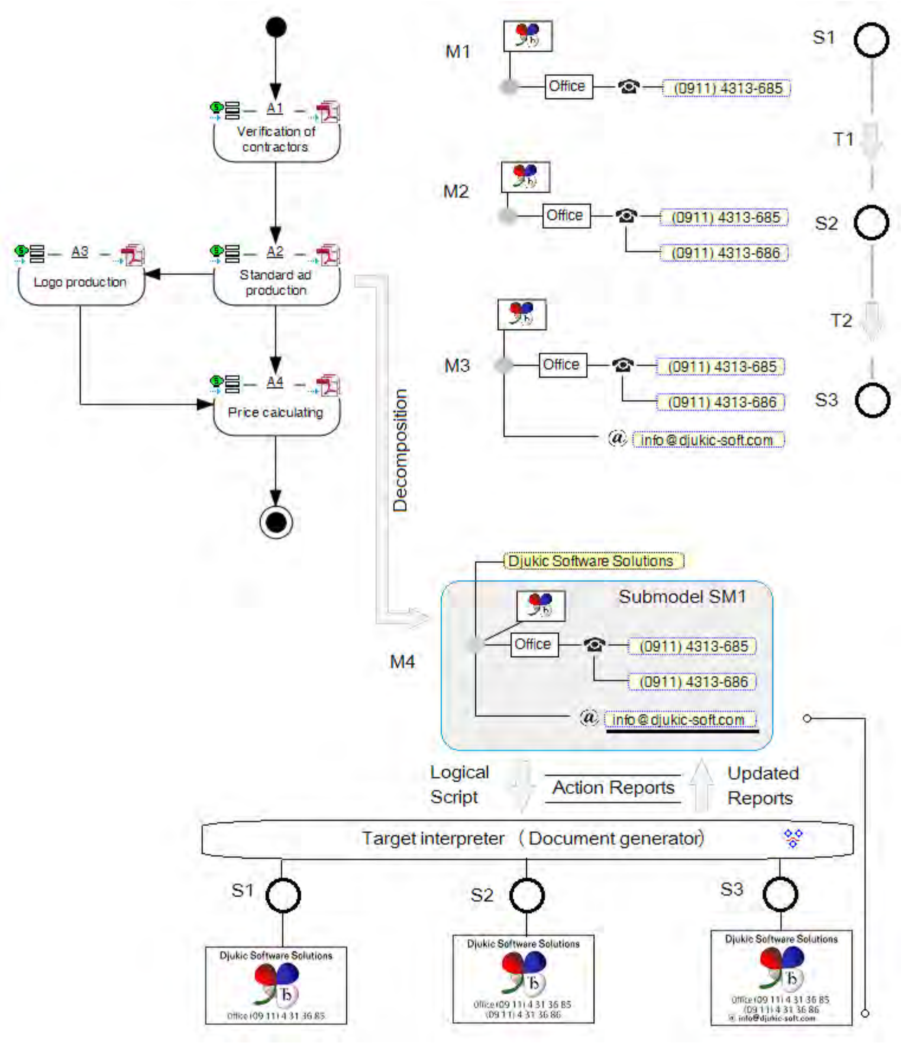


Fig. 3. Submodels, transactions, and testing of models and the target interpreter

Listing 1. Action report example

```
Report 'AR1'
CALL_TYPE = event; /*interval,cyclic,event*/
foreach >ContentUnit {
do .()
{'<'type '>'
if type = 'LOGO' then
   ID ',' :Alignment; ',' :Height;
else :Value; endif
newline
dowhile ~Phones in> Phone connections~Phone rings in.()
```

```
   {
      '<' type '>' :Value; newline
      ACTION_BEGIN
      '<STATE>'objID
      :Font.Underline=true;
      ACTION_END
   }
}
```

The existing syntax of DVRepLang, which is used for M2T transformations, is extended with: (i) `CALL_TYPE` command for the declaration of conditions or intervals for the exchange of action reports with the target interpreter, and (ii) `ACTION_BEGIN` and `ACTION_END` primitives, which mark a report code section related to synchronization. In Listing 1, the new language commands are marked in bold.

Example 2. During the interpretation of the *AR1* report from Example 1, a DSM tool generates target text. In this particular case, it is a DSL script in the DVAdLang syntax, which is featured in Listing 2. The definition of action reports is inserted into the `<AR_META>` tag. This definition is required by the target interpreter during the whole synchronization process done with the modeling tool and client applications.

Listing 2. Embedded definition of an action report in the DSL script

```
<AR_META>="REPORT AR1..."
<CU>Initial DSL script
<STATE>S1
<CU>Increment for S2 (Transaction T1)
<STATE>S2
<CU>Increment for S3 (Transaction T2)
<STATE>S3
```

The `<STATE>objID` commands in a DSL script in the target language explicitly denote states, and define transitions and semantic action during model execution. During the interpretation of each `<STATE>` command, a client application or document generator finds an action definition within the `<AR_META>` tag and executes that action while informing the modeling tool about the interpretation state. In this example, the property-setting operation `Font.Underline=true` (marked by `ACTION_BEGIN` and `ACTION_END`) is called.

Semantic action of synchronization through an action report may be arbitrarily complex. It may include incremental specification and rendering of documents inside MeMID activities. In this particular example, since the target interpreter is a document renderer, the semantic action represents both a proof of model execution and a rendered documentation about model testing. For the visualization of the execution of document models and business process models, very fast document generators are required [4]. An example of one such simulation that follows the life cycle of documents is presented in a video clip [5].

6. User Application and Modeling Tool

In a typical DSM scenario, HMI components of a user application are generated or parameterized from models. User applications are not utilized in modeling but are products of modeling that are obtained in the automatic generation of source code. In environments where DSM is being applied, users often have their own framework and HMI components whose layout and functionality are too complex to be specified using editors for meta-modeling. Therefore, it is useful to allow simple integration and use of external HMI components in DSM tools. This integration does not only include exchange of values according to the scenario described in the previous section, but also implies use of external HMI components for visual representation of abstract DSL concepts. In the following discussion, we restrict ourselves to the pragmatic approach that utilizes action reports and common properties of visualization elements in the DSM tool and HMI components.

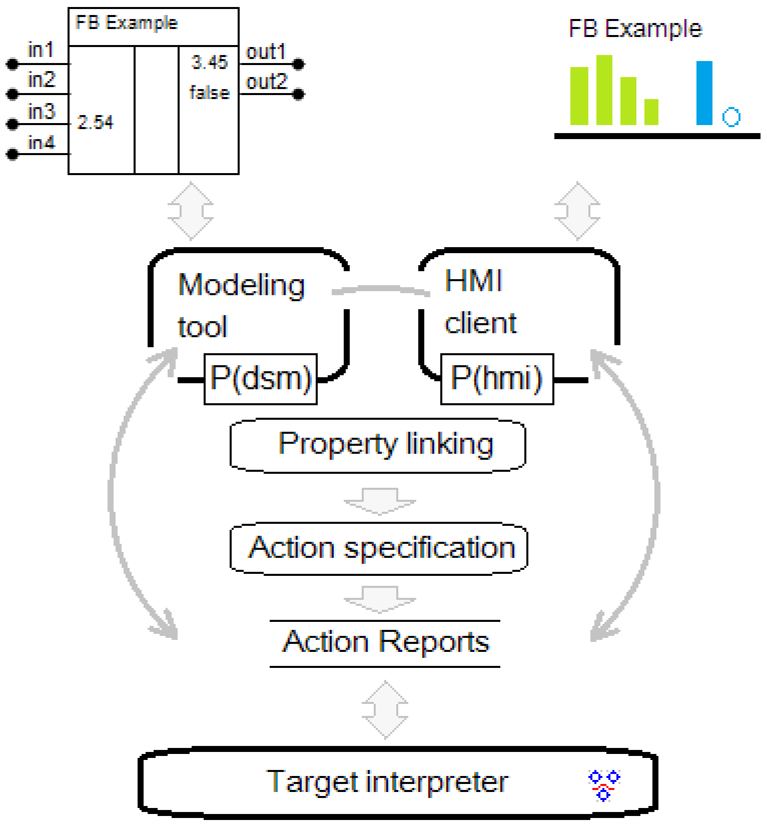


Fig. 4. Editor of common properties, action specifications, and synchronization

In Fig. 4, we illustrate an approach to the integration of user HMI components into DSM tools. In the upper left corner of Fig. 4, there is a function block object in a default visual representation created using a DSM tool. In the upper right corner of the same figure, there is a user HMI component that in the form similar to a bar chart shows input and output values of variables associated with the function block. The output variable *out2* is of the bool type, so it is represented in the HMI component as an empty circle when its value is *false*, or as a filled circle when its value is *true*. Both the DSM tool and the HMI component support reading and changing the property values in several ways, e.g., mouse operations and using a text editor. The *P(dsm)* label denotes properties defined using the DSM tool, while the *P(hmi)* label denotes properties belonging to the HMI component. The integration procedure consists of three steps: (i) property linking (also shown in Fig. 4), in which the semantically equivalent properties are found between the two visual representations, irrespectively of the actual form of visualization; (ii) defining user actions on the elements of the graphical representation when certain semantic actions should be executed (labeled *Action specification* in Fig. 4); and (iii) defining the semantics of actions using a language for action reports.

The target interpreter, which is shown in the lower section of Fig. 4, executes the current specification, i.e., interprets the model and action reports. In the context of the target interpreter, it is not important whether the action reports were created by a DSM tool or user application. The role of the target interpreter is to fetch the values of some properties from the current state of the interpretation, update the action report, and send it back. The communication may also go in the opposite direction. Based on the state of the real system, the target interpreter detects the conditions when the semantic actions, whose structure and content are represented by the previously defined action reports, should be called. In this manner, the state of the model within the DSM tool or the state of the user application may be updated. Modifications in the model are not restricted only to setting new values of some properties, but they may be arbitrarily complex and include any operation that is supported within the graphical interface of the DSM tool, HMI components, and user application containing those HMI components.

In the context of the example from Fig. 4, Listing 3 illustrates what is executed by the action report interpreter featured in the target interpreter.

Listing 3. Structure of the semantic action for synchronization

```
ACTION_BEGIN
:in3='2.54'
ACTION_END
```

The value of the *in3* property is set to *2.54* and the updated action report is sent back to: (i) the modeling tool for the purpose of modifying interface properties and (ii) the HMI client application for the purpose of setting the values for visualization controls. Report exchange is performed periodically or on a certain event that is not time dependent, according to the role of an external HMI component. This approach to the synchronization between the HMI components and target interpreters is not supported within the general purpose DSM tools, so the testing is performed using DVRepLang and DVDocIDE, which are DSM tools for document engineering.

7. DSM and Action Reports vs. UML in the Domain of Measurement and Control Systems

Software models are widely used in the manufacturing of measurement and control systems (MCSs), as well as in processes that are automated by these systems. In the field of MCS, there are numerous specifications and solutions that were created in previous decades without significant use of standardized modeling languages. There are several important reasons why UML has not become widely adopted in the MSC industry:

- UML is a graphical language that is not intuitive for domain-specific problems;
- there is a discrepancy between abstract models and a target language that is used in model implementation;
- UML cannot be used to easily transform submodels of abstract specifications into various target languages; and
- UML tools offer limited possibilities when it comes to model execution.

Some of the aforementioned restrictions, which used to impede the full-fledged application of UML in the MCS industry, have been overcome, however many practical issues still remain. MSC solutions have to satisfy rigorous requirements related to low system resources consumption, precision, execution speed, and reliability of control programs. Application of abstract UML models was not attractive to domain experts in spite of potential benefits that could be expected in software development from such an approach. Practical experience of domain experts shows that the gap between an ontology and the linguistic concepts of UML that describe the meaning increases with the specialization of a production environment.

DSM languages and tools have become more prominent as a result of trying to avoid numerous issues that arise from using GPLs to model domain-specific problems. The goals of DSM are to completely formally describe a data structure and process using domain-specific concepts and to generate code from abstract models while using all the capabilities of a target environment. One particularly beneficial effect of using DSM tools, especially those that support access to their repositories through a web service, could be a move from domain-specific to domain modeling. This means that, in some business domains, a problem solution based on DSM may be made available to users from similar domains by offering: (i) a set of domain specific languages for modeling different aspects of a system; (ii) libraries containing abstract model transformations for various target environments (concrete programming languages, interpreters, and hardware languages);

(iii) a predefined set of constraints for different contexts of use; and (iv) concepts for describing model variations and the customization of services to a concrete environment that are both formal and simple for users.

7.1. Applying Action Reports to Models of Car Control Systems

The example given below illustrates the application of action reports in the synchronization of complex services and actions in a simplified version of a car control system. The DSL that is featured in Fig. 5 was constructed starting from the Real-time Object-oriented Modeling Language (ROOM) [35], whose numerous variations are used in the automotive industry. The basic concepts of this language include objects (Actor, External client port, External server port, and Switch) and relations (Binding and Visualization). These language concepts are sufficient for describing driver's interaction with car devices, command processing, state indications on a display, and the feedback between the current car speed and the way the system reacts on driver's commands and states of different sensors.

The model shows a collection of external client ports, such as gas pedal, brake pedal, rotation counter, engine thermometer, and fuel state indicator. These mostly analogue devices are connected through sensors to controllers or external server ports, from which measured values are forwarded to display components (for speed, rotation, temperature, and fuel level). Switches that turn engine and cruise control (tempo limiter) on and off are connected to gas and speed controllers. This abstract model of a car control system has two units. The first unit includes objects that read values and forward them to controllers. The other unit contains objects that are used to display values. In the development of car control systems, a practitioner would have the following expectations from DSM:

- to be able to extend the language and graphical representations of concepts (meta-modeling);
- to be able to describe any complex control system using diagrams and to test such models (modeling);
- to connect a model to analogue devices, external applications, or HMI components that support advanced graphics;
- to generate code for different target systems and controllers; and
- to automatically document each test case in a readable format (PDF).

Such expectations are well founded because across different industries there are many software solutions that satisfy the majority of these requirements to some extent. At the moment, connecting to external applications, and documenting of test cases are areas that still need significant improvement. This example is generally focused on illustrating the use of HMI components with the advanced Windows Presentation Form (WPF) graphics [39]. The advertisement example featured in Section 5 illustrates how documents are generated during the testing of models.

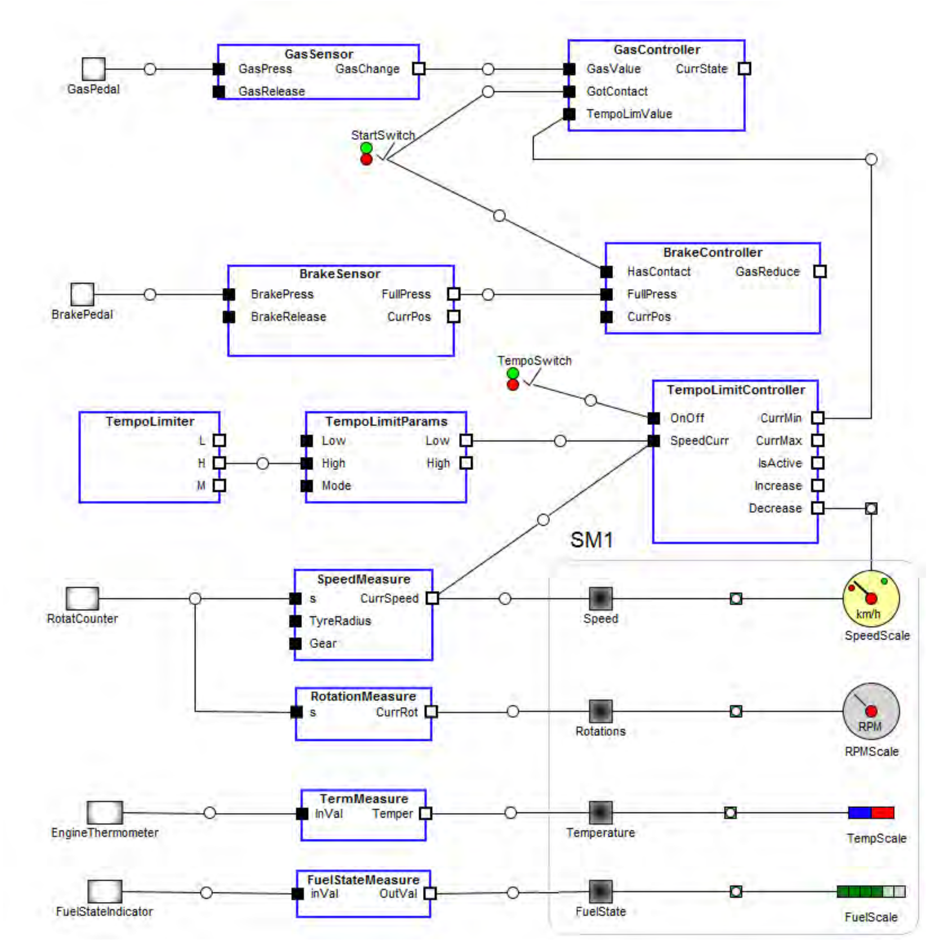


Fig. 5. Car control system as specified in a DSL

HMI components or user applications are connected to a model in two ways (see Fig. 6). In the first scenario, HMI instances are generated from models, while some of the properties are set according to the model state. In this case, graphical components are implemented using WPF. In the second scenario, HMI components are default visual representations of linguistic concepts that are used for modeling. In both cases, linking of model elements and visual representations is based on property linking (see Fig. 4) and using action reports. All external server ports that correspond to different types of scales, such as speed, rotations, temperature, and fuel state featured in Fig. 5, are implemented as web services. These services are used to retrieve the latest state and forward a new value. All scales that are located to the right side of the SM1 submodel are implemented using the WPF components. In the existing DSM tools, the aforementioned functionality dedicated to connecting DSM and HMI components may be achieved only indirectly, because these tools do not include an implementation of action report interpreters. The indirect method involves using APIs to access the repository of DSM tools with the goal of creating objects and setting property values.

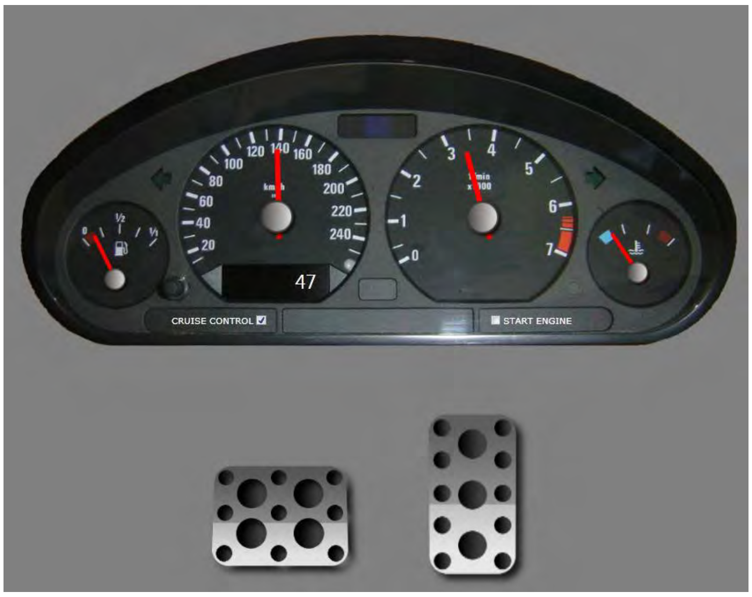


Fig. 6. HMI components as created in WPF

In Listing 4, we present a code generator for the model featured in Fig. 5. It is a MERL report that generates code for web service calls.

Listing 4. MERL report that generates web service calls

```
Report 'External Server Ports'
$mUrl = :VusualURL;
foreach .External Server Port;
{
   filename :CodeTargetFolder;1 :Name; '.h'  write
   '#ifndef C_' :Name;'_HEADER_H_' newline
   '#define C_' :Name;'_HEADER_H_' newline
   newline
   '#include "GenericServerPort.h"
class C' :Name; ' : CGenericServerPort' newline
   '{
public:'
   newline
   ' C' :Name; '(int mCurrVal) : CGenericServerPort(currVal)
   {
      //TODO: ???
   }' newline
   ' virtual~C' :Name; '(void) {
   }'
   newline
   do ~ValueOnPort;~UsedFor;.()
   {
      ' int Get' type '()
   {
   ';
      ' String mUrl = "' $mUrl 'Get' type '";
   }'
      newline
   }
   ' void On' :Name; 'Update(int currVal)
   {'
   newline
   do ~ValueOnPort;~UsedFor;.()
   {
      '   String mUrl = "' $mUrl 'Set' type
      do :()
      {
         '?' type'=m_'type;
      }
      '";'
      newline
   }
   ' };'
   do ~Server~Server.()
   {
      if :IsSensor;='T' then
   newline ' C' :Name; '& m_' :Name; ';'
      endif
   }
   newline

   '#endif'
   newline
   close
}
endreport
```

From the model, we generate web service addresses and HTTP GET requests that read and set the current speed. An excerpt from the code that was generated using the aforementioned report is presented in Listing 5.

Listing 5. An excerpt from the generated code for calling web services

```
#ifndef C_Speed_HEADER_H_
#define C_Speed_HEADER_H_
#include "GenericServerPort.h"
class CSpeed : CGenericServerPort
{
public:
```

```
CSpeed(int mCurrVal) :
CGenericServerPort(currVal)
{
   //TODO: ???
}
  virtual~CSpeed(void) {
  }
int GetSpeedScale()
{
   String mUrl = "http://localhost:13216/
   CarDashWebService.asmx/GetSpeedScale";
}
void OnSpeedUpdate(int currVal)
{
   String mUrl ="http://localhost:13216/
   CarDashWebService.asmx/SetSpeedScale?
   ScaleName=m_ScaleName?MinValue=m_MinValue?
   MaxValue=m_MaxValue?Precision=m_Precision?
   CurrValue=m_CurrValue";
};
  CSpeedMeasure& m_SpeedMeasure;
#endif
```

7.2. Applying Action Reports to Function Block Diagrams

In this subsection, we present another practical example that highlights our experience in the application of GPLs and DSLs in measurement and control systems. The example involves using DSM tools to construct and apply a graphical language for the description of function block diagrams according to the IEC 611.31 specification [18].

The IEC 611.31 specification features five parts, two of which, structured text and function block diagrams, are especially important in the subsequent discussion. Structured text (ST) is a textual GPL with a syntax similar to that of Pascal and with features similar to those of C++, but containing certain language concepts that provide some benefits when applied to MCSs. A function block diagram (FBD) is a graphical GPL that may be used to specify flows in measurement and control processes by diagrams. In practice, numerous tools for specifying FBDs (modeling MCSs using FBDs) are used. A common characteristic of ST and FBD languages is the fact that the syntax is fixed in advance. For that reason, in most tools, algorithms for generating code from the model are hard-coded. The main shortcoming of tools for modeling using FBD is the fact that domain-specific problems are modeled using general purpose language concepts that are often not compatible with the models in real systems. For modeling activities, experienced IEC 611.31 programmers and companies are often hired, however, their productivity in actual projects cannot be readily predicted. In order to point out possible solutions to the aforementioned problems, in the provided example we applied the DSM approach which includes the following activities:

– applying DSM tools in the construction of a IEC 611.31 language,
– specifying code generators and action reports using a M2T transformation language,
– generating ST and native code from models; and
– interpreting models where incremental updating is supported.

For the construction of the IEC 611.31 graphical GPL, we used the MetaEdit+ modeler. In Fig. 7, there is an example of FBD, which is further used to explain main concepts of the language. The language features objects of the following types: function block (1), type convertor (2), distributor (3), input and output connectors (4), and connectors of logical pages (5). Function block (FB) has three subtypes: built-in FB (1.1), intrinsic FB (1.2), and external FB (1.3). Each function block has ports through which it exchanges input and output values with other objects. In the process of language construction, we defined several variants of concrete graphical syntax, model constraints, and diagnostics for incorrect operations and inconsistent model states. We selected the textual IEC 611.31 (ST) and Abstract Syntax Tree (AST) to be our target languages. In line with the example from the introduction (Fig. 1), our intention was to generate GPL specifications in the IEC 611.31 ST syntax from model, together with native code for Intel and ARM processors that is optimized for the target domain, by using AST as input structure for native code generation. Since in both cases a target interpreter is required to execute a model, for that purpose we used a special RTS that executes segments of native code. As native code generation is closely related to compiler construction, to this end, we relied on various industry and academic solutions and experiences.

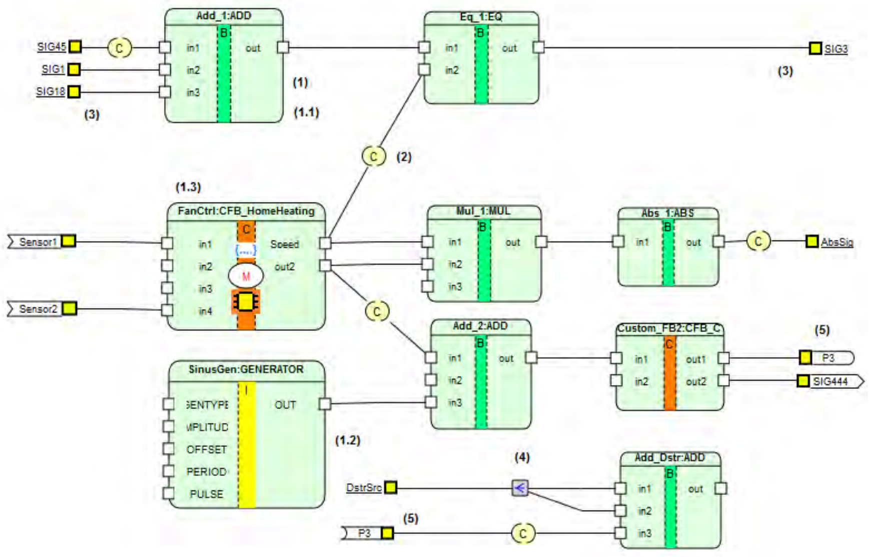


Fig. 7. A FBD example in IEC 611.31

In Listings 6 and 7, we give short excerpts from the generator of ST code, as well as the end result related to the model in Fig. 7. Generators were written in MERL. The ST code generator iterates through all Custom FBs and checks whether they are macros. In the case they are macros, it calls a generator that retrieves the code defined by the macro. In the case they are not macros, by relying on properties, it retrieves definitions of input (`VAR_INPUT` … `END_VAR`) and output (`VAR_OUTPUT` … `END_VAR`) signals, as well as internal variables (`VAR` … `END_VAR`). Whenever a function block is declared as a macro, its graphical representation is changed so that a circled letter M appears in the center of the symbol (see Fig. 7). The body of the Custom FB is retrieved from the `:IEC_StructText;` property.

Listing 6. Excerpt from the generator of ST code

```
report '_IEC_CodeForCustomFB'
foreach .IEC_CustomFB;
{
   if :IEC_IsMacro; = 'T' then
         do decompositions
      {
         subreport '!IEC_STCode' run
         newline
      }
   else
      'FUNCTION_BLOCK ':IEC_CustomFBName; newline
      $p = ''
      do :IEC_Inputs; {$p ='T'}
      if $p = 'T' then
         'VAR_INPUT' newline
         do :IEC_Inputs;
         {
            ' ':IEC_PortName; ':' :IEC_DataType;
            if :IEC_Default; <> '' then
              ' := ' :IEC_Default;
            endif ';'
            newline
         }
         'END_VAR' newline
      endif
      $p = ''
      do :IEC_Outputs; {$p ='T'}
      if $p = 'T' then
         'VAR_OUTPUT' newline
         do :IEC_Outputs;
         {
            ' ':IEC_PortName; ':' :IEC_DataType;
            if :IEC_Default; <> '' then
              ' := ' :IEC_Default;
            endif ';'
            newline
         }
         'END_VAR' newline
      endif

      $p = ''
      do :IEC_LocalVars; {$p ='T'}
      if $p = 'T' then
         'VAR' newline
         do :IEC_LocalVars;
         {
            ' ':IEC_PortName; ':' :IEC_DataType;
            if :IEC_Default; <> '' then
              ' := ' :IEC_Default;
            endif ';'
            newline
         }
         'END_VAR' newline
      endif
      :IEC_StructText; newline
   endif
   if :IEC_IsMacro; = 'F' then
   'END_FUNCTION_BLOCK' newline newline
   endif
}
endreport
```

The resulting ST code is produced by calling the generator, which translates the whole model and associated submodels. Generation of Custom FBs is only one segment of the translation process. In the generated code, after the `PROGRAM` keyword, there is the name of the model featured in Fig. 7, followed by the definitions of all the input and output ports or signals. Input and output signals are translated into input and output variables of the corresponding types, while external signals are translated into external variables. At the end of the code excerpt, there is the body of the ST program, which contains a description of the relations defined by the model. The code in the line `Add_1_out := ADD(INT_TO_UDINT(SIG45), SIG1, SIG18);` indicates that the *out* port of the FB instance *Add_1* is modified by adding *SIG45*, *SIG1*, and *SIG18*, where *SIG45* was previously converted from INT to DINT.

Listing 7. Generated ST code

```
PROGRAM Example_with_all_language_concepts
VAR_INPUT
   DstrSrc:INT;
   SIG1:UDINT := 7;
   SIG18:UDINT := 21;
   SIG45:INT := 10;
END_VAR
VAR_OUTPUT
   AbsSig:USINT;
   SIG3:BOOL;
END_VAR
VAR_EXTERNAL
   SIG444:REAL;
   Sensor1:INT;
   Sensor2:INT;
END_VAR
VAR
   Abs_1_out :INT;
   Add_1_out :UDINT;
   Add_2_out :REAL;
   Add_Dstr_out :INT;
   Eq_1_out :BOOL;
   Mul_1_out :INT;
   SinusGen:GENERATOR;
   Custom_FB2:CFB_Commands;
   FanCtrl:CFB_HomeHeating;
END_VAR

   Add_1_out := ADD(INT_TO_UDINT(SIG45), SIG1, SIG18);
   Eq_1_out := EQ(Add_1_out, INT_TO_UDINT(FanCtrl.Speed));
   SinusGen(1, 1.0, 5.0, 10.0, 2.0);
   Add_2_out := ADD(INT_TO_REAL(FanCtrl.out2), 55.9,
SinusGen.OUT);
   Custom_FB2(Add_2_out, 46.0);
   FanCtrl(Sensor1, 9, 10, Sensor2);
   Mul_1_out := MUL(FanCtrl.Speed, FanCtrl.out2, 40);
   Abs_1_out := ABS(Mul_1_out);
   AbsSig := INT_TO_USINT(Abs_1_out);
   SIG3 := Eq_1_out;
   Add_Dstr_out := ADD(DstrSrc, DstrSrc, REAL_TO_INT
(Custom_FB2.out1));
   SIG444 := Custom_FB2.out2;

END_PROGRAM
```

By constructing the language and using the IEC 611.31 ST generator, we have achieved two important goals that can be accomplished neither by modeling tools that focus only on FBDs nor by UML tools. The first goal was to construct a language that could be easily transformed into a DSL in order to satisfy some domain-specific requirements. The second goal was to transform abstract models into an arbitrary target language, as well as into native code, For some FBs, it is possible to generate code according to some syntax, e.g., to that of VHDL, that would initialize wired-logic controllers. In Fig. 7, such a FB is shown with a processor symbol in the middle. Submodels of a model are transformed into even more different languages. Since DSM tools do not support explicit declaration of a submodel, we achieved this by introducing the IsWired property to FBs and writing a generator that utilizes that property.

From the user's point of view, in addition to fast and complete specification of a modeling language, it is also very important how models are verified.

Numerous tools support model verification but only for complete specifications. Our approach is based on the following idea: each specification, from an empty model to the most complex specification, should be interpreted simultaneously with the modeling process. We refer to such model execution as the interpretation with incremental updating. Similar approaches may be found within simulation tools, such as Simulink [36] or LabView [23]. However, in those cases, the semantics of a modeling language is fixed in advance, which significantly simplifies the whole process. Because of the restrictions associated with language construction, model execution using these tools cannot be considered as a full-fledged MeMID activity.

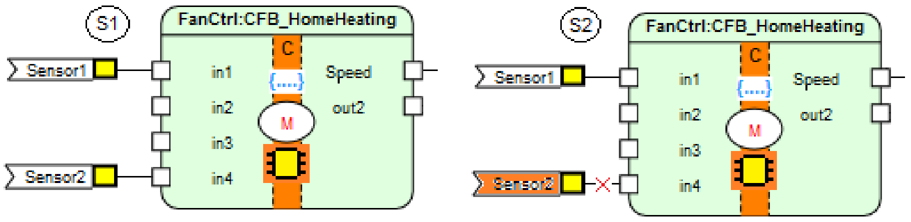


Fig. 8. Incremental update of a MCS

In the rest of this section, we present a practical example of using incremental updating and action generators in a typical MeMID activity. In Fig. 8, two states of a model for fan control, *S1* and *S2*, are depicted as submodels of the model featured in Fig. 7. The state of the model *S1* corresponds to the state of a real system when Sensor 1 (*T1*) is functioning normally. The state of the model *S2* corresponds to the state of the real system when sensor *T1* is being repaired or replaced. This is the case when a problem with rotation speed of a fan may occur due to a thermometer malfunction. In the model, thermometer replacement is defined as a complex transaction that is made of various MeMID activities. It is also possible for an external application that is synchronized with the model or interpreter to display an image which shows that the installation is in progress. Sensor change is recorded in a document that contains information about the location, time, and identifier of the replaced sensor. In order to better understand the example featured in Fig. 8, it may be worth consulting the specification of function block diagrams in accordance with the IEC 611.31 specification [18] and watching a video clip [9] that demonstrates the construction of a DSL and model execution in a target interpreter.

According to the MeMID scenario, a sensor replacement procedure and documenting of the replacement include the following actions:

– An action report that simulates the replacement is executed. It changes the model from state *S1* to state *S2* and sets an appropriate image in a client application.
– An action report that generates a service order in PDF format is executed. All specifications are in various DSLs.

- *Sensor 1* (*T1*) is detached from the function block and a default value that corresponds to the temperature which is measured by some other thermometer is assigned to the input *i1* (`i1=21°C`). The transaction is then confirmed by the model. Using this information, a code update is generated for a target interpreter. This update is only an increment and not a complete program.
- A service person replaces the sensor.
- In the simulator, the model changes to the previous state and checks the functioning of a new sensor (*Sensor 1* is reattached to *i1*).
- The model is connected to the real system and returns to interpreting from the previous state.
- An action report that generates the documentation about the changes in the system during sensor replacement is executed.

Documenting model changes, as a part of the MeMID activity, is partially covered in the example featured in Section 5. When action reports are used in documenting results of the testing of a MCS, they retain a similar structure. They feature nested commands that contain a DSL script or functions which return document content increment.

The aforementioned examples illustrate one advanced scenario of applying DSM tools in specialized production environments. While DSM tools support meta-modeling and modeling well, when it comes to the transformation of submodels to certain target languages, their use in complex MCSs is limited. The main reason is the way how they synchronize with external applications and their poor support for logical connection of actions in a real system to operations on models. General purpose DSM tools are less user-friendly for modeling when compared to specialized CASE tools or applications for modeling measurement and control systems. Efficient use of DSM tools also requires improvement of their graphical interfaces. In the following section, these improvements are described as user operations on models.

8. Action Reports and Operations on Model

DSM tools are usually more advanced in terms of concepts when compared to CASE tools and applications used to model MCSs. On the other hand, dedicated CASE tools and applications have better suited graphical interfaces that support drawing of models considerably closer to the specific standards of a particular business domain. In previous sections, we demonstrated how DSM tools may be improved for the purpose of supporting: (i) model execution and (ii) usage of DSM tools as client applications for monitoring, i.e., surveilance of states in a real system [9]. In this section, we explain how the graphical interface of a DSM tool may be improved for the purpose of its more efficient utilization in specific application domains.

Using action reports for formal specification and implementation of three groups of operations constitutes the basis for the improvement of DSM tools. The first group includes operations that accelerate the construction of a DSL and different visual representations of language concepts in a DSM tool by relying on the existing user HMI components. The second group includes operations used to define the behavior of the graphical interface for basic user operations: insert, delete, connect, disconnect, update, move, etc. The third group includes operations on submodels. With some minor extensions, navigation languages for M2T transformations could support all three groups of operations.

The general structure of reports used to define operations of the first group, i.e., those used to transfer a part of the definition of an external HMI component to a meta-model, is presented in Listing 8. As previously discussed, the DSM tool and user application need to include instances of an action report interpreter capable of interpreting specified actions.

Listing 8. General structure of reports defining operations that transfer definitions of external HMI components to meta-models

```
ACTION_BEGIN
   ObjectDef | RelDef | RoleDef | PropDef
ACTION_END
```

Operations used to define the behavior of the graphical interface should provide expected spatial arrangement of model elements during all kinds of user actions. One method of defining the behavior of a graphical interface is to apply structural patterns in the way that we used them to define document layout. In Listing 9, we present only some of the typical patterns, while a more detailed description of grammar rules and examples may be found in [14]. Each pattern consists of an ordered (OL) or unordered list (UL) of elements, which represent objects and relations in a DSM model. Validation or customization of the model according to the specified patterns is performed during the execution of user operations (insert, delete, connect, etc.). Semantic actions that perform validation according to the patterns are executed using action reports. During this process, rules of spatial layout and structural rules are translated into topological properties of model elements.

Listing 9. Pattern examples

```
PATTERN A UL(B,C,D) END
// The A element consists of three elements, which may appear
in any order.
PATTERN A OL(B,C,D) END
// The A element consists of three elements, which may appear
only in the specified order.
PATTERN A UL(B,C,D) isLeftOf(C,D) END
// The A element consists of three elements, but the C element
must appear before the D element.
```

```
PATTERN A UL(B,C,D) isLeftOf(C,D) isBelow(D,B) END
// The A element consists of three elements, but the C element
must appear to the left of the D element while the D element
must appear above the B element.
PATTERN A UL(B,C[3..5],D) END
// The B element appears exactly once, the C element appears
from three to five times, while the D element appears exactly
once. The elements may appear in any order.
PATTERN A OL(B*,OL(C,D)) END
// The B elements must appear first for any number of times,
followed by the C element and the D element, respectively.
PATTERN A UL(B*,C*,D*) END
// The elements B, C, and D may appear for any number of times
in any order.
```

The third group of operations, whose semantics may be expressed through action reports, is used to: (i) construct submodels and carry out all operations on (sub)models without the need for the execution of low-level API functions on the repository; and (ii) define transactions.

The construction of submodels and corresponding operations is similar to the definition of views in relational databases or the definition of complex objects in object databases. We focus on operations that could significantly improve MeMID activities when the modeling tool is linked to the target interpreter via action reports. Therefore, we give an overview of the selected operation set:

– **CreateSubmodel** (listOfElems) – creates a submodel based on the specified list of objects, connections, relations, roles, and properties from an existing model;

SetCurrentSubm (m_ID) – sets one of the defined submodels as the current one;

DeleteSubmodel (m_ID) – deletes the submodel definition;

AddModel (m_1,m_2) – joins two submodels into one without modifying any relations;

Subtract (m_1,m_2) – removes *m_2* from the existing composite model *m_1*;

Multiply (m_1,n) – creates a new model by repeating the model *m_1 n* times;

Intersection (m_1,m_2) – returns a model containing intersecting element from *m_1* and *m_2*;

Union (m_1,n) – joins two models without repeating elements having same identifiers;

SimDifference (m_1,m_2) – finds a symmetric difference between the two models;

Remove (objType|relType) – removes objects or relations of the specified type from the submodel; and

Clone (objType|relType|roleType) – clones the complete model or just object, relations, roles, and properties of the specified type or matching the specified pattern.

We used DVDocIDE, a DSM tool for document modeling, to test usage of action reports and patterns as means of a more efficient DSM modeling of documents and their templates. We used DVQL [25], a command/query language for documents, to implement operations on submodels. In order to verify usefulness of these operations in general purpose DSM tools, the latter should be considerably extended. This issue is also one of the topics of our future research.

9. Related Work

Over the last few years, Executable UML has been a recurring topic in both the academic and engineering community [32]. Numerous papers and practical solutions extend its usability for simulations and model execution [17], [23], [36]. However, it seems that the transfer of very narrow specialized knowledge to web services (Cloud computing) is advancing more rapidly as opposed to the use of UML tools for the domain-specific problems. In the academic community, much of the model transformation research relies on the OMG's specification Query/View/Transformation (QVT) [28]. The specification consists of three interrelated languages: (i) Relations, (ii) Core and (iii) Operational Mapping. Atlas Transformation Language (ATL) [2] by the Eclipse Foundation [15] is an example of a model-to-model (M2M) transformation language in accordance with the QVT standard. Among the commercial tools, one of the best known transformation languages is MetaEdit+ Reporting Language (MERL) [30]. It is a language mainly focused on model-to-text (M2T) transformations. It partially supports transformations that conduct synchronization between the model, client applications, and target interpreter. By minimally extending MERL to allow specification and interpretation of action reports, it would be possible to synchronize applications that feature disparate user interfaces, and target interpreters or "execution machines" [1], [4], [6], [24], [31], [38].

In [20] and [27], the authors present ideas and solutions for domain-specific model transformations and debugging. Our consideration of code generators differs slightly from the one presented in [20]. We believe that template-based M2T transformations are complex, insufficiently flexible, and complicated to be implemented within the HMI components and target interpreter of models.

In [16], the authors present a translational and an interpretational approach to execution of domain-specific models. These approaches are based on explicit definition of semantics for execution of each model. The translational approach relies on generating code that should be compiled and then executed, while the interpretational approach relies on model interpretation by a target interpreter. The disadvantage of the former approach is that it is unsuitable for simulations and rapid prototyping. On the other hand, the latter approach is considerably more suitable for both rapid prototyping and incremental update of an active system. The authors recognized the necessity of the use of transactions and logging of all model changes for the purpose of backtracking. They resolve the issue of the synchronization between a model and the execution engine by relying on the concurrent access to configuration files used by the DSM editor and execution machine. From their simple example implemented using Eclipse EMF, it seems that the application of their idea is limited to less complex cases. In our approach, which is based on the use of M2T transformations, there are slight extensions of existing navigational languages for M2T transformations and two logically independent execution engines: a report interpreter and a target interpreter of models.

In [37], the authors describe the OMG's approach to standardization of UML model execution, which involves using Action Semantics, i.e., explicit definition of execution rules at the level of the UML meta-model. The goal of this standardization is to allow: (i) software independent specification of actions on UML models; and (ii) execution of UML models. Their approach is based on the following three abstractions: meta-model, execution model (UML model), and actions. The semantics of actions is defined, but not the concrete syntax, because it depends on the target language used in code generation from a model. Because this approach requires knowledge about UML meta-modeling, it seems unlikely that it will be widely applied in domain specific problems, particularly for modeling measurement and control systems.

Among numerous tools for modeling measurement and control system that may be used in the extension of DSM tools, or for better illustration of action reports and use of modeling tool as client applications, the following two stand out: Simulink [36] and IbaLogic [17]. Simulink is a tool primarily aimed at drawing function block diagrams. It features a large library of function blocks that may be customized and supports generation of source code in the C language. In the context of the MeMID activities, Simulink does not adequately support meta-modeling and generation of documentation about model execution. IbaLogic is a tool for modeling measurement and control systems that employs structured text and function block diagrams according to the IEC 611.31 specification, where a function block model is also an execution model. This tool supports linking to various run-time systems that may interpret or execute a model. However, meta-modeling and code generation for different programming languages are not supported. Owing to the featured implementation of a set of basic operations on models, it supports: (i) every version of the incremental update for a target system during interpretation; and (ii) visualization of the state of a real system within the modeling tool.

10. Conclusion

In this paper, we present the first practical results and foundations of an approach aimed at further improvement of DSM tools. Our objective is to better automate the MeMID activities: meta-modeling, modeling, testing of models, generated code, and interpreter, and generation of documentation about test cases. In the areas of document engineering and development of measurement and control systems, the action report approach allows us to specify the following procedures within abstract models: (i) the process of documenting model validation; and (ii) in the context of certain business rules and procedures, the synchronization of actions on a model to the state of the real system. Owing to this, action reports are especially effective when combined with DSM tools that, instead of relying on patterns, conduct M2T transformations by using a dedicated target language and interpreter. In production systems where business procedures are specified both precisely and formally, there is also a need to document each action on the model or to execute each action on the model by relying solely on the previously generated and authorized document. By using action reports, it is possible to synchronize not only the different components that are part of the MeMID activities but also the heterogeneous business and control processes, which feature complex business rules and operation of arbitrary control systems.

Our future research directions include: (i) construction of a language for the description of constraints on presentation elements (graphs), which in turn would simplify the customization of meta-modeling and modeling tools for different domains of application; (ii) construction of M2T transformations, i.e., code generators that would produce binary or assembly code for different processors by starting from abstract models; and (iii) conceptualization of run-time systems that would interpret abstract models, which in turn would be transformed into different target languages, software logic or wired logic. The ultimate goal of our research is to support, to the greatest extent possible, the MeMID scenario, which consists in using modeling tools as client applications to manage business and control processes. The approach presented in this paper was created to be focused on the domain of application and provide pragmatic support to users. For these reasons, its application capabilities may not be fully generic. However, the goal of developing the approach is not primarily oriented to this end, but to provide the foundation for a quality support to users in the domain of monitoring the measurement and control processes. At present, our approach supports modeling and executing models of measurement and control systems. We expect that our ideas, examples, and practical solutions presented in this paper are going to contribute to a better use of DSM tools as client applications for the monitoring of measurement and control processes.

Acknowledgment. This research was supported by Ministry of Education, Science and Technological Development of Republic of Serbia, Grant III-44010: Intelligent Systems for Software Product Development and Business Support based on Models. The authors are grateful to their colleague Tefik Bećirović for help with the testing of action reports using user WPF components and also to Juha-Pekka Tolvanen of the MetaCase company for valuable support in the rapid construction of the IEC 611.31 language using the MetaEdit+ Modeler.

References

1. Apache Software Foundation: FOP. [Online]. Available: http://xmlgraphics. apache.org/fop/0.95/index.html (Accessed: May, 2013)
2. ATL - A Model Transformation Technology. [Online]. Available: http://www.eclipse.org/atl/ (Accessed: May, 2013)
3. Beaudoux, O., Blouin, A.: Using Model Driven Engineering technologies for building authoring applications. Proceedings of ACM Symposium on Document Engineering. (2010)
4. Djukić, V.: DVDoc Renderer Benchmak. [Online]. Available: http://www.dvdocgen.com/Framework/DVDocRenderBench.pdf (Accessed: May, 2013)
5. Djukić, V.: DVDocFlowLang Demo, video. [Online]. Available: http://www. dvdocgen.com/Framework/DVDocFlow.wmv (Accessed: May, 2013)
6. Djukić, V.: DVDocGen Framework, Application Interface. [Online]. Available: http://www.dvdocgen.com/Framework/DVDocFramework.pdf (Accessed: May, 2013)
7. Djukić, V.:DVDocLang Language Reference. [Online]. Available: http://www. dvdocgen.com/Framework/DVDocLang.pdf (Accessed: May, 2013)
8. Djukić, V.: DVRepLang Demo, video. [Online]. Available: http://www. dvdocgen.com/Framework/ModelTransformation.wmv (Accessed: May, 2013)
9. Djukić, V.: MeMID Activities, DSM Tools and Model Execution, video. [Online]. Available: http://www.dvdocgen.com/Framework/MetaEditModelExec.wmv (Accessed: May, 2013)
10. Djukić, V.: Using DVDocIDE, video. [Online]. Available: http://www.dvdocgen.com/
Framework/UsingDVDocIDE.wmv (Accessed: May, 2013)
11. Djukić, V., Luković, I., Popović, A.: Domain-Specific Modeling in Document Engineering. Proceedings of the Federated Conference on Computer Science and Information Systems, Poland. (2011)
12. Djukić, V., Luković, I., Popović, A., Dimitrieski, V.: Domain-Specific Modeling Tools as Client Applications Providing the Production of Documents. Proceedings of the Industrial Track of Software Language Engineering workshop, Dresden, Germany. (2012)
13. Djukić, V., Luković, I., Popović, A., Ivančević, V.: Using Action Reports for Testing Meta-models, Models, Generators and Target Interpreter in Domain-Specific Modeling. Proceedings of the Federated Conference on Computer Science and Information Systems, Wroclaw, Poland. (2012)
14. Djukić, V., Popović, A.: .DVRepLang Grammar Specification. [Online]. Available: http://www.dvdocgen.com/Framework/DVDocRepLang.pdf (Accessed: May, 2013)
15. Eclipse Modeling Framework Project (EMF). [Online]. Available: http://www.eclipse.org/modeling/emf/ (Accessed: May, 2013)
16. Hartmann, T., Sadilek, D. A.: Undoing Operational Steps of Domain-Specific Modeling Languages. Proceedings of the 8th OOPSLA Workshop on Domain-Specific Modeling (DSM 2008), University of Alabama at Birmingham. (2008)
17. IbaLogic, IbaAG. [Online]. Available: http://www.iba-ag.org (Accessed: May, 2013)
18. IEC 611.31 Specification. [Online]. Available: http://www.dvdocgen.com/ Framework/ModelTransformation.wmv (Accessed: May, 2013)

19. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. ISBN: 978-0-470-03666-2. Wiley-IEEE Computer Society Press. (2008)
20. Klatt, B.: A Closer Look at the Model2text Transformation Language. [Online]. Available: http://wiki.eclipse.org/Model2Text_using_Xpand_and_QVT_for_Query (Accessed: May, 2013)
21. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley, ISBN: 0-321-55345-4. (2008)
22. Kosar T., Oliveira N., Mernik M., Pereira M. J. V., Črepinšek M., Cruz D., Henriques P. R.: Comparing General-Purpose and Domain-Specific Languages: An Empirical Study. Computer Science and Information Systems (ComSIS), ISSN: 1820-0214, Vol. 7, No. 2, 247-264. (2010)
23. LabVIEW System Design Software. [Online]. Available: http://www.ni.com/labview/ (Accessed: May, 2013)
24. Luković, I., Djukić, V.: DVDocLang vs. XSL-FO. [Online]. Available: http://www.dvdocgen.com/Framework/DVDocLang_XSL-FO.pdf (Accessed: May, 2013)
25. Luković, I., Djukić, V.: DVQL Language Specification. [Online]. Available: http://www.dvdocgen.com/Framework/DVQL.pdf (Accessed: May, 2013)
26. Luković, I., Mogin, P., Pavićević, J., Ristić, S.: An Approach to Developing Complex Database Schemas Using Form Types. Software: Practice and Experience, ISSN: 0038-0644, Vol. 37, No. 15, 1621-1656. (2007)
27. Mannadiar, R., Vangheluwe, H.: Debugging in Domain-Specific Modelling. SLE'10 Proceedings of the Third international conference on Software language engineering. (2010)
28. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. [Online]. http://www.omg.org/spec/QVT/1.0/ (Accessed: May, 2013)
29. MetaEdit+ 5.0 Beta Primer. [Online]. Available: http://www.metacase.com/download/metaedit/MetaEdit+ 5.0 Beta Primer.pdf (Accessed: May, 2013)
30. MetaEdit+ Workbench, MetaCase. [Online]. Available: http://www.metacase.com (Accessed: May, 2013)
31. Microsoft Extensible Application Markup Language (XAML). [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=19600 (Accessed: May, 2013)
32. Milićev, D.: Model-Driven Development with Executable UML. Wiley Publishing Inc. (2009), ISBN: 978-0-470-48163-9
33. Object Management Group. [Online]. Available: http://www.omg.org/ (Accessed: May, 2013)
34. OMG Systems Modeling Language. [Online]. Available: http://www.omgsysml.org/ (Accessed: May, 2013)
35. Selic, B., Gullekson, G., Ward, P.T.: Real-time Object-oriented Modeling. ISBN 0-471-59917-4. John Wiley & Sons, New Jersey, USA. (1994)
36. Simulink – Simulation and Model-Based Design. [Online]. Available: http://www.mathworks.com/products/simulink/ (Accessed: May, 2013)
37. Sunyé, G., Pennaneac'h, F., Ho, W. M., Le Guennec, A., Jézéquel, J. M.: Using UML Action Semantics for Executable Modeling and Beyond. In Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) Advanced Information Systems Engineering (CAiSE 2001), LNCS, Vol. 2068, Springer Berlin Heidelberg, 433-447. (2001)
38. User Interface Markup Language (UIML). [Online]. Available: https://www.oasis-open.org/committees/download.php/28457/uiml-4.0-cd01.pdf (Accessed: May, 2013)

39. Windows Presentation Foundation. [Online]. Available: http://windowsclient. net/wpf (Accessed: May, 2013)

Verislav Djukić received his M.Sc. degree in the area of Software Support for Information Systems from the Faculty of Military and Technical Sciences in Zagreb. At the University of Belgrade, Faculty of Organizational Sciences, he completed his Mr degree in the area of Formal Specification of Software Interfaces. He is currently a Ph.D. student at the University of Novi Sad, Faculty of Technical Sciences. He lives in Germany where he works as a director of a software company specializing in domain-specific modeling in document engineering, and measurement and control systems.

Ivan Luković received his M.Sc. degree in Informatics from the Faculty of Military and Technical Sciences in Zagreb in 1990. He completed his Mr (2 year) degree at the University of Belgrade, Faculty of Electrical Engineering in 1993, and his Ph.D. at the University of Novi Sad, Faculty of Technical Sciences in 1996. Currently, he works as a Full Professor at the Faculty of Technical Sciences at the University of Novi Sad, where he lectures in several Computer Science and Informatics courses. His research interests are related to Database Systems and Software Engineering. He is the author or co-author of over 90 papers, 4 books, and 30 industry projects and software solutions in the area.

Aleksandar Popović graduated from Faculty of Science at the University of Montenegro. He completed his Mr (2 year) degree at the University of Novi Sad, Faculty of Technical Sciences. Currently, he is a Ph.D. student and teaching assistant at the University of Montenegro, Faculty of Science. He assists in teaching several Computer Science and Informatics courses. His research interests include Software Engineering, Database Systems and Domain Specific Languages.

Vladimir Ivančević is a PhD student in Applied Computer Science and Informatics and a teaching assistant at the Faculty of Technical Sciences, University of Novi Sad (Serbia), where he also gained his BSc and MSc in Electrical Engineering and Computing. His research interests include domain specific languages (DSLs), data mining (DM), and databases. At the moment, he is involved in several projects concerning application of DSLs and DM in the fields of software engineering, education, and public health.

Received: December 28, 2012; Accepted: June 21, 2013

DOI: 10.2298/CSIS121210067R

Possible Realizations of Multiplicity Constraints

Zdeněk Rybola[1] and Karel Richta[2][3]

[1] Faculty of Information Technology, Czech Technical University in Prague
Thákurova 9, 160 00 Prague
zdenek.rybola@fit.cvut.cz
[2] Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, 118 00 Prague
richta@ksi.mff.cuni.cz
[3] Faculty of Electrical Engineering, Czech Technical University in Prague
Technická 2, 160 00 Prague
richta@fel.cvut.cz

Abstract. Model Driven Development (MDD) approach is often used to model application data and behavior by a Platform Independent Model (PIM) and to generate Platform Specific Models (PSMs) and even the source code by model transformations. However, these transformations usually omit constraints of the binary association multiplicities, especially the source class optionality constraint.
This paper is an extended version of the paper 'Transformation of Special Multiplicity Constraints - Comparison of Possible Realizations' presented at MDASD workshop at the FedCSIS 2012 conference. In this paper, we summarize the process of the transformation of a binary association from a PIM into a PSM for relational databases. We suggest several possible realizations of the source class optionality constraint to encourage the automatically transformation and discuss their advantages and disadvantages. We also provide experimental comparison of our suggested realizations to the common realization where this constraint is omitted.

Keywords: MDD, UML, transformation, multiplicity constraints, source class optionality constraint, OCL, SQL.

1. Introduction

Model Driven Development (MDD) is a development process that is based on modeling and transformations. In our case, it is based on the Model Driven Architecture (MDA) developed by the Object Management Group (OMG) [8, 10]. This process usually consists of creating a set of models of various abstraction levels and points of view. The process also consists of various transformations between these models. These transformations usually support both forward engineering and reverse engineering, the processes of transforming abstract models into more specific models or source code, or specific models into more abstract models, respectively.

The most common use case of MDD approach is the development of a Platform Independent Model (PIM) of the application data and its transformation to a Platform Specific Model (PSM) for a relational database, as well as the generation of SQL scripts to create the database schema. However, these transformations usually do not take the multiplicity constraints into account, and therefore a database schema created according to the generated PSM can be inconsistent according to the defined multiplicity constraints and the database can contain invalid data.

Therefore, in this paper, we deal with the transformation of binary associations along with their multiplicity constraints from a PIM into a PSM for relational databases. Many CASE tools such as Enterprise Architect [16] support the model transformation and the source code generation. However, they have many limitations regarding the integrity and multiplicity constraints [3]. The tools usually do not transform these constraints to an implementation.

In particular, we focus on a special case of a multiplicity constraint – the source class optionality constraint – that we consider the most often neglected constraint during the transformations. We define this constraint using another formalism then the grafical notation of the class diagram of the Unified Modeling Language (UML) – as an invariant in the Object Constraint Language (OCL). We believe such a definition can be transformed into the implementation more straightforwardly than it is done so far. For instance, OCL tools such as DresdenOCL Toolkit [4] can be used to transform such a constraint into an implementation.

Our motivation for this research is the intent to bring this issue in attention of the community of data analysts and database designers and to show that this constraint can be quite easily realized in common relational databases. We also believe that the integration of the suggested realizations in the transformation processes of CASE tools may save a lot of effort of analysts and database designers when trying to design a consistent database and even improve the database consistency as this effort is usually neglected. Therefore, we want to stimulate the motivation of CASE tool and transformation tool builders to include a realization for such a constraint in their tools to support this case of the MDD approach. Therefore, we propose several possible implementations for this constraint in relational databases and we discuss advantages and disadvantaged of each suggested implementation. Finally, we provide an experimental comparison of suggested implementations to the common approach, without the source entity optionality constraint implemented. The comparison is done from the point of view of database operations – inserts to the database, queries to the database and deletes of the data from the database.

This paper is an extended version of [14]. It extends the work presented in [13] and [11] where the rules for the transformation of a binary association from a PIM into a PSM are discussed. The contributions of this paper are the constraint implementation, including the update and delete operations, and new experiments for the delete operation and more suitable examples.

The paper is structured as follows: In Section 2, we present a running example to define basic assumptions and illustrate our approach. In Section 3, we discuss related work and existing tools and their problems in comparison to our approach. In Section 4, the transformation of a binary association and its multiplicity constraints from a PIM into a PSM for a relational database is discussed using an example. Various possible realizations of the special constraint for the source entity optionality are defined and discussed in Section 5. Experiments and their results are discussed in Section 6. Finally, in Section 7, the conclusions are given.

2. Running Example

In the running example, we use UML to express models and we use SQL as the domain specific language for relational databases used in the implementation. UML [9, 2] is a general-purpose visual modeling language for specifying, constructing, and documenting the artifacts of systems. Additional constraints for UML models are usually defined in OCL [7], which is a part of UML specification. OCL is a specification language used to define restrictions, such as invariants, pre- and post-conditions for the connected model elements. The invariants are conditions that must be satisfied by all instances of the element. OCL can also be used as a general object query language.

In the PIM, each object of a problem domain is represented by a class – in some languages called *an entity* – with a set of attributes and its instances [2]. The classes are linked together by associations to represent the relationships between the objects – instances of the respective classes. Each association has its name to describe the meaning of the relationship and multiplicities to define the number of instances of each class related to each other. Fig. 1 shows a general form of modeling a binary association by the means of a UML class diagram [2].

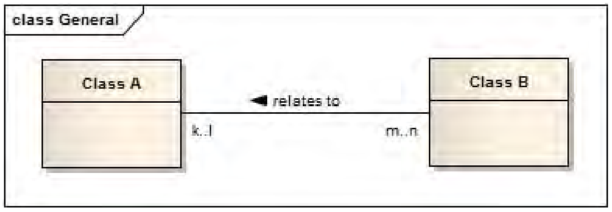


Fig. 1. Labeling of the multiplicities of an association between two classes

The *minimal multiplicity* defines the minimal number of instances of one class related to a single instance of another class. In Fig. 1, value *k* denotes the minimal multiplicity of instances of the *ClassA* for a single instance of the *ClassB* and the value *m* denotes the minimal multiplicity of instances of *ClassB* for a single instance of the *ClassA*. Although this constraint can be generally used to restrict the minimal number of instances to any value possible, for instance at least 11 members for a soccer team, usually the constraint is only used to restrict the *optionality* of the instances – if there needs to be at least one instance related – value *k = 1* – or if there can be no instances related at all – value *k = 0*.

The *maximal multiplicity* – also called *cardinality* – defines maximal number of instances of one class related to a single instance of another class. In Fig. 1, the values *l* and *n* denote the maximal multiplicities of the *ClassA* and *ClassB*, respectively. Although this constraint can be generally used to restrict the maximal number to any possible value, for instance at most 11 members of a soccer team playing in a match at a time, usually the constraint is used just to distinguish if there can be just one instance related – value *l = 1* – or there can be a collection of instances related to the same instance – value *l = **.

Further on, we will deal only with the minimal multiplicity values of *0* and *1* and the maximal multiplicity values of *1* and ***. However, our approach can be generalized for any special multiplicity values, whenever we want to restrict the number to other values.

When transforming the PIM into a PSM for a relational database, each one–to–many association is transformed to a foreign key constraint. Because the foreign key is unidirectional, we need to distinguish between the *source* and *target* class or table. The *source* class of an association is the class that is transformed into the table where the foreign key value is situated. The *target* class of an association is the class that is transformed into the table that is referred by the foreign key constraint. Usually, the *source* class is the class at the end of the association where the maximal multiplicity value is *n = ** and the *target* class is the class at the end of the association where the maximal multiplicity value is *l = 1*. Also notice that the association in the PIM is non-directional. That is because on the PIM level we only define that two classes of instances are related and define the association multiplicities but we do not define the direction of the association's realization – the direction is defined on the PSM level or during the transformation. The determination of the source and target classes of an association are discussed in more detail in [13].

In this paper, we focus mainly on the source class multiplicity constraint used in one-to-many relationships where the minimal multiplicity value of the *many*-class is equal to *one*. This constraint is often used in models when we need to restrict the required existence of both related entities in such a relationship – none of them can exist without the other one. Our approach will be illustrated on an example of ordered items, where each order must include at least one item and each item must be part of an order. The PIM of the example is shown in Fig. 2. According to the maximal multiplicities of the association, the *OrderItem* class is the *source* class and the *Order* class is the *target* class.

3. Related Work

The problem of the transformation of a PIM of the application data into the relational database is not new. There is a lot of books such as Rob and Coronel [12] describing the principles of the data modeling and the transformation techniques to the database. It is also a part of the information technology education

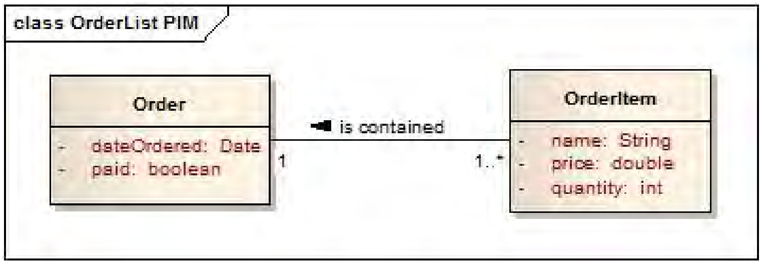


Fig. 2. PIM of one-to-many relationship of an order and its items

in most universities worldwide. Tools such as DresdenOCL [4] and Enterprise Architect [16] provide the support for such a modeling and transformations.

Rob and Coronel [12] presented the basic transformation of an ER model into database tables. They utilized FOREIGN KEY constraint to realize binary relationships and UNIQUE and NOT NULL constraints to restrict the multiplicities. They also suggested using an ON DELETE RESTRICT clause for the FOREIGN KEY constraint to prevent violation of the target entity optionality constraint, if required. However, this clause restricts only the target entity optionality. Furthermore, they suggested no solution to restrict the source entity optionality. Their suggested transformation can be also used for the transformation of a PIM into a PSM for a relational database as discussed in Section 4 with additional constraint for the *source* class optionality constraint.

In [3], Cabot and Teniente identify various limitations of a current code generation tools. The limitations concern the integrity constraints defined in PIMs, including OCL constraints and multiplicity constraints. In our paper, we focus on the multiplicity constraints and propose possible realizations of such constraints in relational databases. Regarding these constraints, Cabot and Teniente [3] identified only one tool called Objecteering/UML [15] that is able to correctly transform multiplicity constraints. In addition to the tools compared in [3], we also identify another CASE tool with similar limitations. Enterprise Architect (EA) [16] is a complex commercial CASE tool for maintenance of models, their transformations, source code generation and reverse engineering process from a source code into PSM. Besides, it provides transformations from PIM data model to a specific database PSM model, and a generation of SQL source code from such a PSM model. However, the default transformations of Enterprise Architect do not consider the optionality of associations to determine neither the direction of the relationship implementation by the FOREIGN KEY constraint nor the required multiplicity restrictions. It does not support special multiplicity values either. Although EA allows the definition of OCL constraints, the constraints are not realized by the transformations.

In [1], the authors also identify a problem of current relational databases in the realization of a source entity optionality constraint – they call this constraint in the database an *inverse referential integrity constraint (IRIC)*. The authors also present an approach to the automated implementation of the IRICs by database triggers in a tool called IIS*Case. This tool is designed to provide a complete support for developing database schemes including the check of the consistency of constraints embedded into the DB [1] and the integration of subschemas into a relational DB schema [5].

DresdenOCL Toolkit [4, 17] is a research project at the Technical University of Dresden. After loading a model and its instance along with a set of OCL constraints, the tool provides OCL syntax checking and OCL constraints evaluation. It also provides generation of SQL tables and views according to the model. OCL constraints are transformed into database views containing only records satisfying the constraint. The tool also offers transformation of the model with constraints into AspectJ for the Java source code. However, the DresdenOCL Toolkit does not consider the minimal multiplicity constraints of associations in the PIM to determine neither the *source* and *target* tables for the FOREIGN KEY constraint nor the other multiplicity constraints' realization.

4. Transformation of PIM into PSM for Relational Databases

Our approach to the transformation of a data PIM into a PSM for relational databases has been introduced in [13, 11]. This section briefly summarizes our approach.

In general, data is stored as rows in tables with a set of columns to store specific data in a relational database. Therefore, the classes of PIM are transformed into database tables with the columns corresponding to the attributes. Each row in a database table is identified by a PRIMARY KEY. The PSM generated by the transformation of the PIM of our running example (see Fig. 2) is shown in 3. The class *Order* is transformed into the *Order* table and the class *OrderItem* is transformed into the *OrderItem* table. Also notice the PRIMARY KEY columns *orderID* and *orderItemID* and constraints denoted with *PK* stereotype and prefix to identify individual rows in the *Order* and the *OrderItem* tables, respectively. In the following, we will use the *source* and *target* tables as the tables generated by the transformation of the *source* and *target* classes of the PIM, respectively, to discuss the realization of the multiplicity constraints in the PSM.

Associations defined in the PIM are realized by a mechanism called FOREIGN KEY [12]. This mechanism adds a special column or columns to the *source* table and defines the FOREIGN KEY constraint linking the FOREIGN KEY column or columns of the *source* table to the PRIMARY KEY column or columns of the *target* table. In the Fig. 3, the *orderID* column in the *OrderItem* table is defined for the FOREIGN KEY value and the FOREIGN KEY constraint is defined for that column to refer to the *orderID* column of the *Order* table. Using this mechanism, each row can refer only to a single target row, thus we can realize only one-to-one and one-to-many associations and the cardinality of the *target* table is always restricted to *1* [6]. However, many-to-many associations can be transformed into two many-to-one associations and an association table and these can be then transformed as usual [12].

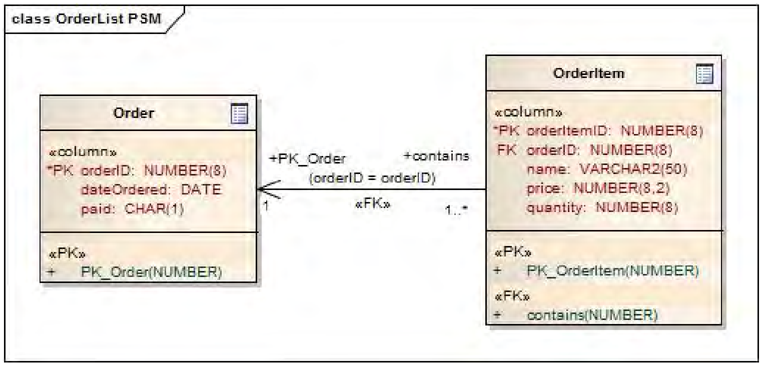


Fig. 3. PSM of one-to-many relationship of an order and its items

In fact, this restriction of the foreign key mechanism is the most important clue to determine the direction of the association. In the running example in Fig. 2, the cardinality *n = ** requires the FOREIGN KEY in the table *OrderItem* which refers to table *Order* as shown in Fig. 3, and therefore it automatically restricts the cardinality of the target table to *l = 1*.

The *target* table optionality *k = 1* can be realized by the NOT NULL constraint defined on the FOREIGN KEY column *orderID* in the *OrderItem* table. This constraint enforces each row in the *source* table *OrderItem* to refer to a row in the *target* table *Order* and thus restricting the *target* table optionality. Furthermore, for the completeness of the multiplicity constraints discussed, a *UNIQUE* constraint on the FOREIGN KEY column of the *source* table may be used to restrict the *source* table cardinality *n = 1* for one-to-one associations, as the constraint prevents the insertion of more rows in the *source* table with the same FOREIGN KEY value. However, this is not the case of our running example.

The only multiplicity value we have not restricted yet is the *source* table optionality *m = 1*. There is no possible way to restrict the source entity optionality by the means of the FOREIGN KEY. As mentioned before, the usual method is to omit this restriction and to provide the constraint checking by the application that uses the database schema [12, 13]. However, we suggest a method to express this constraint by an OCL invariant, and realize it in various ways in SQL to keep the database consistent, independently of the application. The OCL invariant is shown in Fig. 4.

```
context o:Order inv minItems:
OrderItem.allInstances()->exists(i|i.orderID = o.orderID)
```

Fig. 4. OCL constraint for the required source entity optionality

This constraint can be violated only by three operations:

1. If a new order is inserted with no items referring to this new order.
2. If the last item of an order is updated, changing its order to another one.
3. If the last item of an order is deleted.

Therefore, when executing these operations, the checks of the defined OCL invariant must be executed to ensure the data consistency. Moreover, in a relational database, one more operation can violate the constraint: if the order's ID is changed to a new value with no items referring to it. But, this operation also violates the *FOREIGN KEY* constraint, and therefore it is not possible to execute such an operation without changing the order's items, as well.

5. Realization of the Source Table Optionality Constraint

SQL scripts for creating database tables can be generated from the PSM by many tools including the EA. The creation scripts for the database tables used in the following examples of realizations of the source table optionality constraint are shown in Fig. 5. All examples are given in the Oracle SQL syntax.

```
CREATE TABLE Order (
orderID      NUMBER(8) NOT NULL,
dateOrdered  DATE,
paid         CHAR(1));

CREATE TABLE OrderItem (
orderItemID  NUMBER(8) NOT NULL,
orderID      NUMBER(8) NOT NULL,
name         VARCHAR2(50),
price        NUMBER(8,2),
quantity     NUMBER(8));

ALTER TABLE Order ADD CONSTRAINT PK_Order
PRIMARY KEY (orderID) USING INDEX;

ALTER TABLE OrderItem ADD CONSTRAINT PK_OrderItem
PRIMARY KEY (orderItemID) USING INDEX;

ALTER TABLE OrderItem ADD CONSTRAINT isContained
FOREIGN KEY (orderID) REFERENCES Order (orderID);
```

Fig. 5. SQL script for creating database tables of the running example

In some cases, after adding another constraint for checking the existing items for an order, we could not be able to insert new data because of two mutually dependent checks – the constraint checking existing items for an order, and the *FOREIGN KEY* constraint *isContained* requiring an existing order for each of the order item. This conflict can be solved by deferring one of the constraints [6]. Defining a constraint as deferrable causes the database engine to check the constraint at the end of the transaction instead of checking it in the time of inserting the data. By the deferred *FOREIGN KEY* constraint, we can insert the order items referring to the order not inserted yet, and then to insert this order. The other constraint would be evaluated when inserting the order but, in that time, there already exist the items referring to it. On the other hand, the *FOREIGN KEY* constraint is not evaluated while inserting the items, it is evaluated at the end of the transaction when the order has already been inserted. The deferred *FOREIGN KEY* constraint can be defined as shown in Fig. 6.

```
ALTER TABLE OrderItem ADD CONSTRAINT isContained
FOREIGN KEY (orderID) REFERENCES Order (orderID)
DEFERRABLE INITIALLY DEFERRED;
```

Fig. 6. SQL script for creating the deferrable FOREIGN KEY constraint

The following subsections deal with the possible implementations of the required source table optionality constraint and their pros and cons.

5.1. Database Views

The most straightforward realization of the constraint are the database views [13, 11]. Each constraint is transformed into a database view to filter only the valid data stored in a table. This approach is inspired by DresdenOCL Toolkit [4] that transforms defined OCL constraints into the database views. These views contain only the rows that satisfy the defined constraint using the WHERE clause. The realization of the constraint for the required optionality of OrderItem in Fig. 3 can be defined as shown in Fig. 7.

```
CREATE VIEW valid_orders AS
SELECT o.* FROM Order o WHERE EXISTS
(SELECT 1 FROM OrderItem i WHERE i.orderID = o.orderID)
```

Fig. 7. SQL script for creating the view to select only valid orders

The realization by the database views does not increase the time required for inserting new entries to the tables because the data is inserted directly into the table without any additional constraints checks. On the other hand, the selection of valid data contains the evaluation of the condition of the view, which increases the time required to query the data.

This approach does not automatically ensure consistency of the data stored in the database. We are still able to insert invalid data, which can violate the multiplicity constraints defined in the PIM. The application itself must use the view to work with the valid data only and must provide support for the correction of the invalid data. For this process, an inverse view can be useful to detect the invalid data violating the constraints. Such an inverse view can be defined as shown in Fig. 8.

```
CREATE VIEW invalid_orders AS
SELECT o.* FROM Order o WHERE NOT EXISTS
(SELECT 1 FROM OrderItem i WHERE i.orderID = o.orderID)
```

Fig. 8. SQL script for creating the view to select invalid orders

Updatable Database Views. To overcome this problem of the invalid data being hidden by the view, DML operations should be executed on the view instead of executing them over the tables directly. To be able to execute DML operations on the view, the view must be updatable. A view is *updatable*, if:

– it does not use a *DISTINCT* quantifier, a *GROUP-BY* or a *HAVING* clause,
– all derived columns appear only once in the *SELECT* list,
– each column of the view is derived from exactly one table,
– and the table is used in the query expression in such a way that its primary key or other candidate key relationships are preserved [6].

```
CREATE VIEW valid_orders AS
SELECT o.* FROM Order o WHERE EXISTS
(SELECT 1 FROM OrderItem i WHERE i.orderID = o.orderID)
WITH CHECK OPTION
```

Fig. 9. SQL script for creating the view to select only valid orders with CHECK OPTION clause

If the view is updatable, then DML operations like inserts, updates and deletes can be executed on the view. In fact, the operations are translated to the corresponding underlying table or tables, and executed on the data directly in these tables. Therefore, it is possible not only to manipulate with the data which is not accessible through the view, but it is also possible to violate the source table optionality constraint. To prevent such operations that affect the data which is not selected by the view, the view must be defined with the *WITH CHECK OPTION* clause [6]. This clause prevents an insertion of not accessible records and update operations that make accessible records inaccessible by the view. Example of the view definition with the check option is shown in Fig. 9.

```
INSERT INTO valid_orders (ORDERID, ORDER_DATE, PAID)
VALUES (3, sysdate, 'N');
```

Fig. 10. SQL script for inserting a new order using the view with CHECK OPTION clause

If we try to insert a new order to the database using the view as shown in Fig. 10, an exception is thrown as shown in Fig. 11.

```
Error report:
SQL Error: ORA-01402: view WITH CHECK OPTION
  where-clause violation
01402. 00000 -  "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

Fig. 11. Exception thrown by Oracle database when trying to insert new record that is not accessible by the view used for insertion

Using the updatable view with a check constraint, we can ensure that no invalid data is inserted into the *Order* table. However, we are still able to violate the source entity optionality constraint either by deleting the last item in the order or by updating the last item to another order. To prevent such operations, a view with *CHECK OPTION* should be defined joining the *Order* table and the *OrderItem* table as shown in Fig. 12. In this view, an *OUTER JOIN* must be used to filter out the orders without any item, and thus violating the *WHERE*-clause as shown in Fig. 11. However, this view is not updatable because of that *OUTER JOIN*, and therefore any updates and deletes result in an exception as shown in Fig. 13.

5.2. CHECK Constraint

In relational databases, *CHECK* constraint can be used to restrict the values in a column of a table [6]. The constraint is checked whenever a value is inserted or updated in the column, and the operation is rolled back when the constraint is violated. Such a constraint can restrict a range for the numeric values or provide

```
CREATE VIEW valid_items AS
SELECT i.*,
(SELECT COUNT(*) FROM OrderItem WHERE orderID = o.orderID) items
FROM Order o
  LEFT OUTER JOIN OrderItem i ON (o.orderID = i.orderID)
WHERE (SELECT COUNT(*)
       FROM OrderItem WHERE orderID = o.orderID) > 0
WITH CHECK OPTION;
```

Fig. 12. SQL script creating the view on the orders and their items with a CHECK OPTION clause

```
Error report:
SQL Error: ORA-01779: cannot modify a column which maps to
  a non key-preserved table
01779. 00000 -  "cannot modify a column which maps to
  a non key-preserved table"
*Cause:    An attempt was made to insert or update columns
           of a join view which map to a non-key-preserved
           table.
*Action:   Modify the underlying base tables directly.
```

Fig. 13. Exception thrown by Oracle database when trying to insert a new record that is not accessible by the view used for the insertion

a list of valid values. By this approach, we can define a *CHECK* constraint to allow only the primary key values of the orders that are referred by the rows in the order items' table. According to the SQL:1999 specification [6], the constraint for the situation in Fig. 3 can be defined as shown in Fig. 14.

```
ALTER TABLE Order ADD CONSTRAINT order_check
CHECK (orderID IN (SELECT orderID FROM OrderItem))
```

Fig. 14. SQL script to create the CHECK constraint

As the *CHECK* constraint and the *FOREIGN KEY* constraint are mutually dependent, one of them must be defined as deferrable. Other ways, we would not be able to insert a new record to any of the two tables. We will suggest the deferrable FOREIGN KEY constraint as shown in Fig. 6.
By this realization, the data consistence is ensured, since it is impossible to insert invalid data. However, there are some problems with this implementation. One of the problems is as follows: if a violation is detected by the deferred constraint, the whole transaction is rolled back because it is not possible to determine which command caused the violation [6]. Another important problem

```
Error report:
SQL Error: ORA-02251: subquery not allowed here
02251. 00000 -  "subquery not allowed here"
*Cause:    Subquery is not allowed here in the statement.
*Action:   Remove the subquery from the statement.
```

Fig. 15. Exception thrown by the Oracle database when trying to create the CHECK constraint

of this realization is the fact that, although specified by the SQL:1999 specification [6], none of the current common database engines support this kind of the *CHECK* constraints because it contains a subquery. The Oracle database returns the error message (see Fig. 15) when trying to create the *CHECK* constraint.

Therefore we cannot use this realization until the database engines provide the support for this specification.

5.3. Triggers

Triggers are special procedures available in many relational databases [6] connected to some special events in the table. In Oracle database, each trigger can be defined to be executed *BEFORE* or *AFTER* such an event, while the event can be any statement to insert new rows, update rows or delete rows, including combinations. Furthermore, the triggers can be defined to be executed for each affected row or for all rows affected by the statement at once. During the execution of the trigger, the original row data and the new row data can be accessed by special keywords. In the following, we will use the syntax of the Oracle PL/SQL language to define triggers but similar approach can be used in other databases and database languages as well. The generic form of triggers for inverse referential integrity constraint can be seen in [1].

In the context of constraints checking, a trigger can be defined to check the validity of the inserted data. Such a trigger could throw an exception if the inserted data is invalid. For the situation in Fig. 3, the trigger would check the existence of order items for the inserted order. The insert trigger for Oracle 10g database can be defined in Oracle PL/SQL as shown in Fig. 16.

This trigger is executed before each insert statement, which is executed for the *Order* table. The order items referring to the inserted order by its PRIMARY KEY are being searched. If no items are found, the exception is thrown, which causes the statement to roll back. As this trigger is always executed in the time of an order insertion, the items must be inserted before this statements. To enable it, the FOREIGN KEY constraint on the *orderID* column of the *OrderItem* table must be defined as deferrable (see Fig. 6).

The trigger ensures the data inserted into the database is consistent, as it does not allow to insert the invalid data violating the multiplicity constraint. However, the check is executed for each order insertion or update searching

```
CREATE OR REPLACE TRIGGER check_existing_items_insert
BEFORE INSERT ON Order
FOR EACH ROW
DECLARE
  l_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO l_count
  FROM OrderItem i
  WHERE i.orderID = :new.orderID;
  IF l_count = 0 THEN
    raise_application_error (-20910,
      'order item not found for the inserted order');
  END IF;
END
```

Fig. 16. SQL script for creating the trigger to check the constraint violation while inserting new orders

for the related items. This search takes the longer time the more records have already been stored in the table. However, this searching time can be decreased by defining an index on the FOREIGN KEY column *orderID* in the source table *OrderItem*. For the situation in Fig. 3, the index can be defined as shown in Fig. 17.

```
CREATE INDEX items_order_index ON OrderItem (orderID);
```

Fig. 17. SQL script for creating the index on orders of items

Moreover, this trigger does not prevent the violation of the source table optionality constraint by updating or deleting the items of an order. To prevent such violations, another trigger must be defined, see Fig. 18. This trigger checks, if there exists at least one order item for the currently referred order after updating or deleting the order item.

However, this trigger causes a mutating table exception, see Fig. 19, when trying to update or delete an item. This exception is caused because a query is executed on the table that is currently being updated and therefore the data cannot be reliable to resolve the query.

This problem can be solved by a trigger fired *AFTER* the event on the *STATEMENT* level as shown in Fig. 20. This trigger is fired after the operation of update or delete was executed and all the data was updated. Then, the trigger checks if there are any orders without the items. If it finds such orders, it throws an exception that causes the whole transaction to roll back. However, such a trigger can not detect which item caused the constraint violation.

```
CREATE OR REPLACE TRIGGER check_existing_items_up_del
BEFORE UPDATE OR DELETE ON OrderItem
FOR EACH ROW
DECLARE
  l_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO l_count
  FROM OrderItem i
  WHERE i.orderID = :old.orderID
    and i.orderItemID <> :old.orderItemID;

  IF l_count = 0 THEN
    raise_application_error (-20910,
      'No item left for the order ' || :old.orderID || '!');
  END IF;
END;
```

Fig. 18. SQL script for creating the trigger to check the constraint violation while updating or deleting items

```
Error report:
SQL Error: ORA-04091: table ORDERITEM is mutating,
           trigger/function may not see it
ORA-06512: at "CHECK_EXISTING_ITEMS_UP_DEL", line 4
ORA-04088: error during execution of trigger
           'CHECK_EXISTING_ITEMS_UP_DEL'
04091. 00000 -  "table %s.%s is mutating, trigger/function
  may not see it"
*Cause:    A trigger (or a user defined plsql function that
           is referenced in this statement) attempted to look
           at (or modify) a table that was in the middle of
           being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read
           that table.
```

Fig. 19. Exception thrown by the Oracle database when trying to update or delete a record from the OrderItem table

```
CREATE OR REPLACE TRIGGER check_existing_items_up_del
AFTER UPDATE OR DELETE ON item_table_trigger
DECLARE
  l_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO l_count FROM (
    SELECT o.orderID
    FROM order_table_trigger o
    LEFT OUTER JOIN item_table_trigger i
      ON (o.orderID = i.orderID)
    GROUP BY o.orderID HAVING COUNT(i.orderItemID) = 0);

  IF l_count > 0 THEN
    raise_application_error (-20910,
      'No item left for an order!');
  END IF;
END;
```

Fig. 20. SQL script for creating the trigger to check the constraint violation after update or delete of items

6. Experiments

To compare our proposed implementations, we made some experiments. These experiments compare our proposed realizations with the commonly used realizations without the *source* table optionality constraint checking in three areas - in inserting new orders, in selecting existing orders, and in deleting order items.

The suggested implementation by the triggers requires the select operations being executed during the insertion of the new entries to the table. Similarly, the insert operations by the view with the *CHECK OPTION* require a condition evaluation. Therefore we made an experiment to compare our suggested implementations by the triggers and views with the *CHECK OPTION* with the commonly used realization omitting this constraint. In our experiment, the implementation by the check constraint should be also tested but it cannot be implemented in the Oracle database, because it does not support the queries in the *CHECK* constraints. The insertion experiment is described in Section 6.1.

The suggested realization by the triggers also requires additional select operations during the deletion of the order items to check whether there always remains at least one item for each order. Therefore, we made another experiment to compare the execution time of our proposed implementation by the triggers with the common implementation without such a validation. The experiment is described in Section 6.2.

On the other hand, the suggested realization by the views used to select only the valid data requires an additional condition evaluation during the selection. Therefore, we also made the experiments to compare the time of the selection of the entries from the *Order* table directly, with the time of the selection using the view *valid_orders*. The experiment is described in Section 6.3.

Before each experiment and test variant, the database should contain from one to five order items for each of the already existing orders according to the following formula:

$$(orderID \; mod \; 5) + 1$$

items, where *OrderId* is the identifier of the order. To such a database, new orders and items are inserted, existing order items are deleted and existing orders are searched.

We used Oracle 10g XE database installed on Acer TravelMate 7730 (Intel(R) Core-(TM)-2 Duo CPU @ 2.00GHz with 2GB RAM, Windows 7 Professional 32-bit) for our experiments. The block size was set to 8kB and the database buffer was 52736 blocks.

6.1. The Insert Experiment

The experiment presents the time comparison of the process of inserting new entries for various implementations of a one-to-many relationship in a relational database. We developed several scripts for creating the database tables with the constraints and appropriate insert scripts for each of the implementation to simulate the process of inserting new entries into the database.

Table 1 presents the constraint implementation for each variant. The *Simple* variant is the standard implementation of one-to-many relationship with a primary key in both tables and a foreign key, which refers to the table *Order*, see Fig. 5. This variant does not restrict the minimal multiplicity for the items in the order. The *View* variant uses the view with the *CHECK OPTION* shown in Fig. 9 to insert new entries while checking whether there exist the items for this order. The *View with an index* variant uses the same view with the index defined in Fig. 17. The *Trigger* variant adds a trigger, as shown in Fig. 16, to check an existing item for each inserted order. In this variant, the trigger prevents inserting the orders without any items. Finally, the *Trigger with an index* variant adds the index on the orderID in the table *OrderItem*, as shown in Fig. 17, to speed up the search of items by their order.

Table 1. Variants of create scripts for various constraint realizations (+ implemented, * implemented deferrable, - not implemented)

Variant	primary keys	foreign key	index	trigger	view
Simple	+	+	-	-	-
View	+	*	-	-	+
View with index	+	*	+	-	+
Trigger	+	*	-	+	-
Trigger with index	+	*	+	+	-

The pseudo-SQL code of the insertion procedure for all the tested variants is given in Fig. 21. The script inserts several items with a reference to the inserted order. The number of the items of the same order differs in checking the options of inserting no, one or more items for the same order, respectively. While inserting and order, the number of its items is determined by the following formula:

$$(orderID \; mod \; 5).$$

The commit operation comes after each group of items of the same order to apply the constraint check. In the case of the *Simple* variant, the items are inserted after the order is inserted, because the FOREIGN KEY constraint is checked immediately, while in the case of other variants, the items are inserted before the order is inserted.
Fig. 22 presents the execution time of the insertion of 100 new orders for each of the variants in database already containing a various number of entries as described in the beginning of Section 6.

As we can see, the *Simple* variant proved that the execution time is nearly independent on the data already stored in the database since there are no constraints to check during the insertion. However, the optionality of the order items for each of the orders is not checked and even the orders without any items are inserted. The *Trigger* variant enforces only valid orders with at least one item to be inserted. However, the constraint check slows down the evaluation when more entries already exist in the tables. The *Trigger with the index* variant proved to be able to eliminate this problem and to be even faster than the *Simple* variant. Similar results were measured for the view implementations. The *View* variant became even slower than the *Trigger* variant because of checking the view condition after trying to insert new data. However, the *View with the index* variant eliminates the slowdown by the index and is almost equivalent to the *Trigger with the index* variant. All the measured data is summarized in Table 2.
The *strange* decrease of the time required for the insertion of data to the database containing 10000 and 100000 records in the *Simple* method is probably caused by the checkpoint processing. In the Oracle database, records are stored in data blocks in the buffer cache and the checkpoint process synchronizes the buffer cache with the data blocks in the persistent storage – usually data files. Also, for each experiment run, we delete the records inserted in the last run to insert the new data in the same database state. Therefore, some data blocks are loaded to the buffer cache just before the insert starts. Then, when inserting into a small database, there is only a few of data blocks is available to insert the data and the checkpoint process blocks the insertion when the blocks are locked for synchronization. On the other hand, in the large database, a lot of blocks is available in the buffer cache that are not locked by the checkpoint process and thus are available for insertion.
However, this applies only for the *Simple* variant as there are no special constraints aside the PRIMARY KEY that need to be checked and which cause the serialization of data access. Additionally, in all the variants except the *Sim-*

```
CREATE PROCEDURE insert_values (p_orders_count)
IS
BEGIN
  l_count := 0;
  SELECT COALESCE(MAX(orderItemID)+1,1)
   INTO l_items_count FROM OrderItem;
  SELECT COALESCE(MAX(orderID)+1,1)
   INTO l_orders_count FROM Order;
  l_starting_orders_count := l_orders_count - 1;

  FOR l_iter IN 1..p_orders_count
  LOOP
    INSERT INTO Order (orderID, order_date, paid)
    VALUES (l_orders_count, sysdate, 'N');
    COMMIT;

    FOR l_iter2 IN 1..l_count
    LOOP
      INSERT INTO OrderItem
      (orderItemID, orderID, name, price, quantity)
      VALUES (
        l_items_count, l_orders_count,
        'item' || l_iter2, mod(l_orders_count, 10)+1,
        mod(l_orders_count, 20)+1);

      l_items_count := l_items_count + 1;
    END LOOP; -- insert items
    COMMIT;

    l_orders_count := l_orders_count + 1;
    l_count := mod (l_count + 1, 5);
  END LOOP; -- insert order
END; -- insert_values
```

Fig. 21. Pseudo-SQL code of experimental insert scripts

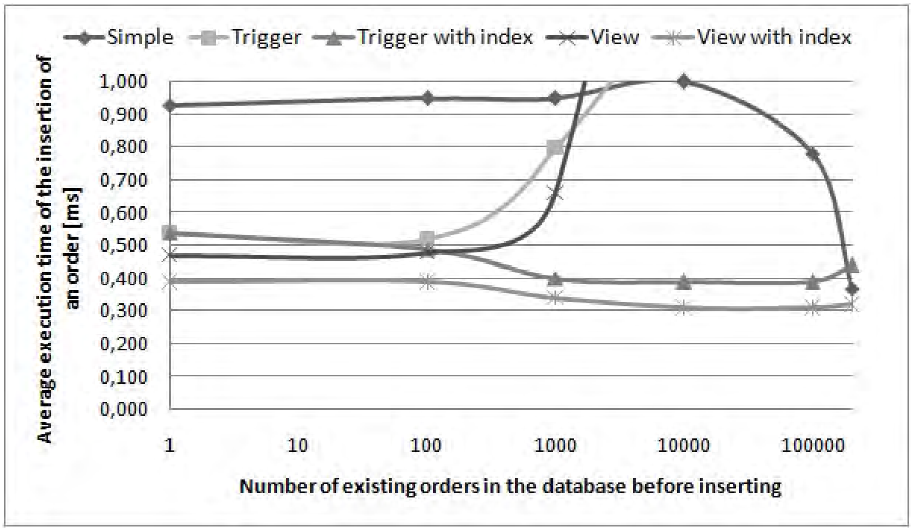


Fig. 22. Execution time of insertion of new entries for various implementation variants

ple variant the FOREIGN KEY constraint is deferrable. This causes the constraint to be checked at the end of the transaction and therefore it requires post-processing that eliminate the advantage of many available blocks in the buffer cache.

Table 2. The results of the insertion experiment - execution times of new entries insertion for various implementations in milliseconds.

Number of entries	Simple	Trigger	Trigger with index	View	View with index
0	0.930	0.540	0.540	0.470	0.390
100	0.950	0.520	0.490	0.480	0.390
1000	0.950	0.800	0.400	0.660	0.340
10000	1.000	1.590	0.390	3.150	0.310
100000	0.780	14.510	0.390	29.000	0.310
200000	0.370	28.980	0.440	57.870	0.320

Also note that the PRIMARY KEY value is generated in a sequence. If it is generated randomly, the insert would take more time as the correct data block would be needed to be loaded to the buffer cache to insert the record in the correct place according to the PRIMARY KEY value. It would especially affect the *Simple* variant in large databases where the execution time would not decrease.

6.2. The Delete Experiment

The delete experiment compares the execution time of the delete operations on the table *OrderItem*. If the source entity optionality constraint is realized by the trigger as defined in Fig. 18, the DELETE operation requires to select the orders and its items to check if the deleted item was not the last one, and thus making the order invalid. This constraint check slows down the DELETE operation as demonstrated by this experiment.

Table 3. Variants of deletes executed and measured (+ implemented, - not implemented)

Variant	Trigger	Index
Simple	-	-
Trigger	+	-
Trigger with index	+	+

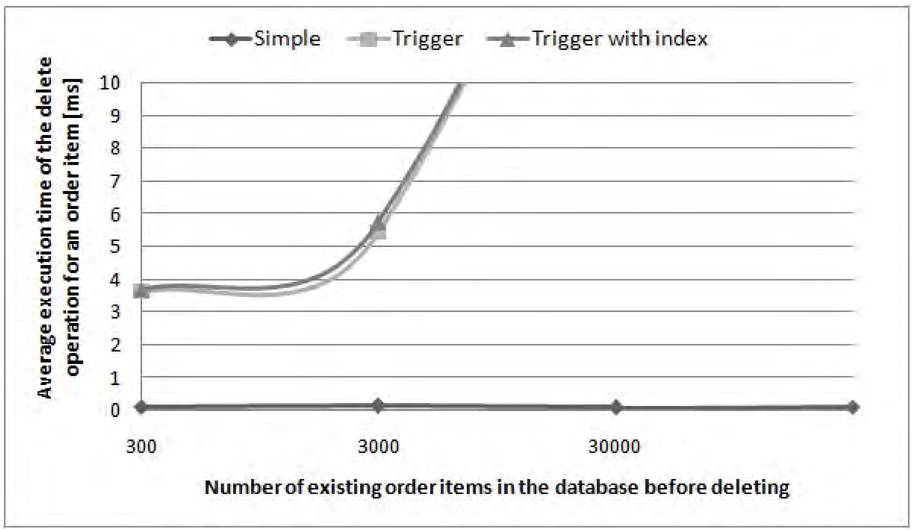


Fig. 23. Execution time of the deletion of the order items for various implementation variants

Three various implementations were tested to compare. The *Simple* variant represents the delete operations on the *OrderItem* table directly without a constraint realization. The *Trigger* variant represents the delete operations with the trigger defined on the *OrderItem* table as shown in Fig. 18. In this variant, only such items are deleted that do not violate the source entity optionality constraint of the orders. The *Trigger with the index* variant uses the same trigger. However, in this variant, the index is defined as shown in Fig. 17 to speed up the search of the items by the order identifier. All the variants are summarized in Table 3.

Fig. 23 presents the execution time comparison for the deletion of the last 100 order items inserted to the database by its *OrderItemID* attribute executed for various number of orders and items existing in the database. Before such a test, the database contains the data as described in the beginning of Section 6.

As we can see, the *Simple* variant is the fastest, since there are no constraints to check when deleting the order items. However, the required optionality of the order items for each order is not checked and the orders without any item can appear in the database. It violates the *source* table optionality constraint. The *Trigger* variant prevents from deleting the last item of an order, however, the execution time is much slower, especially if more orders and order items are stored in the database. Even the index does not help because it is not used in the checking *SELECT* operation in the trigger while joining orders and its items. The measured data is summarized in the Table 4.

Table 4. The results of the deletion experiment - execution times of deletion of order items for various implementations in milliseconds.

Number of order items	Simple	Trigger	Trigger with index
300	0,11	3,63	3,69
3000	0,14	5,45	5,74
30000	0,09	26,45	26,33
300000	0,13	256,76	256,72

6.3. The Select Experiment

This experiment presents a comparison of the execution time of a *SELECT* operation from the table *Order* directly, and by the view for accessing the valid data only. The *SELECT* statement is shown in Fig. 24, where *X* is a random order identifier. It searches for an order by its *OrderID*. No other conditions were measured because we compared the effect of the source entity optionality constraint check for the selected data. Therefore the more data is stored in the database, the slower the *SELECT* statement is.

```
SELECT * FROM Order WHERE OrderID = X;
```

Fig. 24. The SELECT statement for the selection experiment

Three various implementations were measured for each selection. The *Simple* variant presents the selection from the table *Order* directly without checking the constraint. The *View* variant presents the selection from the view *valid_orders* defined over the table *Order* to check the existing entries in the *Order* table and to select only from valid orders. The *View with the index* variant presents the selection from the view *valid_orders* defined over the table *Order* with the index defined on the order identifier in the table *OrderItem* to speed up the search of the items of the order while checking the existence. All select variants are summarized in Table 5.

Fig. 25 presents the results of the experiment. It shows the execution time of each variant for various number of orders stored in the *Order* table together with associated items in the *OrderItem* table as described in the beginning of Section 6.

The *Simple* variant proved to be the fastest variant – as expected – since there is no additional condition to check during the selection. However, the query returns back both valid and invalid data according to the source entity optionality constraint. The *View* variant results become much slower when more entries are stored in the tables, because of an additional constraint with a subquery for checking the valid orders. However, only valid orders according to the source entity optionality constraint are returned back. The index defined over the foreign key value in the *OrderItem* table speeds up the subquery execution rapidly as shown by the *View with index* variant results. Therefore, the *View with the index* variant seems to be nearly equivalent to the direct selection from the table in the execution time. However, the *View with the index* provides only the valid data. The measured data of the experiment is summarized in Table 6.

Table 5. Variants of selects executed and measured

Variant	Source	Index in the OrderItem table
Simple	table Order	not defined
View	view valid_orders	not defined
View with index	view valid_orders	defined

Table 6. The results of the selection experiment - execution times of *SELECT* operations for various implementations in seconds

Number of entries	Simple	View	View with index
100	0.002	0.002	0.003
1000	0.000	0.000	0.000
10000	0.000	0.002	0.000
100000	0.004	0.009	0.014
500000	0.039	0.061	0.031
1000000	0.050	0.519	0.039
2000000	0.054	3.091	0.041

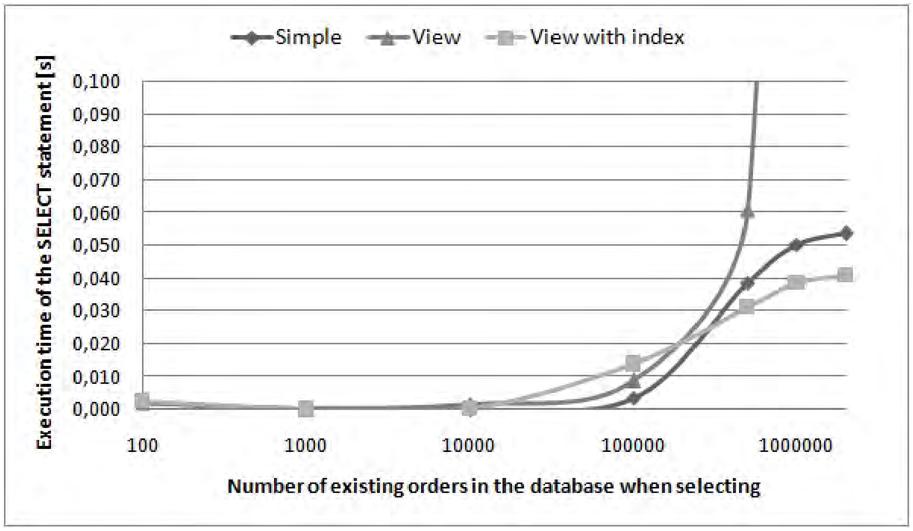


Fig. 25. Execution time of selection of entries for various implementation variants

7. Conclusions

In this paper, we summarized the currently used method for modeling binary associations in the data models using UML class diagrams. We showed the way to specify multiplicity constraints in the model. Furthermore, we showed a usual transformation of the model from PIM to PSM for the relational database and the usual transformations for multiplicity constraints using *FOREIGN KEY*, *NOT NULL* and *UNIQUE* constraints in SQL.

We pointed out the constraint for the source entity optionality. This constraint is often used in the model but not realized in the database because the foreign key is insufficient instrument for full implementation. Therefore, we defined this constraint in another formal way by an OCL invariant and suggested several methods how this constraint can realized in a relational database.

We also compared the suggested implementations to the currently used approaches in the context of the execution time while inserting new data to the tables, deleting data from the tables and selecting existing data from the tables. The experiments showed that the trigger realization and the view realization slow down the insertion of new data the more rapidly the more data has been stored in the tables. However, when the index is defined in the referring table, this slowdown is eliminated and the insertion is even faster. The results also showed that selecting the data using the view with the index on the FOREIGN KEY column is equivalent in the execution time to the direct access while providing only the valid data. However, when trying to check the *source* table optionality constraint by the trigger when deleting the data, the trigger implementation showed to be very slow even with the defined index.

According to the experiment results, we suggest the constraint should be realized in CASE tools' transformations of data models to relational databases either by the trigger or the view with the check option to prevent it from inserting invalid data or by the view to filter invalid data from the selection. However, the realization of the constraint check for the delete and update operations should be objectives of the future research to be able to fully prevent the invalid data being present in the database. We also believe that the integration of the suggested realizations in the transformation processes of CASE tools may save a lot of effort of analysts and database designers when trying to design a consistent database and even improve the database consistency as this effort is usually neglected.

Acknowledgments. We would like to thank for financial support of Student Grant Competition of CTU in Prague, grant number SGS13/099/OHK3/1T/18 and also to AVAST Foundation in Prague.

References

1. Aleksić, S., Ristić, S., Luković, I.: An approach to generating server implementation of the inverse referential integrity constraints. In: Proceedings. AL-Zaytoonah University of Jordan, Amman, Jordan (May 2011)
2. Arlow, J., Neustadt, I.: UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition). Addison-Wesley Professional (2005)
3. Cabot, J., Teniente, E.: Constraint support in MDA tools: A survey. In: Rensink, A., Warmer, J. (eds.) Model Driven Architecture Foundations and Applications. Lecture Notes in Computer Science, vol. 4066, pp. 256–267. Springer Berlin / Heidelberg (2006), http://www.springerlink.com/content/4902321654674181/abstract/
4. Demuth, B.: DresdenOCL. http://www.reuseware.org/index.php/ DresdenOCL (Jan 2011)
5. Luković, I., Mogin, P., Pavićević, J., Ristić, S.: An approach to developing complex database schemas using form types. Software: Practice and Experience 37(15), 16211656 (Dec 2007), http://dx.doi.org/10.1002/spe.v37:15
6. Melton, J.: Advanced SQL:1999. Morgan Kaufmann Publishers (2003)
7. OMG: Object constraint language, version 1.3. http://www.omg.org/spec/OCL/2.2/PDF (Feb 2010)
8. OMG: Object management group. http://www.omg.org/ (Dec 2011)
9. OMG: UML 2.4.1. http://www.omg.org/spec/UML/2.4.1/ (Aug 2011), http://www.omg.org/spec/UML/2.4.1/
10. OMG, Miller, J., Mukerji, J.: MDA guide version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf (Jun 2003)
11. Richta, K., Rybola, Z.: Transformation of relationships from UML/OCL to SQL. In: ITAT 2011: Zborník príspevkov prezentovaných na konferencii ITAT. vol. 11. University of P. J. Šafárik, Košice, Slovakia, Terchová, Slovakia (Sep 2011), http://itat.ics.upjs.sk/proceedings/itat2011-zbornik.pdf
12. Rob, P., Coronel, C.: Database Systems: Design, Implementation, and Management. Boyd & Fraser, 2nd edn. (1995)

13. Rybola, Z., Richta, K.: Transformation of binary relationship with particular multiplicity. In: DATESO 2011. vol. 11, pp. 25–38. Department of Computer Science, FEECS VSB - Technical University of Ostrava, Písek, Czech Republic (Apr 2011), http://www.informatik.uni-trier.de/˜ley/db/conf/dateso/dateso2011.html
14. Rybola, Z., Richta, K.: Transformation of special multiplicity constraints - comparison of possible realizations. In: Proceedings of the Federated Conference on Computer Science and Information Systems. pp. 1357–1364. FedCSIS, Wroclaw, Poland (Sep 2012)
15. Softeam: Objecteering/UML. http://www.softeam.com/technologies_objecteering.php (Dec 2012)
16. Sparx Systems: Enterprise architect - UML design tools and UML CASE tools for software development. http://www.sparxsystems.com.au/products/ea/index.html (Mar 2011)
17. Wilke, C., Thiele, M., Freitag, B.: Dresden OCL: manual for installation, use and development (Oct 2010)

Zdeněk Rybola is a PhD. student and an assistant professor at the Department of Software Engineering at the Faculty of Information Technology, Czech Techical University in Prague. His area of interest includes Model Driven Development in context of relational databases and multiplicity constraints and the usage of OntoUML in software engineering.

Karel Richta is an associate professor at the Department of Software Engineering at the Faculty of Mathematics and Physics, Charles University in Prague, and also at the Department of Computer Science and Engineering at the Faculty of Electrical Engineering, Czech Technical University in Prague. His research is primarily focused on formal specifications and similar approaches usable in software engineering. He has published more than 100 publications, including 5 books. He is the president of Czech ACM Chapter.

Received: December 10, 2012; Accepted: September 2, 2013.

DOI: 10.2298/CSIS130115069L

Testing framework for embedded languages[⋆]

Dániel Leskó[1] and Máté Tejfel[1]

Eötvös Loránd University
Department of Programming Languages and Compilers
{ldani; matej}@elte.hu

Abstract. Embedding a new programming language into an existing one is a widely used technique, because it fastens the development process and gives a part of a language infrastructure for free (e.g. lexical, syntactical analyzers). In this paper we are presenting a new advantage of this development approach regarding to adding testing support for these new languages.
Tool support for testing is a crucial point for a newly designed programming language. It could be done in the hard way by creating a testing tool from scratch, or we could try to reuse existing testing tools by extending them with an interface to our new language. The second approach requires less work, and also it fits very well for the embedded approach. The problem is that the creation of such interfaces is not straightforward at all, because the existing testing tools are mostly not designed to be extendable and to be able to deal with new languages.
This paper presents an extendable and modular model of a testing framework, in which the most basic design decision was to keep the – previously mentioned – interface creation simple and straightforward. Other important aspects of our model are the test data generation, the oracle problem and the customizability of the whole testing phase.

Keywords: testing support for embedded languages, testing framework, abstraction over evaluation.

1. Introduction

Nowadays, embedding a language into an existing one (host language), is a well known and widely used approach to create a new programming language. This quickens the development process, because the host language's infrastructure (lexical, syntactical analyzer) can be reused. Modern functional host languages are flexible enough that the resulted combination has more the feel of a new language than just a library.

The "embedded" approach has proved to be an excellent technique for specifying and prototyping domain-specific languages (DSLs) [11]. Basically two approaches exist: shallow embedding, which directly maps the new language constructs to their semantics, while the deep embedding first builds an abstract syntax tree and later this tree is mapped to the language semantics.

⋆ Supported by ELTE TÁMOP-4.2.2/B-10/1-2010-0030

Shallow embedding can be seen as an augmentation to an existing language. According to Mernik et al. [17] every library can be seen as a shallowly embedded language. While deep embedding really forms a new language on the foundations of the host language. Therefore the deep approach is more suitable for building a compilable and optimizable language. In this case a host language can be seen as a very powerful template or macro language.

There are numerous papers about embedded DSLs, such as how to design [15, 13], implement [10] or compile [7] them. However, as far as we know there are no specific paper, which aims to present a general solution for adding testing tool support – at low cost – for already existing embedded languages, while – as we all know – it is crucial for a language to have a proper testing tool support.

Implementing a testing system in a DSL is mostly not an option due to its labour-intensive nature and the fact that most DSLs are not designed for that kind of task. The realistic option is to use an already existing test environment, written in the host language. In this case we need to extend the existing framework with an interface to the embedded language, while the business logic remains untouched. Doing this is a much smaller task, than creating the testing support from scratch. This approach also fits nicely with the motivation of embeddedness, namely to save time and resources by reusing as much from the host language as possible.

The main problem with existing testing tools (QuickCheck [5], JUnit [2], HUnit [12]) that they were designed to test programs of *one* specific language and sometimes even for specific testing and test data generation methodologies. Therefore they are not easily extendible with an interface to another programming language or to another evaluation method. Furthermore they could be quite specific in certain aspects, like how the input test data are produced, or how the results are evaluated and decided whether a test case failed or passed. Another aspect regarding a testing framework is the viability and clear designability of supporting both property based and differential testing methodologies.

Our goal in this paper is to present a general and permissive model of a testing framework which can address properly all the previously mentioned aspects. The model is abstracted along four orthogonal aspects such as test data generation, the used evaluation method, the oracle problem and partly the used testing methodology.

Based on our model, a Haskell implementation was created. Its main characteristics are modularity and extensibility. The existence of such framework in Haskell (or in any host language) results that an embedded language can get a testing support by implementing only a simple and straightforward interface which spans the gap between the host and the embedded language.

2. The model

The following model was inspired by all the previously mentioned reasons and aspects, and it was designed to nicely fit for all of them. Figure 1 shows the data-flow model of a test case in our model.

The *generator*'s responsibility is to provide correctly typed input for *transformer*s. This input will be referred as the generated test data.

A *transformer* encapsulates the to-be-tested computation, and gives an universal interface, which hides such unnecessary details as how and by what the computation will be evaluated. A *transformer* can be thought as a function, whereat the generated test data is applied, and it yields the result of the computation. The number of the *transformer*s are not limited, it could be as many as the user wants.

A *property* is a function, which receives each *transformer*'s result, and also the originally generated test data. The outcome is a boolean value, which represents whether the specific property (given by the user) holds in that particular case, or not.

An *operator* is basically a driver, which controls the data-flow between the small boxes (in the figure). The configuration comes from the outside world (from the model's point of view), and it affects the *operator* (e.g. the number of performed tests).

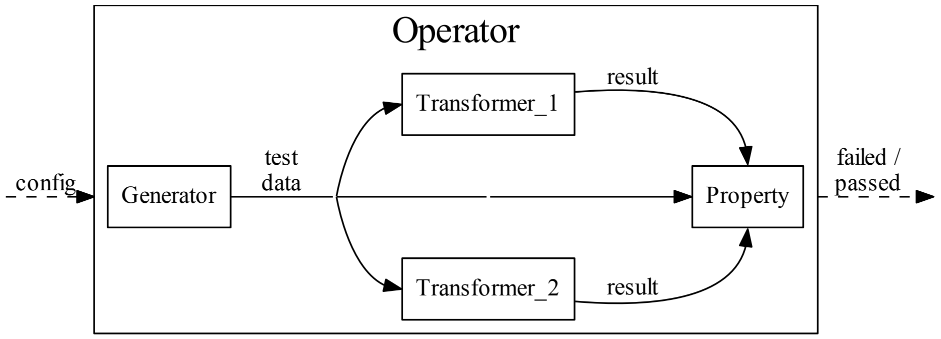


Fig. 1. The data-flow model of a test case with two transformers

To describe a test case in the terms of this model, we need one test data *generator*, a non-empty list of *transformer*s, and one *property*. To be able to run a test case, we also need an *operator*, which specifies the way how to do that.

One of the earliest design decision was that the model has to be as general as possible, in terms of that the four major parts of the model (*generator*, *transformer*, *property*, *operator*) have to be separate, independent and modular parts while the interactions between them should done through well defined and public channels.

2.1. Generator

The model's only expectation about a *generator* is that it has to supply test data with a matching type for the given *transformer*s. However there are a lot of uncontrolled aspects by the model, such as the used generation tactics (random, exhaustive), the test data distribution or controlling the size of the generated test data. All of these are entirely depending on the particular implementation of a specific *generator*.

To give control for the user over the previously mentioned aspects, we definitely need a small domain-specific language for building and specifying *generator*s. Since QuickCheck [5] and also SmallCheck [20] has some really powerful tools to do that, we can easily reuse those existing tools as a library. Note that the re-usage of such libraries not conflicts with the language independence of the model, since QuickCheck is re-implemented for more than 20 languages.

The model and the *generator* notion is not limited to the previously mentioned tools, any DSL or library which aims to generate test data could be integrated easily into the model and also into a testing framework, based on the model.

2.2. Transformer

Using a *transformer* is a way to abstract over specific evaluation methods (etc. interpreting, compiling) and specific programming languages. The model represents a *transformer* as one function, but under the hood it is a bit more complicated. A *transformer* is usually created by applying the to-be-tested function to a *transformer pattern*. Therefore the previously mentioned abstraction are done by the means of *transformer patterns*.

A specific *transformer pattern* could evaluate any program of a specific language with a specific evaluation method (e.g. C compiler, Haskell compiler, Haskell interpreter). On the implementation level it is a higher order function which takes a function (the to-be-tested) as its first argument and a complex data structure (holds the input data) as its second argument. Adding support for a new language or evaluation method can be done simply by creating a new *transformer pattern* for it.

2.3. Property

One of the most fundamental questions in automated software testing is to decide whether a test case is passed or failed, because it is hard to create an algorithm/oraculum which is general enough to correctly judge the results. However, if we somehow succeed, it is not a flexible solution to hardwire this decision method into a general framework.

A common resolution of this problem is to devolve this job to the user, who writes the actual test case. In our model it can be done with the use of the so-called *property*, which can be thought as a boolean function with two parameters. Note that our notion of *property* is quite far from QuickCheck's. Our version means a much smaller part of the model. An expression evaluated to `true` indicates a successful test, `false` indicates a counter example. The first parameter of a *property* is the originally generated test data (the input of the transformers), the second is a list of transformer's results. The number of the compared transformers are not limited, but they have to have the same type signature.

The model does not specify what kind of results should be passed to a *property*, because that could depend on the used specific *transformer pattern*. For example a *transformer pattern* of a C compiler could pass compile time and ELOC (effective lines of code) information besides the result of the computation. Having also non-functional results can allow us to build more sophisticated *properties*.

2.4. Operator

The role of an *operator* is much more technical then the previously mentioned three parts of the model. This difference comes from the fact that an *operator* isn't part of a test case, it is only needed to perform the execution of it.

On the model level an *operator*'s job is to handle the data-flow from the *generator*s towards to a *property* through one of the *transformer*s. On the framework level there are several other technical responsibilities such as controlling the number of required iterations, the level of verbosity, setting logging and the working directory. The framework contains one predefined *operator*, which gives a standard way to handle the previously mentioned aspects. So normally a user doesn't have to create a new *operator*.

2.5. Testing framework - based on our model

The presented model is general in the sense that it doesn't require any specific programming language constructs, so it can be implemented in almost any host language. We have chosen Haskell, because it nicely fits for this job [11], and lately Haskell is a very favoured host language.

The implemented framework supports both property based testing and differential testing. As a property based tester it is a kind of generalization of QuickCheck and SmallCheck with the support for additional test data generation approaches. The most important feature – as a differential testing tool – is a "common ground" for different evaluation methods and also for different languages. The existence of this common ground makes comparability and modularity really easy.

In the following section we are discussing a detailed use case of the framework. At first sight it may look like that we are using it solely as a differential testing tool, but in reality it is rather about new *transformer pattern*s, mostly answering the why and how questions.

3. A detailed use case - Testing support for Feldspar

We used the framework to test the Feldspar[1] [1] language, which is a new embedded language for describing digital signal processing algorithms. We have more than 300 test cases, which were used on a daily basis to test the Feldspar language itself, and the existing Feldspar example programs. We also had to ensure that the Feldspar compiler and interpreter are in sync, and the results are valid (compared to trusted reference implementations or expected output sets). Besides testing equivalence, most of the test cases are checking non-functional properties too.

The actual implementation of the testing framework reused QuickCheck's `Gen` class and SmallCheck's `Serial` class as *generator*s. The used *properties* are mostly check equivalence either strictly or with a given epsilon threshold. We also used non-functional *properties*, which were about compile-time, run-time, ELOC, memory-usage.

The most important and interesting part is the *transformer*, which we present in the following subsections. Each subsection firstly introduces an aspect of Feldspar which will be tested, than shortly presents how such a specific *transformer pattern* can be created and how the testing can be achieved.

3.1. Testing the Feldspar interpreter

Unfortunately Feldspar doesn't have a written specification about semantics, so the interpreter couldn't be tested against that. But in many ways Feldspar is really similar to Haskell, in fact – by definition – a lot of primitive function and operator of Feldspar has the exact same semantics like their equivalents in Haskell.

This realization means, that we could test the Feldspar's primitive functions against their Haskell equivalents. In order to do that we need two new *transformer pattern*s. The first one will be responsible for the evaluation of a Feldspar program, using the Feldspar interpreter, while the other one will evaluate a Haskell function by the Haskell interpreter.

3.2. Testing the Feldspar compiler

The Feldspar compiler [6] has a capability of supporting several different back-ends to generate code for them. The main and mostly used platform is ANSI C, there are other developments like Ti specific instrinsics or LLVM back-end, but from the testing point of view, the C platform is the important one.

It is important from the testing point of view, that the generated C code is just a function, which has almost the same arguments, like the original Feldspar program.

[1] Developed by the Feldspar project, which was a joint research project of Ericsson; Chalmers University of Technology (Göteborg, Sweden) and Eötvös Loránd University (Budapest, Hungary).

As there is no written specification for Feldspar, the interpreter could be thought as some kind of executable reference implementation of the semantics. So the best way to verify the compiler is to test it against the interpreter.

The following steps have to be done, if we would like to compile and execute a Feldspar program and run from Haskell. So creating a *transformer pattern* – which evaluates a Feldspar program by the Feldspar compiler – requires that these steps have to be automated and built-in:

1. Compile a Feldspar program into a C function.
2. Generate a proper C main() function, which reads the input data from standard input, passes these data to the previously generated C function, and prints the result to standard output.
3. Compile the previous two C files with gcc.
4. Start an external process from Haskell to run the executable file, and feed it with proper test data.
5. Wait until the execution ends, read the result from the process, and close it.

3.3. Testing Feldspar programs against reference implementations

As it was mentioned earlier, Feldspar is a domain specific language (DSL), targeting digital signal processing (DSP). So it is obvious that there is a certain set of algorithms, which are very typical for that domain. We can assume that there are notable algorithms (e.g. Fast Fourier Transform [4]), which already have an implementation from a reliable source. These implementations can be treated as a reference to check and test the expressiveness, the usability and the correctness of the Feldspar language itself, and also the programs written in Feldspar.

Since DSP algorithms are mostly implemented in C, we need a *transformer pattern* that can evaluate an arbitrary C function. The function is either given as a string, or as an external file. Evaluation means here that the *transformer pattern* takes the C code, compiles it with a C compiler (e.g. `gcc`), generates a proper `main()` function, feeds this function with the generated input data, and reads the result back to the testing framework.

Since the C language is not embedded into Haskell, and there is no strong connection between those two languages, we are losing some type information there. While in case of Haskell and languages embedded into Haskell we could statically type-check the assembled test cases. This means that if a generator-transformer-property is ever wrongly paired or assembled, then we get a compile time type-check error.

Because of the less type information, the *transformer pattern* for C has to assume that a few invariant – regarding to the number and order of the parameters – have not been violated, otherwise we could end up with sudden run-time errors.

The typical use case of this pattern is to first have a *transformer* for a Feldspar program and then compare that with a reference C program. At first this testing approach could be a little bit confusing, because a failure doesn't mean always a bug in the Feldspar language, it is always a possibility that the Feldspar implementation of the chosen algorithm is simply wrong. But as these test cases are based on real life examples, eventually the goal is to produce a properly working Feldspar implementation of those algorithms.

3.4. Simple testing approaches

In the following we will present, how the testing framework supports some basic testing methodology, such as negative testing, or testing with constant input.

Testing with constant data There are certain cases when the random test data generation is not enough, and we want to create some hardwired test case with specific input data. But in this case, we also might want to specify the result of the computation. It's basically the oldest and simplest version of testing, the input and the result are given manually, and we check it against the computed result.

To support this kind of testing, we need *transformer pattern*, which gets a constant as its first arguments, and results a constant transformer, always yielding the given constant.

Negative testing As it was mentioned earlier in subsection 2.2, a *transformer* could fail, but this failure is also handled as data. Besides the error message, there are some information about the source of error (e.g. Haskell, gcc, Feldspar compiler, Feldspar interpreter).

In order to build such a test case, which passes when one of the transformers fails, we need a new *transformer pattern* which constantly yields failure with the given error source.

The most likely use case for this kind of testing, if there are a certain set of Feldspar functions, which shouldn't be compiled to C, because they clearly hasn't got enough information (e.g. too general type signature) to produce a proper C code.

3.5. Concrete test case

The following is a Feldspar function, which takes an int stored on 32 bit (as the starting value of the accumulator) and a list of ints. The list is folded, while every element is added to the accumulator, which is the result of the function.

```
foldAdd :: Data Int32 -> DVector Int32 -> Data Int32
foldAdd = Feldspar.Vector.fold (+)
```

The `tc1` example test case tests the `foldAdd` function by comparing the Feldspar compiler, the Feldspar interpreter and the corresponding `foldl (+)` Haskell function.

```
tc1 = TestCase
    { tc_name = "testing fold with addition"
    , gen     = genInt32 ::> vectorOf 200 genInt32 ::> ()
    , trans   = [ refHaskellTP (Prelude.foldl (+))
                , evalTP foldAdd
                , compilerTP foldAdd]
    , prop    = base 1 strictEquality}
```

The results are strictly compared to the result of the Haskell function, which is treated as a reference implementation for Feldspar's `fold` function.

The test data is coming from a list of QuickCheck generators. Please note, that Feldspar uses fixed length lists (vectors) due to efficiency and optimization reasons, therefore we have to fix this information (200) at test data generation, otherwise we could use QuickCheck's `arbitrary` function or SmallCheck's `serial` function too, as a *generator*.

4. Improving transformers

Every testing system is made to support the development or maintenance by saving time and resources which allows to create and run more tests in an automated way.

The presented model and framework supports this goal, but still we have to create every test case manually for every function (like `foldAdd`) we would like to test, which is a very boring and time consuming work. Besides this obvious drawback there is an other disadvantage, namely that it is very easy to leave out a few functions during test case production.

This motivated us to enhance the previously presented *transformer* concept by generalizing the parameter of a *transformer pattern*. Previously a pattern expected a concrete function as its parameter to form a *transformer*. The new, enhanced *transformer pattern*s expects only a type signature instead of a concrete function.

Meta-transformers Type wise a meta-transformer looks and feels like an average *transformer*, but internally it is a bit more complicated. We can form a meta-transformer by applying only some type signature information on a *transformer pattern*. Based on this type information, the meta-transformer will internally generate the to be tested function with a matching signature to the given type information. This internal generation instantiates a meta-transformer and creates an ordinary *transformer*.

Basically we need type signature guided, automatic program generation. Since it is not an easy task, we solve it in two consecutive steps. The first phase does the real generation, and ensures the type correctness by producing closed and correctly typed lambda calculus terms. While the second phase translates the generated lambda term into a concrete program.

Palka et al. [18] successfully applies correctly typed lambda term generation to generate and test Haskell list expressions. Our approach is basically a generalization and improvement of their work.

The details and background of this two-phase generation go way beyond the scope of this paper and testing framework. Our point here is that the presented model and testing framework is so modular and flexible enough to accommodate even this kind of improvements too.

5. Related work

Helvetia [19] is a tool chain for developing an internal DSL by transforming an abstract syntax tree. The benefit of this approach is that a homogeneous tool support can be given for the newly created embedded languages. Therefore – in this case – there is no need for our testing framework. However our approach is applicable for already existing embedded languages, while to benefit from the homogeneous tool support of Helvetia we have to reimplement and embed our language into Helvetia, which is a much bigger task than just creating a *transformer* for our test framework.

So Helvetia only suits well for you if you are at the beginning of the language development process, but later it is not really an option to apply. While our solution is easily applicable for new and also existing languages too.

Testing embedded languages Grima et. al [8] developed an embedded language (in Haskell) addressing geometrical problems. The paper presents two different methodologies to test programs, written in that new language. The first simply reuses QuickCheck, while the second works on C level. Both using random input generation to verify the given properties, but they are two, completely separate solution on implementation level.

Our test framework could solve this in an *unified* way instead of those separate solutions. Furthermore the usage of our framework would save a considerable amount of time and resources, because we only need to create the two *transformer pattern*s (one for the Haskell level and one for the C level testing), the rest is already in the framework.

Test data generation The test data generation is always a crucial point in automated software testing. Numerous property or specification based testing tools are using some kind of test data generation. For example: QuickCheck [5], SmallCheck [20], Gast [14], Korat [3]. QuickCheck uses random generation (with the ability to shrink the founded counterexamples), while SmallCheck, Gast and Korat do exhaustive generation up to a limit given by the user.

It looks like that every tool supports only a specific programming language and a specific test data generation methodology. Our framework is designed to support arbitrary number of different test data generation methodologies and also to hide their differences by giving a unified *generator* interface. For example the presented model can accommodate QuickCheck's random generator as well as SmallCheck's serial generator at the same time.

Differential testing McKeeman states the following: "The ugliest problem in testing is evaluating the result of a test." [16]. He was the first, who described the use of randomized differential testing for C compilers. His domain was essentially static: a test data was randomly generated based on a model of the valid inputs. The tested programs were compiled by different translators, and if the obtained results are different, the situation is considered to be potentially erroneous. The word "potentially" is important here, because the results – given by the two tested programs – may differ and yet still be correct depending on the requirements. This is the starting point of our *property* notion, which solves the oracle problem by simply porting it to the user.

McKeeman's model is really close to ours in the sense that he had a test data generator, translators – which corresponds with our *transformer* notion – and some kind of very simple property to check the results. However, his solution was specifically designed to test different C compilers with random test data, therefore there is no chance for such kind of extendability and flexibility like new test data generation methods, language interfaces and properties.

A reusable framework One of the simplest reusable framework is JUnit [2], and it's clones for other languages, like HUnit [12] for Haskell. Our model aims to preserve the simplicity of the previously mentioned tools, but also tries to support differential and property based testing, arbitrary test data generation methodologies and language independence in the sense of the tested program.

A test framework was developed for testing the Flash file system, and later it was reused for two other testing projects [9]. Their conclusion was that initial efforts to develop an effective test system pay off in re-use on similar projects, because the significant differences were less important than the similarities. Their experiences confirms that we have chosen the right design decision in case of our model. It also points out that resources can be saved in the long run, by developing a modular and extensible testing framework, like ours.

6. Conclusion and future work

We presented a permissive model of a modular and extensible testing framework. The main contributions of the model are the followings:

- testing support for embedded languages at low cost. To add support for a new language, we only have to create a new *transformer*, the test data generation and the basic properties are already there.
- an unified interface to support different test data generation methods. The integration of a new test data generator is very easy, nearly "plug and play".

– using abstraction over different evaluation methodologies. This latter assures that programs written in different languages can be tested against each other.

The model essentially supports property based and differential testing, where the oracle problem is ported to the user.

Language and platform independency was an early design decision for the model. A real test of this would be to try to create another (non-Haskell) implementation of the model, maybe in an object-oriented or imperative programming language.

A possible future work is to extend the framework with new *transformer patterns* to support new programming languages. For Feldspar, it could be a reasonable goal to add support for testing against reference MatLab programs.

References

1. Axelsson, E., Dévai, G., Horváth, Z., Keijzer, K., Lyckegård, B., Persson, A., Sheeran, M., Svenningsson, J., Vajda, A.: Feldspar: A Domain Specific Language for Digital Signal Processing algorithms. In: Proc. Eighth MemoCode (2010)
2. Beck, K., Gamma, E.: Test infected: Programmers love writing tests. Java Report 3(7), 51–56 (1998)
3. Boyapati, R., Khurshid, S., Marinov, D.: Korat: Automated testing based on java predicates. In: In Proc. International Symposium on Software Testing and Analysis (ISSTA. pp. 123–133 (2002)
4. Brigham, E.O.: The Fast Fourier Transform (1974)
5. Claessen, K., Hughes, J.: Quickcheck: A Lightweight Tool for Random Testing of Haskell Programs. In: ACM SIGPLAN Notices. pp. 268–279 (2000)
6. Dévai, G., Tejfel, M., Gera, Z., Páli, G., Nagy, G., Horváth, Z., Axelsson, E., Sheeran, M., Vajda, A., Lyckegård, B., Persson, A.: Efficient Code Generation from the High-level Domain-specific Language Feldspar for DSPs. In: Proc. ODES-8: 8th Workshop on Optimizations for DSP and Embedded Systems, assoc. with IEEE/ACM International Symposium on Code Generation and Optimization (CGO) (2010)
7. Elliott, C., Finne, S., Moor, O.d.: Compiling embedded languages. In: Proceedings of the International Workshop on Semantics, Applications, and Implementation of Program Generation. pp. 9–27. SAIG '00 (2000)
8. Grima, M., Pace, G.J.: An embedded geometrical language in haskell: Construction, visualisation, proof (2007)
9. Groce, A., Holzmann, G., Joshi, R.: Randomized differential testing as a prelude to formal verification. In: Proceedings of the 29th international conference on Software Engineering. pp. 621–631. ICSE '07 (2007)
10. Hudak, P.: Modular domain specific languages and tools. In: Proceedings of the 5th International Conference on Software Reuse. pp. 134–142. ICSR '98 (1998)
11. Hudak, P.: Building domain-specific embedded languages. ACM Comput. Surv. 28 (December 1996)
12. HUnit: Haskell unit testing. http://hunit.sourceforge.net/ (2012)
13. Kamin, S.N.: Research on domain-specific embedded languages and program generators. In: Electronic Notes in Theoretical Computer Science (1998)

14. Koopman, P., Alimarine, A., Tretmans, J., Plasmeijer, R.: Gast: Generic automated software testing. In: The 14th IFL'02, Selected Papers, volume 2670 of LNCS. pp. 84–100 (2002)
15. Leijen, D., Meijer, E.: Domain specific embedded compilers. SIGPLAN Not. 35, 109–122 (December 1999)
16. McKeeman, W.M.: Differential testing for software. Digital Technical Journal 10(1), 100–107 (December 1998)
17. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37, 316–344 (December 2005)
18. Palka, M.H., Claessen, K., Russo, A., Hughes, J.: Testing an optimising compiler by generating random lambda terms. In: Proceedings of the 6th International Workshop on Automation of Software Test. pp. 91–97. AST '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1982595.1982615
19. Renggli, L., Girba, T., Nierstrasz, O.: Embedding languages without breaking tools. In: In ECOOP 2010: Proceedings of the 24th European Conference on Object-Oriented Programming (2010)
20. Runciman, C., Naylor, M., Lindblad, F.: Smallcheck and lazy smallcheck: automatic exhaustive testing for small values. In: Proceedings of the first ACM SIGPLAN symposium on Haskell. pp. 37–48. Haskell '08 (2008)

Dániel Leskó is pursuing his Ph.D. in Test data generation and static analysis at Eötvös Loránd University in Hungary, where he received his B.Sc. and M.Sc. degree in software technology in 2008 and in 2010. His research interests include functional programming, compilers, automated testing and test data generation.

Máté Tejfel is an assistant professor at Eötvös Loránd University, Faculty of Informatics, Department of Programming Languages and Compilers. His main research topics are parallel programming, functional programming and verification.

Received: January 15, 2013; Accepted: September 10, 2013.

DOI: 10.2298/CSIS130115070M

Extending Programming Language to Support Object Orientation in Legacy Systems

Hemang Mehta, S J Balaji, and Dharanipragada Janakiram

Department of Computer Science and Engineering,
Indian Institute of Technology Madras,
Chennai 600036, India
{hemang,sjbalaji,djram}@cse.iitm.ac.in

Abstract. The contemporary software systems written in C face maintainability issues because of tight coupling. Introducing object orientation can address these problems by raising the abstraction to objects, thereby providing better programmability and understandability. However, compiling a C software with a C++ compiler is difficult because of the incompatibilities between C and C++. Some of the incompatibilities such as designated initializers are nontrivial in nature and hence are very difficult to handle by automation such as scripting or by manual efforts. Moreover, runtime support for features such as global constructors, exception handling, runtime type inference, etc. is also required in the target system. Clearly, the traditional procedural language compiler cannot provide these features. In this paper, we propose extending programming language such as C++ to support object orientation in legacy systems instead of completely redesigning them. With a case study of Linux kernel, we report major issues in providing the compile and runtime support for C++ in legacy systems, and provide a solution to these issues. Our approach paves the way for converting a large C based software into C++. The experiments demonstrate that the proposed extension saves significant manual efforts with very little change in the g++ compiler. In addition, the performance study considers other legacy systems written in C and shows that the overhead resulting from the modifications in the compiler is negligible in comparison to the functionality achieved.

Keywords: g++, programming language, Linux kernel, legacy systems, object orientation.

1. Introduction

Many well-known large scale software systems such as Linux kernel[5], Apache webserver[7], PostgreSQL[6], etc. have been programmed in procedural language C. As these systems evolve with time, they become prone to issues related to cohesion and coupling. These issues make the systems difficult to maintain and reduce the understandability of the code. For example, Linux kernel has undergone studies[12,11] which reveal that it is a tightly coupled system and the instances of common coupling are increasing exponentially with new versions. Our previous work on object oriented(OO) wrappers[8] shows that the introduction of minimal OO features can help increase the maintainability. There are two main challenges when introducing OO concepts in a procedural language based system. The first is to compile all C files with a C++ compiler. The other challenge is to provide runtime support for features such as invoking of constructors for global and static objects.

Though C++ is perceived as a superset of C language, there are many incompatibilities between them which restrict compilation of the legacy system with a C++ compiler. The incompatibilities between C and C++ can be classified in trivial and nontrivial categories. The trivial issues can be addressed using scripting (e.g. renaming C++ keywords used as identifiers) or can be resolved manually if there are limited instances (e.g. pascal style function definitions). On the other hand, nontrivial issues can not be easily solved by the same means. An example of such incompatibility is support for nontrivial designated initializers.

Designated Initializers(DI) are used for initializing complex datatypes such as structures and arrays as shown in the following code snippet.

```
struct book { char name[]; int pages; }book_var = {
              ``Programming'', 200 };
       int arr[5] = { 10, 12, 21, 25, 32 };
```

Support for trivial DI was included in C89 standard[1] and C++ standard[3] also conforms to the same. On the other hand, nontrivial DI were introduced in C99[2]. The nontrivial DI provides the following features on top of trivial DI:
Labeled Initializing: Structure members can be selectively initialized out of the order in which they are defined.
Indexed Initializing: Assign values to specific array members using their indexes.
Ranged Initializing: specific range of an array can be initialized.
These features grant increased flexibility of initialization to the programmers. In addition, the absence of explicit constructors for structure variables leads to extensive and complex usage of DI (We shall use term 'DI' for nontrivial designated initializers henceforth.) in a C based software. Since C++ standard does not include DI, compiling a software written in C with g++ is not possible.

In this paper, we present an approach to extend g++ so that it recognizes the ranged, labeled, indexed initialization and nesting of them. With the help of a case study of Linux kernel, we show that it recognizes numerous instances of DI in the kernel and saves significant efforts. The main challenges we envisage are:
1. Different types of DI for structures and arrays,
2. Different combinations of types used in a single initialization and
3. Nested usage of DI in macro preprocessors.

Additionally, we explain how runtime support for C++ can be added in a legacy system with an example of Linux kernel.

The rest of the paper is organized as follows: Section 2 explains the motivation behind extending g++, as compared to other possible approaches. The design and implementation of the g++ extension are presented in Section 3. Section 4 describes the usefulness of the proposed approach using Linux kernel as a case study and also presents a performance study of the same. In Section 5, we explain how the runtime support for global constructors and volatile typecasting was included in the Linux kernel. The concluding remarks with future working directions are presented in Section 6.

2. Motivation

The first motivation of extending g++ was the absence of support for DI in g++ compiler. It is not included in the latest C++ standard [4] that was developed in 2011. To the best of our knowledge, there have been no attempts in the literature to explore this area. In case of C++, the complex datatype widely used is class and constructors are used to initialize objects. There was no need previously to statically initialize structure variables because of this, and hence the need for supporting DI was not felt in C++. However, with Linux kernel, we need some structure variables to be initialized at compile time since their initial state is required for system booting. The constructors are ill-suited in this case as they are called only after basic system initialization is complete. Secondly, many systems are implemented with both C and C++ like MySQL[10] and Windows kernel[9]. The primary reason behind this is that C++ provides higher abstractions in form of objects and many other useful features such as inheritance, polymorphism, templates, etc. If g++ could compile DI, the efforts in the development of these system can be significantly reduced. Thus, the primary objective is to make g++ recognize DI. We explain different possible approaches to tackle this issue. We motivate the need of extending g++ as a solution by comparing them with those solutions.

2.1. List Initialization

C++ standard supports list initialization of structure variables. The list method allows assigning values to *all* members of a structure in the *exact order* in which they are defined in the structure. Hence one approach is to replace labeled DI with list method using an automatic process such as scripting. However, converting labeled DI to list construct has two major issues. Firstly, the uninitialized members of the structure should be assigned their default values. Thus the script performing the replacement has to infer the datatypes of the member variables and their default values. Secondly, the values of members to be initialized have to be ordered. This becomes difficult especially when there is a nesting of DI (i.e. a member of a structure is also a structure and is initialized using the DI) and the nested element is initialized with a macro.

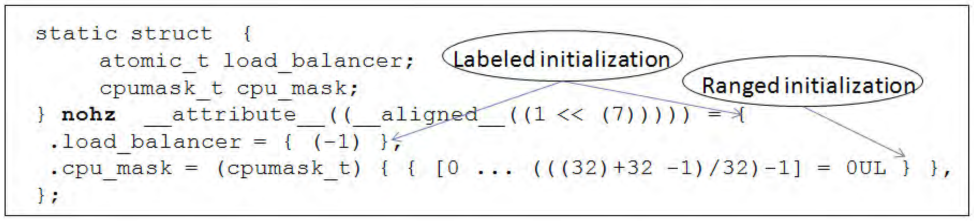
Apart from the labeled DI for structures, indexed and ranged initialzers are also required to be replaced by the list method. The list method is ill-suited if the array is very large and uses indexed or ranged methods.

2.2. Constructor for Structures

Another way to make g++ compatible with DI is to replace usages of DI with constructors for structures. However, this method adds another function call at runtime for initialization of structure members adding to overhead. If these variables are in global scope, then there are two issues. The first is that static (compile time) initialization is not possible, which is a requirement in case of Linux kernel. This is because when the system boots, some global system variables should be initialized before global and static constructors are called. Moreover, the order among global constructors can not be guaranteed. This means that if an uninitialized global structure variable is being used in another initialization, it may lead to system crash. Finally, this method is not suitable for array initialization and hence the issues of ranged and indexed initializations remain unresolved.

2.3. Extending g++

We examined two different approaches other than extending the C++ compiler. We found that both the approaches are difficult in implementation as well as in verification of their correctness. This is because large systems like Linux kernel have various ways in which DI are used and it is a cumbersome process to examine all of them. An example of one such usage of DI in Linux kernel is shown in Figure 1.



```
static struct  {
     atomic_t load_balancer;
     cpumask_t cpu_mask;
} nohz  __attribute__((__aligned__((1 << (7))))) = {
 .load_balancer = { (-1) },
 .cpu_mask = (cpumask_t) { { [0 ... (((32)+32 -1)/32)-1] = 0UL } },
};
```

Fig. 1. A precompiled code snippet from `kernel/sched.c`, the scheduler of Linux kernel. It shows nesting of labeled(`load_balancer`) and ranged initializer(`cpu_mask`) with labeled initializer(`nohz`).

The automation used for one software may not work for other softwares as the use cases may be different for them. In this way, changing the C++ compiler is a practical and easy solution to the problem. The proposed extension of g++ allows out-of-order and selective initialization of members. It also facilitates static and global initialization as it is done by gcc for any C based system. In addition, it is independent of the target software and does not involve any effort on the developer's part to apply any script or perform any kind of manual modification. Finally, as will be explained in Section 3, it is relatively simple and involves changing only 3 files of g++ source code.

3. Design and Implementation

This section explains the design and implementation of the extension to support designated initializers in C++ compiler. We have designed the extension with g++ version 4.4.5 as the base compiler. The proposed design primarily involves recognizing DI and performing corresponding semantic actions. Though this process spans across only 3 files, identifying the places to modify required careful analysis of the compiler code. The files to be modified for the implementation of the design include `parser.c`, `decl.c` and `typeck2.c`. They are part of the g++ branch of gcc compiler source tree(`gcc/cp/`). Each file represents a phase which the code being compiled passes through. In this section we explain the extended functionality of each phase to recognize all 3 DI types.

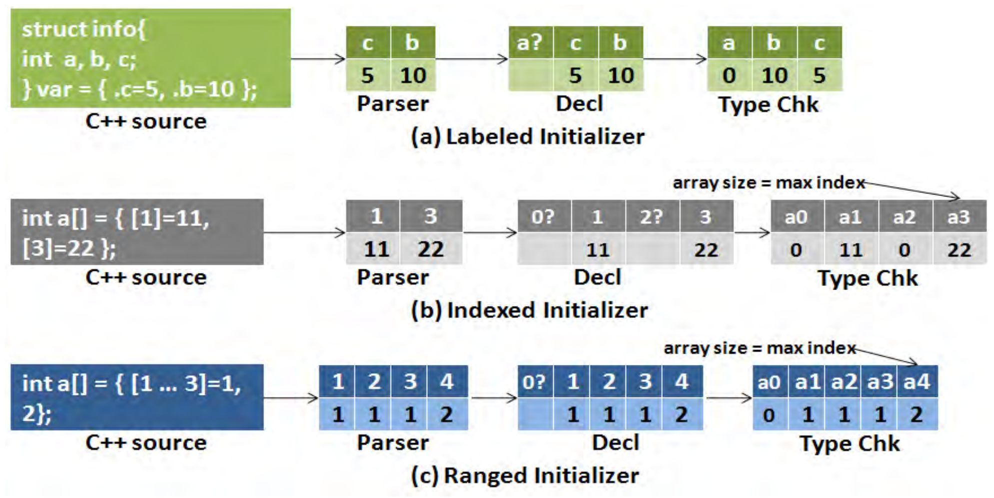


Fig. 2. The g++ extension design to recognize labeled, indexed and ranged Initializer involving parser, declarator and type-checker.

Parser: Since C++ standard does not include DI, the original parser throws syntax error when it encounters out of order labeled initializer. In order to support the same, we have added grammar rules in g++ parser. These rules recognize the signature patterns of DI and store initializer (values) and identifiers (member variables) in a list. This list is known as unprocessed vector, which is a fixed size array. The outcome of this process is shown in column 1 of Figure

2 (corresponding to the first phase). The parser passes the vector to declarator of g++.

Declarator: We have modified the declarator to process the unprocessed vector provided by the parser based on the type of initializer. The declarator validates the vector by checking if any identifiers are left uninitialized. It creates entries for uninitialized identifiers in the vector and marks them as 'erroneous' (See Figure 2, column 2). The partially processed vector is then passed on to the type checker.

Type-checker: The type checker has been refactored to consume the vector passed by the declarator and to perform final processing on the same. It infers the type of the identifiers of the erroneous entries for structures. It assigns default values to such identifiers with the assistance of back-end of the g++ compiler. The basic datatypes like numbers are assigned 0, pointers are assigned NULL, characters are assigned '\0' and boolean identifiers are assigned 0-bit. On the other hand, derived datatypes are broken into basic datatypes based on their members and the same procedure is followed.

For array (indexed and ranged) initializers, type-checker scans the initialization list till the end when explicit size is not provided. Then it allocates memory of the size according to the maximum index specified in the initialization. All erroneous entries are filled with 0. This accomplishes the processing of vector and the values are copied to the actual structure / array members as shown in the last phase of Figure 2.

4. Evaluation

The evaluation of the proposed g++ extension is divided into two parts. The first part explains how the extension reduces manual efforts by using Linux kernel as a case study. The second part presents the comparison of both original and extended compiler by measuring their performances.

4.1. Case Study: Linux Kernel

We explain the evaluation of the proposed extension to g++ in this section using Linux kernel as a case study. Linux kernel is a large software written in C, which makes extensive use of DI to initialize its global variables. We measure the increased productivity and ease of porting Linux kernel to C++ by counting the number of occurrences of DI in different subsystems.

We have extended gcc version 4.4.5 to count the number of DI instances everytime it encounters one. Table 1 shows the counter values calculated after the compilation of Linux kernel 2.6.23 with the modified compiler.

The experiment shows that more than 5600 variables are initialized using DI in core kernel and other subsystems of the Linux kernel. This result shows that making manual modifications is not a feasible solution. We have already seen how complex the usage of DI in the kernel can be, which renders scripting ineffective as an option. On the other hand, our extension only modifies 238

Table 1. Number of instances of designated initializers of different types in Linux kernel 2.6.23 as counted by the g++ extension (Core includes management of processes, timers, scheduling, etc.)

Kernel Subsystems	Labeled	Indexed	Ranged	Total
Core	750	30	109	889
Memory	165	1	2	168
Network	1415	0	225	1640
File System	509	0	41	550
Architecture	300	16	90	406
Device Drivers	1818	7	124	1949

lines (including addition, removal and modification) of the compiler source code. Thus, our exploration in modifying g++ is justified by the results.

4.2. Performance Study

Performance is one of the key concerns when a system such as compiler is refactored. In order to discover the overhead that results from the proposed extension of g++, we compared building times for 3 different systems; Linux kernel, Apache HTTPD (Web) server and PostgreSQL database. Our objective here was to ensure that performance is not sacrificed in order to gain more functionality. Additionally, this experiment verifies how the new compiler compares with the original one while compiling other systems than the Linux kernel.

The base system for the experiments consisted of Intel Core i7 quad core CPU at 2.4 GHz, 6GB RAM and Fedora 13 operating system. gcc compiler version 4.4.5 was used as original compiler and the same version was modified as explained earlier in the paper. The Linux kernel version was 2.6.23 while the versions of PostgreSQL and HTTPD were 9.2.4 and 2.4.6 respectively. The compilation times for building these systems were obtained using `time` utility of Linux to the precision of millisecond.

Table 2. Comparison of building (compilation) times (in seconds) for Linux kernel, Apache HTTPD server and PostgreSQL database

System Name	Extended Compiler	Orignial Compiler
Linux Kernel	774.060	774.039
Apache HTTPD	75.938	75.940
PostgreSQL	211.381	211.384

Table 2 summerizes the results of the experiment carried out for comparing the performance of both versions of compiler. It is evident that Linux kernel is the largest among all systems that were tested, as it takes longest to build. The extended compiler takes slightly longer time for Linux kernel than the original one since it has numerous instances of nontrivial designated initializers.

However, the overhead in this case is in order of milliseconds, which is a very small fraction of the total time taken for compiling the kernel and hence can be considered to be reasonable.

On the other hand, HTTPD and PostgreSQL are compiled in almost the same time by both the compiler versions. Again, the reason being the absence of nontrivial designated initializers. Thus, it can be observed that the modifications in the compiler do not have any adverse effect on the compilation of the software that does not utilize nontrivial designated initializers heavily.

5. Runtime Support for g++ in Linux Kernel

The Linux kernel needs built-in library support for basic operations since it is the only code in execution during the bootstrap of the system and it can not use any runtime linking for library functions. Hence certain C library functions and runtime have been included in Linux source tree at `lib/` directory. In order to include runtime support for g++, we added the necessary files from g++ source to this location. This section explains the issues that arose during this process and how they were addressed.

5.1. Volatile Typecasting of Complex Types for C++

Background: In some cases, certain compiler optimizations are a hindrance to the functional objective of the program. Usage of memory mapped I/O in Linux kernel is one such instance. In memory mapped I/O, an I/O device is mapped to a memory location. Accessing that location results in read/write operation on that device. However, compiler optimizes that location to be accessed from cache memory only and the operation does not happen on the device.

Problem: In order to stop compiler from optimizing operations on certain variables (memory locations), they are typecast as `volatile` in C. Linux kernel uses this mechanism very often. It uses `ACCESS_ONCE` macro to accomplish this task. Following is the definition of `ACCESS_ONCE` macro.

```
#define ACCESS\_ONCE(x) (*(volatile typeof(x)*)&(x))
```

This definition works well in C compiler for complex datatypes such as structures. However, this definition only works for basic datatypes in g++.

Solution: We have extended this definition to make it compatible with g++ using runtime type inference (RTTI) and reinterpret casting. Basically the central idea of the solution is as follows:
1. Infer the type of data at runtime using `typeid` construct of g++.
2. If the datatype is basic, the C style definition should be used.
3. In case the datatype is complex, reinterpret casting should be used.

The reinterpret cast, as defined by the C++ standard [4], allows casting of a pointer to any other (including unrelated) type. Additionally it ensures that if the pointer is cast back to the original type, its value is preserved. This is achieved by reinterpreting the bit pattern of the value to the target type. Thus, when used with `volatile`, reinterpret cast treats the variable as `volatile`, and directs the compiler not to apply any cache optimization on the variable.

The definition of `ACCESS_ONCE` macro, according to the proposed solution, is shown below:

```
#ifdef __cplusplus  /*g++ compiler*/
#include <iostream>
#include <typeinfo>
using namespace std;
#define ACCESS_ONCE(x)                              \
  (             typeid(x).name()[0] == 'P' ||    \
                typeid(x).name()[0] == '1' ||    \
                typeid(x).name()[0] == '2')      \
  ? reinterpret_cast<volatile typeof(x) &>(x)    \
  : (*(volatile typeof(x)*)&(x))
#else /*C compiler*/
#define ACCESS_ONCE(x) (*(volatile typeof(x)*)&(x))
#endif
```

In this definition, `typeid()` and `typeof()` are RTTI constructs, which explains the necessity of runtime support for C++ in Linux kernel. It returns ‘P’, ‘1’ or ‘2’ in case of pointers, structures and classes respectively, implying that the identifier is of complex type and reinterpret cast should be used to make it `volatile`.

5.2. Global Constructors

Background: The constructors for global and static objects are usually called by a special function named `__do_global_ctors_aux (void)`, which is inserted by g++ compiler in the linked object file. It is called before the `main()` function and thus before any possible usage of the global/static objects. Similarly, destructors are called after `main()` using a special function which is inserted by g++ compiler. In order to achieve this, a compiled object file is linked with `crtbegin.o` and `crtend.o` files. These pre-generated files are used by g++ to traverse through the given file to find global and static objects. For each such object its constructor and destructor are placed in the lists of global constructors and destructors respectively. The starting and ending of each list is denoted by g++ compiler variables `__CTOR_LIST` and `__CTOR_END` in case of constructors; and `__DTOR_LIST`, `__DTOR_END` in case of destructors. `__do_global_ctors_aux` function traverses the constructor list in downward fashion, i.e. from `__CTOR_END` to `__CTOR_LIST`.

Problem: For normal application programs, this is handled by g++ and the linker automatically. However, for Linux kernel, we have to provide this runtime support. Hence we had to add that support in Linux kernel by adding

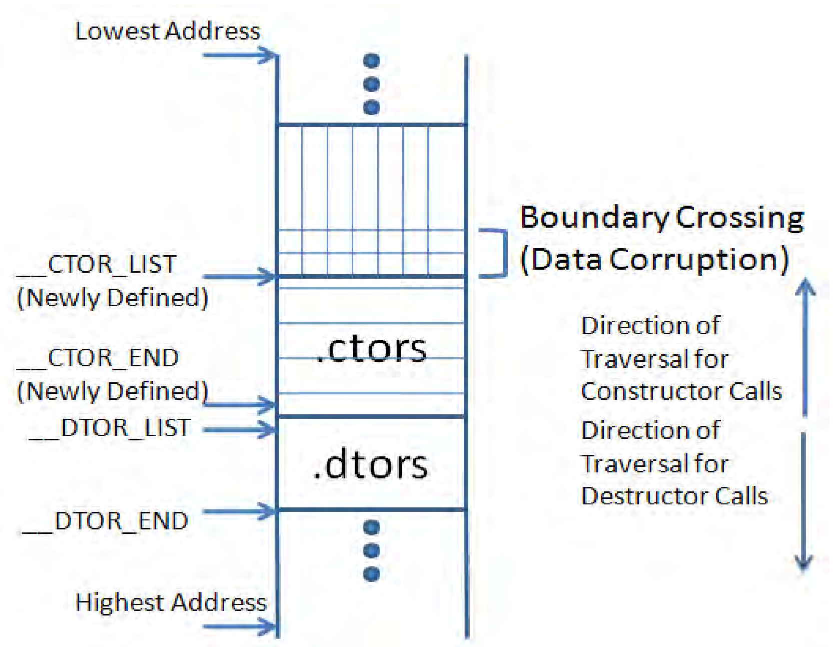


Fig. 3. The memory layout of vmlinux kernel image depicting boundary crossing of .ctors section and its solution

`crtstuff.c` file and `libstdc++` directory from source of g++. We also made suitable changes in the kernels makefiles at different levels, so that a C++ file can be compiled with g++ compiler. However, in the existing g++ files for that support, while traversing through constructor list, the initial boundary (`__CTOR_LIST`) was getting missed. This led to function `__do_global_ctors_aux` getting into the previous section of constructors. This resulted in execution of non-executable data and subsequently crashing of kernel.

Solution: Figure 3 shows the layout of `vmlinux`, the kernel image in the memory. It is an `elf` image that is made of different sections. We have added a new boundary for `__CTOR_LIST`, the beginning of the constructor section which `__do_global_ctors_aux` function checks when it iterates through the list. We have added this symbol as a kernel image (`vmlinux`) symbol, which g++ compiler can access in `__do_global_ctors_aux` function. The actual definition of the symbol is as follows:

```
#ifdef CONFIG_CONSTRUCTORS
#define KERNEL_CTORS() . = ALIGN(8);                      \
                       VMLINUX_SYMBOL(__ctors_start) = .; \
                       *(.ctors)                          \
                      VMLINUX_SYMBOL(__ctors_end) = .;
#else
#define KERNEL_CTORS() VMLINUX_SYMBOL(__ctors_start) = .; \
                      *(.ctors)                          \
                      VMLINUX_SYMBOL(__ctors_end) = .;
#endif
```

6. Conclusions and Future Work

This paper presented how object orientation can be supported in a large scale system such as Linux kernel by extending g++ compiler. As a part of compile-time support, a g++ extension for nontrivial designated initializers(DI) for structures and arrays was added. It handles usage of ranged, indexed, labeled and nesting of all types of DI in an application transparently. Furthermore, the paper showed how global constructors and volatile typecasting in C++ can be supported in Linux kernel. Finally the experiments proved that the proposed approach saves a lot of manual efforts with a very reasonable overhead.

We envisage that an automated tool for converting legacy systems written in C into C++ can be well appreciated by the software engineering community. At present, the proposed approach has limited features and there are still many incompatibilities between C and C++ that require addressing. In future, this tool can be made more enhanced and sophisticated by integrating the modified compiler with scripting support to tackle these issues. This tool can be used to cater to specific issues of other legacy systems, as opposed to just Linux kernel. At a later stage, this work can be extended to raise abstractions from objects to services in legacy systems. The services are more abstract than procedures or objects and hence are independent of the language they are implemented in, which can make maintenance of the legacy systems easier.

Acknowledgments. The authors acknowledge Prateek Dhawaalia, IIT Madras for his inputs in implementation of the g++ extension. The authors also thank DIT, Government of India for financially supporting this work.

References

1. ISO/IEC 9899:1990 - Programming languages - C (1989), `http://www.iso.org/iso/catalogue_detail.htm?csnumber=17782`
2. ISO/IEC 9899 - Programming languages - C (1999), `http://www.open-std.org/jtc1/sc22/wg14/www/standards.html#9899`
3. ISO/IEC 14882:1998 - Programming languages – C++ (2005), `http://www.iso.org/iso/catalogue_detail.htm?csnumber=25845`
4. ISO/IEC 14882:1998 - Programming languages – C++ (2011), `http://www.iso.org/iso/catalogue_detail.htm?csnumber=50372`
5. Beck, M., Bohme, H., Kunitz, U., Magnus, R., Dziadzka, M., Verworner, D.: Linux kernel internals. Addison-Wesley Longman Publishing Co., Inc. (1996)

6. Douglas, K.: PostgreSQL. Sams (2005)
7. Fielding, R., Kaiser, G.: The apache http server project. Internet Computing, IEEE 1(4), 88–90 (1997)
8. Janakiram, D., Gunnam, A., Suneetha, N., Rajani, V., Reddy, K.V.K.: Object-oriented wrappers for the Linux kernel. Software Practice & Experience 38(13), 1411–1427 (2008)
9. Solomon, D.A., Custer, H.: Inside Windows NT. Microsoft Press, Redmond, WA, USA, 2nd edn. (1998)
10. Widenius, M., Axmark, D.: MySQL reference manual: documentation from the source. O'Reilly Media, Inc. (2002)
11. Yu, L., Schach, S.R., Chen, K., Heller, G.Z., Offutt, J.: Maintainability of the kernels of open-source operating systems: A comparison of Linux with FreeBSD, NetBSD, and OpenBSD. Journal of Systems and Software 79(6), 807–815 (2006)
12. Yu, L., Schach, S.R., Chen, K., Offutt, J.: Categorization of common coupling and its application to the maintainability of the Linux kernel. IEEE Transactions Software Engineering 30, 694–706 (October 2004)

Hemang Mehta is an MS research scholar at Department of Computer Science and Engineering, Indian Institute of Technology Madras, India. His research interests include design of operating systems, distributed systems and compilers. Specifically, his work focuses on applying principles of service oriented computing to improve the design of operating systems.

S J Balaji is an MS student of Computer Science and Engineering at Indian Institute of Technology Madras. He received a BE degree in Electronics and Telecommunication Engineering from Mumbai University in 2010. His research interests are operating systems design, distributed systems, cloud-based systems, energy aware system designs and manycore operating systems.

Dharanipragada Janakiram is currently a professor in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Madras, India, where he heads and coordinates the research activities of the Distributed and Object Systems Lab. He obtained his Ph.D from IIT, Delhi. His current research involves building large scale distributed systems focusing on design pattern based techniques, measurements, peer-peer middleware based grid systems, cloud bursting, etc. He is currently an associate editor of IEEE Transactions on Cloud Computing, the SIG Chair of Distributed Computing of Computer Society of India, Chair of ACM Chennai Chapter and is also the founder of the Forum for Promotion of Object Technology in India.

Received: January 15, 2013; Accepted: August 20, 2013.

DOI: 10.2298/CSIS130120071K

Context Parsing (Not Only) of the Object-File-Format Description Language

Jakub Křoustek and Dušan Kolář

Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno University of Technology, Božetěchova 1/2, 612 66 Brno, Czech Republic
{ikroustek,kolar}@fit.vutbr.cz

Abstract. The very first step of each tool such as linker, disassembler, or debugger is parsing of an input executable or object file. These files are stored in one of the existing object file formats (OFF). Retargetable tools are not limited to any particular target platform and they have to deal with handling of several OFFs. Handling of these formats is similar to parsing of computer languages — both of them have a predefined structure and a list of allowed constructions. However, OFF constructions are heavily mutually interconnected and they create context-sensitive units. In present, there is no generic system, which can be used for OFF description and its effective parsing.

In this paper, we propose a formal language that can be used for OFF description. Furthermore, we present a design of a context parser of this language that is based on the formal models. The major advance of this solution is an ability to describe context-sensitive properties on the level of the language itself. This concept is planned to be used in the existing retargetable decompiler developed within the Lissom project. In this project, the language and its parser will be used for an object file parsing and its automatic conversion into the internal uniform file format. It is important to say that the concept of this parser can be utilized within other programming languages.

Keywords: object file format, context parsing, scattered context grammar, priority function, attributed grammar, decompilation, Lissom, ELF

1. Introduction

Reverse compiler (i.e. decompiler) is yet another tool that takes executable files on its input. Its purpose is to translate this input into a high level language (HLL) representation, such as a C code. This tool can be used for source code reconstruction, binary code migration, malware analysis, etc. Retargetable decompilation is a more difficult task because it must handle all the platform, operating system, and programming language specific features.

Platform-specific decompilation is a well-described discipline, e.g. see [5,7,40]. On the other hand, retargetable (i.e. platform-independent) decompilation is still a quite unexplored area, despite the first attempts done decades ago. However, several steps of retargetable decompilation have been already done, such as uniform extraction of instruction semantics, machine-code decoding, reverse compilation into HLL, etc. See [43,44] for details.

However, there is still one non-covered phase of retargetable decompilation — the handling and conversion of platform-dependent object file formats (OFFs). This is the preliminary step of decompilation. Within this step, an input file is being analyzed, validated, and converted into the internal, uniform representation. This conversion transforms all the necessary information (e.g. machine code, data, symbols) into the internal structures. It might be possible to use one of the existing single-purpose converters or write a new one from a scratch. However, none of these is a truly generic solution.

This problem can be divided into two tasks. (1) Description of a target OFF using a specific description language. (2) Automatic generation of an OFF converter based on this description. Afterwards, the converter can be used for conversion of applications stored in a particular OFF into an internal code representation used by a decompiler.

Structure of most OFF is relatively complicated (e.g. Windows PE, UNIX ELF) because its elements are mutually interconnected and the structure is heavily influenced by content of these elements. We can say that these elements create a context-sensitive behavior. This is a problem for design of such OFF description language because the theory of computer-language compilation settled down on the concept of context-free parsing for most of the existing languages during the last sixty years.

Within the context-free parsing concept, syntax of programs is usually processed using automatically generated context-free parsers. Parser generators like YACC, Bison, or ANTLR are able to create a skeleton of target language-specific parser. However, this skeleton has to be enriched of hand-written HLL code implementing semantics checking (so called semantic actions). This concept is prone to errors and each change of the target language needs reimplementation of the parser (at least its semantic actions).

In this paper, we present a new formal language for the description of OFFs that is capable to describe context-sensitive elements. We propose a context parser of this language that is based on the newly created formal models (attributed scattered context grammar with priority function, etc.).

The concept is planned to be used in the existing retargetable decompiler developed within the Lissom project [23]. In this project, the OFF language is used for object-file handling and its automatic conversion into the internal Common-Object-File-Format (COFF)-based file format, which is processed by the decompiler afterwards, see [45] for details. Moreover, the language is designed to be general enough for usage in other retargetable tools (e.g. loaders, disassemblers, debuggers) and the context parser can be used for parsing of different programming languages, not just OFF description language.

The paper is organized as follows. Section 2 introduces some preliminaries. Section 3 briefly characterizes common OFFs. Then, we discuss existing conversion techniques and applications in Section 4. The Lissom project is briefly described in Section 5. Our language for OFF description is presented together with an example of its usage in the subsequent Section 6. Within this section, we also depict several context-sensitive features of this language. In Section 7, we present a concept of the context parser as well as the definition of the new formal models that the parser is based on. We also give a short overview on the current state of the parser's implementation together with experimental results within the same section. Finally, discussion of future research closes the paper in Section 8.

2. Preliminaries and Definitions

We assume a reader is familiar with the formal language theory (for further reference, see for example [26]).

Definition 1. A *phrase-structure grammar* is a quadruple

$$G = (V, T, P, S),$$

where

- V is a *total alphabet*;
- $T \subset V$ is a finite set of *terminal symbols* (*terminals*);
- $S \in V - T$ is the *start symbol* of G;
- P is a finite *set of productions* $p = x \rightarrow y$, $x \in V^*(V-T)V^*$, $y \in V^*$.

The symbols in $V-T$ are referred to as *nonterminal symbols* (*nonterminals*). We set lhs$(p) = x$ and rhs$(p) = y$, which represents the *left-hand side* and the *right-hand side* of the production p, respectively.

Definition 2. A *context-sensitive grammar* (CSG) is a phrase-structure grammar

$$G = (V, T, P, S),$$

such that every production $p = x \rightarrow y \in P$ satisfies $|x| \leq |y|$.

Definition 3. A *context-free grammar* (CFG) is a phrase-structure grammar

$$G = (V, T, P, S),$$

such that every production $p = x \rightarrow y \in P$ satisfies $A \rightarrow x$, where $A \in V-T$ and $x \in V^*$.

Definition 4. A *scattered context grammar* (SCG, see [11]) is a quadruple,

$$G = (V, T, P, S),$$

where

- V is a total alphabet;
- $T \subset V$ is a finite set of terminals;

– $S \in V - T$ is the start symbol;
– P is a finite set of productions of the form

$$(A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n),$$

where $A_i \in V - T$, $x_i \in V^*$ for all $i : 1 \leq i \leq n$.

Definition 5. A *propagating scattered context grammar* (PSCG) is a SCG

$$G = (V, T, P, S),$$

in which every $(A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n) \in P$ satisfies $x_i \in V^+$ for all $i : 1 \leq i \leq n$.

Definition 6. Let $G = (V, T, P, S)$ be a (propagating) SCG. If

$$y = u_1 A_1 u_2 \ldots u_n A_n u_{n+1},$$
$$z = u_1 x_1 u_2 \ldots u_n x_n u_{n+1},$$

and $y, z \in V^*$, $p = (A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n) \in P$, then y *directly derives* z in the SCG G according to the production p,

$$y \Rightarrow_G z\ [p] \text{ (or simply } y \Rightarrow_G z\text{)}.$$

Let $\Rightarrow_G^+$ and $\Rightarrow_G^*$ denote the *transitive* and the *reflexive-transitive closure* of $\Rightarrow_G$, respectively. To express that G makes the *derivation* from u to v by using the sequence of productions $p_1, p_2, \ldots, p_n \in P$, we write $u \Rightarrow_G^* v\ [p_1 p_2 \ldots p_n]$ (or $u \Rightarrow_G^+ v\ [p_1 p_2 \ldots p_n]$ to emphasize that the sequence is non-empty). We abbreviate $\Rightarrow_G$ to $\Rightarrow$ when it is clear which grammar we are referring to. This definition also holds for other SCG-based grammars listed below.

Now we are able to define scattered context grammars regulated by priority functions, see [21] for details of their properties.

Definition 7. A *(propagating) scattered context grammar with priority*, abbreviated as ((P)SCGP), is a quintuple

$$G = (V, T, P, S, \pi),$$

where (V, T, P, S) is a (propagating) scattered context grammar and π is a *priority function*

$$\pi : P \rightarrow \mathbb{N}.$$

Definition 8. Let $G = (V, T, P, S, \pi)$ be a (P)SCGP. We say that y directly derives z in (P)SCG G according to the production p, $y \Rightarrow_G z\ [p]$ (or simply $y \Rightarrow_G z$), if and only if:

– $y = u_1 A_1 u_2 \ldots u_n\ A_n u_{n+1} \in V^*$,
– $z = u_1 x_1 u_2 \ldots u_n x_n u_{n+1} \in V^*$,
– $p = (A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n) \in P$, and

- there is no $p' = (A'_1, \ldots, A'_n) \to (x'_1, \ldots, x'_n) \in P$, such that:
   1. $y = u'_1 A'_1 u'_2 \ldots u'_n A'_n u'_{n+1} \in V^*$, and
   2. $\pi(p') > \pi(p)$.

Definition 9. A *(propagating) scattered context language with priority* is language generated by a (propagating) scattered context grammar with priority. The family of (propagating) scattered context languages with priority is denoted by $\mathcal{L}$((P)SCP). In [21], it has been proved that

$$\mathcal{L}(CS) = \mathcal{L}(PSCP) \subset \mathcal{L}(RE) = \mathcal{L}(SCP),$$

where RE stands for the set of all recursively enumerable languages.

3. Object File Formats

The term *object file format* refers to a format of an executable code, library code, or object code that has not been linked yet. In the following text, we focus mainly on the executable code. A generic OFF usually consists of the following parts [22]:

- *Header* – contains essential information about the file (e.g. its identification, size, section pointers);
- *Object code* – i.e. sections containing machine code and application data;
- *Relocations* – "*Relocation is the process of assigning load addresses to the various parts of the program, adjusting the code and data in the program to reflect the assigned addresses*" [22]. We can find a wide range of relocation types for each target architecture. Some relocations can be resolved during compilation by linker; while the other ones has to be resolved by loader before program's execution;
- *Symbols* – symbols are usually stored in tables and they characterize its local, imported, and exported symbols (variables, functions, etc.);
- *Debugging information* – generated by compilers for debug support. There exist several debugging information standards [20]. The presence of the debugging information is optional.

Unfortunately, there is no such generic format and each platform (i.e. a combination of an operating system and a processor architecture) has its own format, or a derivative of an existing one. In present, we can find two major OFFs — UNIX ELF [39] and Windows PE [29], see Fig. 1. However, other formats are on arise (e.g. Apple Mach-O), see [19]. In the Lissom project [23], a COFF-based file format is used for internal code representation. The overview of other common formats can be found in [19].

The **UNIX ELF** [39] file format is a standard on all UNIX-like systems. It is independent on a particular target architecture (e.g. Intel IA-32, SPARC, ARM). The leading part of the ELF file is a header with all the essential information. It also points to the program and section header tables. These tables contain information about particular segments and sections, respectively (e.g. their sizes, offsets within the file). Each section can store different content (e.g. code, data, symbol, hash tables); furthermore, one or more sections may form a segment.

From the linker point of view, an ELF file consists of a group of sections defined in a section-header table. Contrariwise, loader handles the ELF file as a group of segments defined in a program-header table, see 1. The very important characteristic of this format is its flexibility. Only the header has a fixed offset within the file, all other elements are optional, as well as their offsets within the file. Therefore, all elements are scattered throughout the file, and the size or content of padding is unspecified.

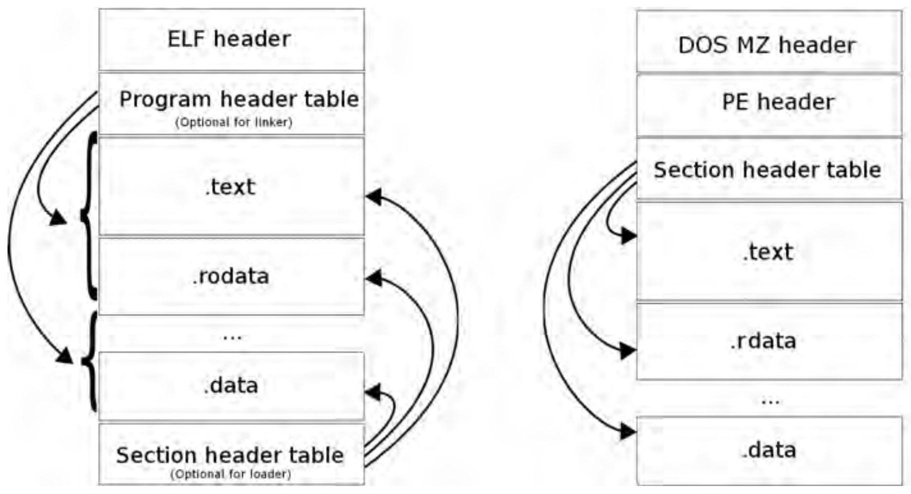


Fig. 1. ELF (on left) and Windows PE (on right) file formats.

The **Microsoft's Windows Portable Executable** (WinPE) [29] format also supports all the three file types — object files, executables, and libraries. Windows PE can be used on all Windows-based systems on architectures Intel IA-32, IA-64, x86-64, ARM, and others. The structure of the PE format is based on the COFF format [10]. It consists of a number of headers and sections that tell the loader how to map the file into memory. Each section has its own header and often a specific purpose, for illustration see 1. For example, the `.text` section holds the program code; `.data` sections hold global variables, `.edata` and `.idata` sections contain exported and imported symbols, etc.

The **E32Image** format is used on the Symbian operation systems, usually used in smartphones [31]. It was developed by Symbian Ltd., which currently belongs to Nokia. E32Image is used only on the ARM architecture.

E32Image is a proprietary format, and its specification has never been publicly published; therefore, all the necessary information was gathered using reverse engineering. This format was originally based on Windows PE, but since Symbian version 9.1 (in 2005), its authors switched to an ELF-like format. The

E32Image file is created from an existing executable PE or ELF file by a special post-linker. The main idea of this format is to provide basic file format structure with low-memory overhead. The differences over the mentioned formats include that the same type sections are merged and might be compressed, information about the target architecture (e.g. word size) is not explicitly encoded, and that unnecessary strings (e.g. symbol names) might be removed.

The **Mach-O** object file format [3] is used in operation systems Darwin, NeXTSTEP, Mac OS X, or iOS from Apple Inc. It is made up of three parts — Mach-O header, followed by a series of load commands, and one or more segments, each of them containing up to 255 sections. Mach-O supports Intel IA-32, x86-64, PowerPC, and PowerPC64 as the target architectures.

A special feature of this format is its support of multi-architecture binaries, where multiple Mach-O files can be combined in a single multi-architecture file. Such binary file contains code for multiple instruction set architectures.

4. Related Work

We can find several projects focused on parsing and binary conversion of OFFs. They are used mostly in reverse engineering or for code migration between particular platforms. The largest group consists of hand-coded tools that are focused on binary conversion between two particular OFFs.

A typical example is the Macintosh Application Environment project [2], which supports execution of native Apple Macintosh applications on UNIX based workstations. AT&T's FreePort Express [6] is another binary translator, which permits conversion of SunOS and Solaris executables into Digital UNIX executables. Wabi allows conversion of executables from Windows 3.x to Solaris [12].

Another important project is the Binary File Descriptor library (**BFD**) [4]. BFD was developed by the Cygnus Support company, and currently forms a part of the GNU Binutils package. It supports unified, canonical format for manipulation tens of OFFs (e.g. ELF, PE, COFF). BFD is used as a front-end of many existing projects, however, it is not a retargetable solution because support of each new OFF must be hand-coded. Furthermore, due to BFD's complexity, the interconnection of the target application and BFD is often difficult. Details about a successful BFD-based solution can be found in [19].

The last group of projects uses their own grammar-based systems for a formal description of binary formats. The architecture description language (ADL) **SLED** [32], developed within the New Jersey Machine Code Toolkit [33], is supposed to describe the instruction sets of target processor architectures, i.e. syntax and binary encoding of each instruction. Such description can be used for the automatic generation of retargetable linker [9], debugger [34], or other tools. However, this language does not support description of OFFs. We can find the same limitation in all common ADLs, see [25] for more details.

We can find two formalisms called **BFF grammar** (Binary File Format Grammar). The first one (**DWG BFF**) [8] was originally designed for description of non-executable file formats. More precisely, it was designed and tested only on the AutoCAD DWG format. This grammar is a state-of-the-art concept, which has never been implemented, nor used in any tool. The grammar is limited to the DWG format, but it can be be possibly used on other OFFs. Its author claims (see [8]) that the grammar is in LL(1) form and it can be parsed by the recursive descent approach.

The DWG BFF grammar inspired project UQBT [42] and its **SRL BFF** grammar [41]. Despite the limitations of the original DWG BFF grammar, it was simplified and used within this project for generation of the Simple Retargetable Loader (SRL). Although, it is claimed that this concept can be used for automatic generation of other retargetable tools, the grammar constructions are limited only for a simple loader. According to [41], the SRL was tested as an ELF loader for existing decompiler `dcc` [5].

Both BFF grammars have several limitations. For example, they are unable to properly model optional elements of OFFs, such as the missing section header table in ELF; relocation information is not taken into account, etc. The most significant drawback of both grammars is the lack of semantic actions [1] (e.g. semantic checks, validation of OFF content, user-defined actions), see Figure 2. Therefore, the grammars are only capable to describe syntactical structure of the input OFFs, but modeling of context-sensitive properties is left on the user.

```
DEFINITION FORMAT
    header
    program_header_table
    sections
    section_header_table
END FORMAT

DEFINITION header
    h_ident SIZE 16
    h_type SIZE 16
    h_machine SIZE 16
    h_version SIZE 32
    ...
END header
```

Fig. 2. Example of the BFF grammar description of the ELF format [42].

In conclusion, none of the previously mentioned concepts can be used for effective handling of OFFs for retargetable decompilation.

5. Lissom Project's Retargetable Decompiler

The Lissom project's [23] retargetable decompiler aims to be independent on any particular target architecture, operating system, or OFF. It consists of two main parts—the preprocessing part and the decompilation core, see Figure 3. Its detailed description can be found in [45,43]. The decompilation process consists of the following phases.

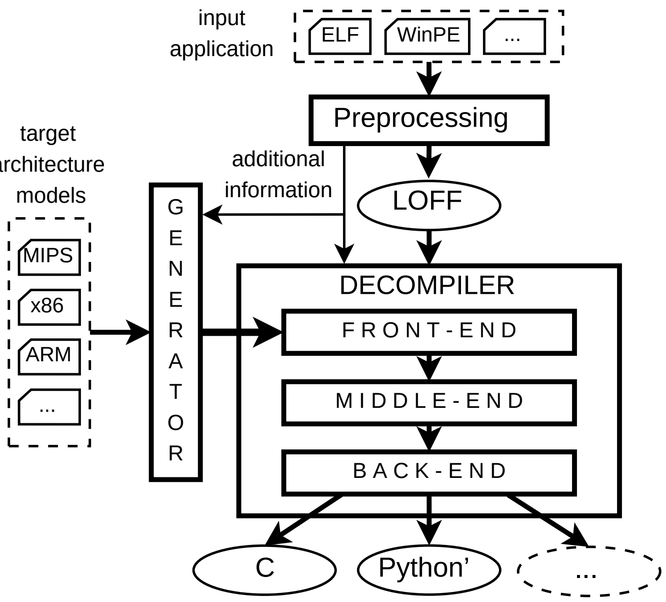


Fig. 3. The concept of the Lissom project's retargetable decompiler.

(1) At first, the input binary executable file is transformed using a plugin-based *binary file converter* from a particular OFF (e.g., Windows PE, ELF) into its own internal object-file-format called **LOFF** (Lissom Object File Format) [16]. LOFF was designed in reference to independence on any particular architecture, universality, and to be well readable. Therefore, it is possible to describe architectures with different types of endianity, byte sizes, instruction lengths, or instruction alignments. It is also possible to store executable, object, or library code within the LOFF format.

The LOFF structure is similar to the COFF format. Basically, it has one header, followed by section headers, sections, and symbolic information (symbols, relocations, and debug information). The section's content is characterized by section flags. The format of LOFF is textual; therefore, it is possible to study its content without any additional tools, see Figure 4. LOFF is used by a complete set of retargetable tools that are automatically generated in this project (e.g. retargetable disassembler, simulator, and decompiler).

```
AgT62kG9y7   // Magic string
32           // Word bit-size
4            // Bytes per word
0            // Byte order, 0-little, 1-big
X            // Flags (eXecutable, etc.)
1            // Is the entry point set?
143654972    // Byte address of the entry point
30           // Section count
1            // Symbol table count
...          // Information about sections
.text        // Section header name
0            // Section byte alignment
1            // Is address absolute?
143654972    // Section address
T            // Section flags (Text, Data, BSS, etc.)
10536        // Section data size in bytes
0            // Count of relocations
20           // First line of section data
0            // First line of relocation data
             // Section data follows
00111111111000000111111000000101 // .text section data
00000000000000000000000000000000
```

Fig. 4. Simplified example of the LOFF format (several attributes are not listed).

In present, the conversion plugin from each supported OFF into LOFF is hand written; thus, the converter is not truly retargetable yet. This is the reason why we need a fully-automatic retargetable solution, such as presented in this paper.

(2) Afterwards, the LOFF file is processed in the *front-end* part which is partially automatically generated based on the description of target architecture (e.g. MIPS, ARM, Intel x86). The architecture description language ISAC [25], developed also within the Lissom project, is used for this purpose. This decompilation phase is responsible for decoding of machine-code instructions, their static analysis, and detection of HLL constructions (e.g. loops, functions). The resulting code is emitted as LLVM IR [24], which is used as an internal code representation of decompiled applications in the remaining decompilation phases.

(3) Afterwards, this program representation is optimized in a *middle-end* using many optimization passes.

(4) Finally, the program intermediate representation is emitted as the target HLL in a *back-end*. Currently, the C language and a Python-like language are used for this purpose and the decompiler supports decompilation of MIPS, ARM, and x86 executables. Both middle-end and back-end are built on the top of the LLVM Compiler System [38].

6. Context-Sensitive Description of Object File Formats

In the previous section, we described the current state of the OFF conversion tool used within the Lissom project. This plugin-based concept has been already implemented and it is used in practice. Its main drawback is the implementation complexity of each newly created plugin. Such plugin has to be written manually either on the top of some existing library or written entirely from scratch.

The complexity of existing parsing-libraries differs dramatically based on the target file format and supported features of library, see Table 1. For example, the BFD library supports multiple file formats (more than 50), but it contains more than half million code lines and its maintenance and extensibility is questionable. On the other hand, there exist lightweight libraries, like ELFIO, containing only few thousand code lines, but they lack any advanced functionality, such as processing of the parsed files.

Table 1. Complexity of several existing OFF parsing libraries.

Parsing library	**Lines of code (LoC)**
Binary File Descriptor library (BFD)	615,856
PeLib	12,220
LibELF	10,930
pyelftools	10,582
ELFIO	3,068

The existing plugins used within the Lissom project have different complexity too, see Table 2. The former three plugins in the table use the third party libraries; therefore, they are relatively small. On the other hand, the E32Image and Android DEX plugins are build from scratch and they are larger than the others.

Table 2. Complexity of Lissom project OFF-conversion plugins.

Conversion plugin	**Lines of code (LoC)**
WinPE	2,831
ELF	2,154
Mach-O	2,227
E32Image	9,419
Android DEX	10,582

According to our experience, the manual implementation of conversion plugins is slow (in matter of implementation) and prone to errors. This approach is also complicated whenever implementing a non-common OFF (e.g. bFLT, XCOFF, OMF) because there are no suitable existing parsing libraries.

In order to achieve a true decompilation retargetability, we should apply the concept similar to one used for description of target processor architectures and automatic generation of the front-end part. This can be done in two steps. (1) Develop a specific language for OFF description. (2) Create an automatic generator of OFF-handling tools (i.e. OFF parsing and conversion) based on the description of the target OFF. Once this concept is adopted, the description complexity of the new OFFs should significantly decrease (i.e. the user will need to describe OFF using several hundred lines without using any external libraries).

In the rest of this section, we introduce the OFF description language. This language should be able to describe structure of each particular OFF as well as its context-sensitive aspects. We specify this language by using a grammar denoting this language and we add its brief description. At the end of this section, an example of ELF description is presented.

6.1. Grammar of the OFF Description Language

In programming language terminology, the grammars (see Section 2) are used for describing syntax of programming languages. In other words, a grammar creates a core of a programming-language parser. Such parser handles input programs (written in this language) using the grammar productions (rules). Parser can be used within compilers, verification software, or just for syntax checking. Within the classical compilation concept (see [1]), grammar serves only for description of syntax. The programming language semantics have to be described manually (e.g. HLL code realizing analysis of parsed code, semantic actions coupled with grammar productions).

In our case, we also use grammar to represent some kind of code — OFF structure. Each particular grammar description specifies one OFF; using this description, we are able to automatically generate the parser of this OFF. This parser will be used as a core of OFF converter to LOFF format. Moreover, our grammar is more advanced and it can also describe context-sensitive properties as well as semantic actions on the level of the grammar itself (this is another difference to existing OFF grammars described in Section 4 that are based on classical context-free grammars as defined in Definition 3). Formal definition and parsing of this grammar are described in the following section.

The language is designed for a description of the common OFFs (and hopefully the future ones, too). Executable or object file on parser's input are viewed as a binary stream. Its parsing is done via interconnected analyzers that invoke each other whenever it is necessary. Analyzers are also able to seek to the desired file offset within the stream. The language is not limited to any particular OFF construction, and it is capable to describe optional or scattered parts of the OFFs.

Modified Extended Backus-Naur Form (EBNF) is used for grammar's syntax description. Terminal symbols are typeset in **`boldface`**. Symbol $\sim$ is used for concatenation. Sequences (i.e. zero or more repetitions) are denoted by {}; optional constructions (i.e. zero or one occurrence) are denoted by `[]`; finally, selections (i.e. a choice between more constructions) are denoted by `|`. The grammar is depicted in Figure 5. For clarity, only the most important productions are specified.

```
start        -> root analyzer_def { parser_def } { production }
analyzer_def -> analyzer id ( [ offset [ , offset] ] ) { {
                statement ; } }
statement    -> element [ { semantic_actions } ]
             -> analyzer_id { [ times ] } [ { semantic_actions } ]
element      -> type id_attribute
             -> type [ value ]
analyzer_id  -> id_attribute ( [ offset [ , offset] ] )
type         -> ( int | uint ) ~ bitwidth_size { [ array_size ] }
attribute    -> [ < id { , id } ] > ]
production   -> ( id_attribute { , id_attribute } ) ->
                ( [ id'_attribute ] { , [ id'_attribute ] } ) [ priority ]
```

Fig. 5. Grammar of the OFF description language.

`start` is the start symbol of the grammar (see Definition 1). The keyword **`root`** denotes the starting analyzer, which is executed at the beginning of parsing. Each analyzer can be controlled by the begin and end `offset`. In that case, analyzer executes its job from the beginning offset and it must finish analysis before the stop offset, otherwise it will end as a parsing error. Analyzers read desired number of bits from an input stream, see Figure 6 for illustration.

The number of bits is specified by `element` with different sizes (specified by `type`). Elements are continual sequences of bits in an input stream. The value of an element can be skipped (i.e. so-called "don't care" value), enforced (i.e. analyzer ends with error if there is an unexpected value on input), or checked by analyzer, see Figure 7.

Elements and analyzers may contain a list of `attributes`. Attributes contain information about properties such as element's value or type. They can be used either in semantic actions (e.g. checking of element) or in context productions. In general, attributes are used for re-referencing previously parsed parts, such as information from OFF header. This context behavior is not common in classical programming language grammars. Therefore, it is possible to use both synthesized and inherited attributes from previously parsed elements within the semantic actions, see [1] for details.

Checking of elements is done either via `semantic_actions`, which are statements of the ANSI C code. Semantic actions can be used either for element checking as well as for interaction with retargetable tools. In our case, they are used mainly for direct LOFF generation. For illustration see Figure 8.

```
// Root (starting) parser of a particular file format XY
root analyzer XY_OFF_parser ()
{
    /* It invokes an analyzer of file-header. The header
       is located on the first 64 bytes. */
    header_parser(0, 512);
    // ...
}

// Parser of header - limited by offset range
analyzer header_parser (start_offset, end_offset)
{
    // ...
}

// Parser not limited by any offset range
analyzer another_parser ()
{
    // ...
}
```

Fig. 6. Example of analyzers definition.

```
analyzer header_parser (start_offset, end_offset)
{
    uint8 'X';  // Two magic bytes - enforced values
    uint8 'Y';
    int16;      // Don't care value (e.g. OFF version)
    // ...
}
```

Fig. 7. Example of statement types that are usable within analyzers.

Parsing can be also controlled via context `productions`; they are formatted as scattered context grammar productions (see Defintion 4); therefore, the number of items within brackets must be the same on both sides (ε-rules are allowed). The nonterminals $\mathtt{id}_{\mathtt{attribute}}$ stand for `element` or `analyzer_id` and they are rewritten according to the right-hand-side of those productions. Attributes are also taken into account during derivation. Finally, it is possible to describe `priority` of each production. A higher value means a higher priority. This is handy whenever we need to perform some actions before any other production (e.g. detecting a fault OFF structure as soon as possible). Details about parsing of these productions are described in the following section.

Finally, analyzers are interconnected via the `analyzer_call` statement. Analyzer can be invoked multiple times using `times`, this is useful for descrip-

```
analyzer header_parser (start_offset, end_offset)
{
    // ...
    int16 architecture <value>
    {
        if (architecture.value != 1)
        {   // Unsupported target architecture type
            parse_error();
        }
        else
        {   // C code producing a part of OFF conversion
            converter->setArchitectureType(architecture.value);
        }
    };
    // ...
}
```

Fig. 8. Usage of attributes and semantic actions within analyzers.

tion of repeating parts (e.g. table items). Analyzer invocation can also be done within the semantic actions by a call to the function with analyzer's name. Therefore, it is possible to conditionally invoke different analyzers based-on an actual context, see Figure 9.

```
root analyzer XY_OFF_parser ()
{
    // Invocation with offset range
    header_parser(0, 512);
    // Invocation without offset range
    another_parser();
    // Invocation 10 times
    another_parser() [10];
    // ...
}
```

Fig. 9. Example of different types of analyzer call.

6.2. Example of Usage

We can illustrate usage of the previously defined language on the 32-bit ELF format. A snippet of this description is depicted on Figure 10. The following description is used for its conversion to the Lissom LOFF format. At first, the header is analyzed by invocation of `elf_header` analyzer. This analyzer starts at zero offset and analyzes all its elements and makes necessary checks.

It also converts basic information (e.g. entry point, endianness) to the LOFF format. The **`value`** attribute is used in several elements for referencing from other elements. At the end of the `elf_header` analyzer, we can see conditional invocation of section-header-table analyzer. It will be executed only if the table is present. We can also see that the analyzer `elf_sht` is invoked together with specification of its beginning and ending offset gathered from previous attributes. This corresponds to the structure depicted in Figure 1.

The last construction depicted on this example is a context production. It controls that executable files do not contain static relocations (e.g. static relocation `R_386_PC32`). It is marked with priority higher than other productions; therefore, it will be checked at first. Whenever the preliminary part is satisfied (e.g. executable file is not properly linked), it blocks parsing by nonterminal `error`, which leads to parsing error.

7. Context Parsing

In this section, we present a concept of the context parser that can be used for parsing the previously described OFF language. The major difference to other existing languages is its support of describing context-sensitive relations. However, parsing of these constructions is non-trivial because there is no suitable formalism capable of describing such grammar in present.

The idea of context parser is not entirely new and we can find several attempts to create a parser for context-sensitive language (Definition 2) in past, see [37,35,1]. These attempts were only partially successful. They were either focused on a very specific aspects of some domain-specific language, or they were not based on formal models; therefore, it was hard to prove such concepts.

Today's traditional techniques perform context analysis via semantic actions written in the host language accompanying usually context-free grammar of a suitable form (see [1]). The other possibility is to use some context-free parser based on any available technique and then to perform analysis of a data structure created as an output of the parser (usually some tree-like structure or some kind of byte-code [30]).

A mixture of several descriptive means (grammar together with host language or another combination) bound by explicit data structures stored in trees, attributes, code, or their mixture is not suitable if an analyzer is to be described formally. Moreover, a change to the input language syntax usually dramatically affects other parts of a parser.

Therefore, in this section, we define two new formal models that are based on scattered context grammars—*attributed scattered context grammars* and *attributed scattered context grammars with priority function*. These grammars can be effectively used for formal description of context-sensitive relations in a particular language. Furthermore, we modify the existing regulated pushdown au-

```
root analyzer ELF32 () {
    elf_header(0) { check_header(); }; // parse ELF header
}
analyzer elf_header (start_offset) {
    uint8 [16] e_ident { /* Check of the "ELF Identification"
                            field */ };
    uint16 e_type <value> {
        if (e_type.value > 4)
            parse_error(); // Unsupported ELF file type
    };
    uint16;              // e_machine - a don't care value
    uint32 1;            // e_version needs to be '1'
    uint32 e_entry <value> { // Direct generation of LOFF
        LOFF->setEntryPoint(e_entry.value);
    };
    // ...
                         // Section header table's offset (SHT)
    uint32 e_shoff <value>;
    // ...
    // Size of entrie in SHT and number of elements in SHT
    uint16 e_shentsize;
    uint32 e_shnum <value> {
        if (e_shoff.value != 0) // Analyze SHT
            elf_sht(e_shoff,
                    e_shoff + e_shnum.value * e_shentsize);
    };
    // ...
}
analyzer elf_sht (start_offset, end_offset) {
    // ...
                         // Analysis of Section Header Table
}
analyzer elf_section (start_offset, end_offset) {
    // ...
                         // Analysis of each particular section
}

// Productions describing context behavior
// Simplified control of appearance of static relocations
// within executable files
(elf_header<is_executable>, elf_relocation<is_static>}) ->
    (error, error) [999] // High-priority production
// Other productions
```

Fig. 10. A code snippet of an ELF description using OFF language.

tomata (see [17]) for parsing these grammars. Finally, we give a brief overview of a context parser construction.

7.1. Attributed Scattered Context Grammars

In this subsection, we define two new formalisms that are based on scattered context grammars. We assume a reader is familiar with the attributed grammars (for further details see [36,30,1].

Definition 10. A *voidy n-tuple* over domain D is the tuple

$$< d_1, \ldots, d_n > \in D^n,$$

where D^n stands for $D_1 \times D_2 \times \ldots \times D_n$ and $n \in \mathbb{N}$; if $n = 0$ then the tuple is void and we write $<>$ or simply we do not write anything if it is clear from the context.

Definition 11. *Variable voidy Cartesian product* $\cup D$ over domain D is defined as

$$\cup D = \cup_{i=0}^{n} D^i,$$

where D^n stands for $D_1 \times D_2 \times \ldots \times D_n$ and $n \in \mathbb{N}$; $D^0 = \{<>\}$.

Definition 12. An *attributed scattered context grammar* (aSCG) is a seven-tuple,

$$G = (V, T, P, S, D, R, \rho),$$

where

- V is a total alphabet;
- $T \subset V$ is a finite set of terminals;
- $S \in V - T$ is the start symbol;
- P is a finite set of productions of the form

  $$(A^1_{w_1}, \ldots, A^n_{w_n}) \to (x^1_{@}, \ldots, x^n_{@}),$$

  where $A^i \in V - T$, $w_i = \rho(A^i)$, $x^i_{@} \in V^*$ for all $i : 1 \leq i \leq n$ and all symbols in $x^i_{@}$ have their corresponding voidy tuple of attributes;
- D is the domain of attributes;
- R is the naming of attributes representing any value from D;
- ρ is a mapping $\rho : V \to \cup R$, where $\cup R$ is the variable voidy Cartesian product.

Definition 13. An *attributed propagating scattered context grammar* (aPSCG) is an aSCG

$$G = (V, T, P, S, D, R, \rho),$$

in which every $(A^1_{w_1}, \ldots, A^n_{w_n}) \to (x^1_{@}, \ldots, x^n_{@}) \in P$ satisfies $x^i_{@} \in V^+$ for all $i : 1 \leq i \leq n$.

Notation of attribute use is the following: we write

$$A_{<a_1,\ldots,a_n>} \text{ if } \rho(A) \ = \ < a_1, \ldots, a_n >$$

for any n, or simply A_w if attribute names are not in our focus; we write $A_{<>}$ if we want to stress that void attribute tuple is assigned to the symbol, or we write just A for the sake of simplicity. If there is a string of symbols $x = A_1 \ldots A_n$ and for every $A_i, i \in \{1 \ldots n\}$ there is w_i such that $\rho(A_i) = w_i$ we write $x_@$ to stress that every symbol of x has its voidy tuple of attributes.

Definition 14. Let $G = (V, T, P, S, D, R, \rho)$ be a (propagating) aSCG. If

$$\begin{aligned} y &= u_1 A^1_{w_1} u_2 \ldots u_n A^n_{w_n} u_{n+1}, \\ z &= u_1 x^1_@ u_2 \ldots u_n x^n_@ u_{n+1}, \end{aligned}$$

and $y, z \in V^*$, $p = (A^1_{w_1}, \ldots, A^n_{w_n}) \to (x^1_@, \ldots, x^n_@) \in P$, and if every attribute occurring in $A^1_{w_1}, \ldots, A^n_{w_n}$ and $x^1_@, \ldots, x^n_@$ has a value from D defined and every occurrence of some attribute $a \in R$ in $A^1_{w_1}, \ldots, A^n_{w_n}$ and $x^1_@, \ldots, x^n_@$ carries the same value from D then y directly derives z in the (propagating) aSCG G according to the production p,

$$y \Rightarrow_G z\ [p] \text{ (or simply } y \Rightarrow_G z).$$

A language generated by (propagating) aSCG is defined the same way as for (propagating) SCG. Similarly, family of (propagating) attributed scattered context languages is defined as $\mathcal{L}$(a(P)SC).

To give a light insight and motivation on usage of attributed grammar, we present a small example. Let us take into account the language $a^n b^n c^n$ for $n \geq 1$. This is truly a context-sensitive language (see [28]). Using SCG[1], we can describe the language by grammar:

$$G_1 = (\{S, X, C, a, b, c\}, \{a, b, c\}, P, S),$$

with P containing

$$\begin{aligned} P = \{ \quad & (S) \to (XC), & [p_1] \\ & (X, C) \to (aXb, cC), & [p_2] \\ & (X, C) \to (ab, c)\} & [p_3] \end{aligned}$$

As an example of derivation by using this grammar

$$\begin{aligned} S &\Rightarrow XC & [p_1] \\ &\Rightarrow aXbcC & [p_2] \\ &\Rightarrow aaXbbccC & [p_2] \\ &\Rightarrow aaabbbccc & [p_3] \end{aligned}$$

From a formal point of view, the presented grammar represents a perfect description of a given language. From a practical point of view, this kind of description is too specific. Let us assume a modification of this language: $z^n u^n v^n$ for $n \geq 1$. This language has the same structure as the previous one except the

[1] This grammar is actually a propagating SCG because it does not contain any erasing rules.

different terminal names. However, this small difference means that the original grammar has to be significantly rewritten.

In particular, terminals a, b, and c are too specific in the original grammar. In a fact, each of them can be considered as an identifier, which was named e.g. 'a'. On the other hand, such an identifier is bound to some particular value ('a') that can be described by a value of attribute bound to particular (otherwise anonymous) terminal.

Thus, we introduce attributes to keep fully formal view and obtaining expressive power and variability. Therefore, we define the attributed SCG

$$G_2 = (V, T, P, S, D, R, \rho).$$

Now we can modify our example in such a way: we add a domain of attributes D of all textual strings (string is written in quotes, e.g. 'z'), we add a naming $R = \{q, w, e\}$, we define mapping ρ such a way, so that:

$$\begin{aligned} \rho(S) &= <> \\ \rho(X) &= < q, w > \\ \rho(C) &= < e > \\ \rho(a) &= < q > \\ \rho(b) &= < w > \\ \rho(c) &= < e > \end{aligned}$$

and present an aSCG grammar productions (as a modification of the previous SCG):

$$\begin{aligned} (S) &\rightarrow (X_{<'a','b'>} C_{<'c'>}) \\ (X_{<q,w>}, C_{<e>}) &\rightarrow (a_{<q>} X_{<q,w>} b_{<w>}, c_{<e>} C_{<e>}) \\ (X_{<q,w>}, C_{<e>}) &\rightarrow (a_{<q>} b_{<w>}, c_{<e>}) \end{aligned}$$

A modification of the presented aSCG allows to change terminals with redefinition of just a single grammar production, in particular, attribute values, as the production remains the same, as such. To get the second mentioned language, we have to change just the first production to $(S) \rightarrow (X_{<'z','u'>} C_{<'v'>})$ and the rest remains the same.

To extend expressive power and bring necessary pragmatic features for practical exploitation of a(P)SCG in context analysis/parsing, we extend a(P)SCG to priority attributed scattered context grammars.

Definition 15. A *(propagating) attributed scattered context grammar with priority*
(a(P)SCGP) is an eight-tuple

$$G = (V, T, P, S, D, R, \rho, \pi),$$

where (V, T, P, S, D, R, ρ) is a (propagating) attributed scattered context grammar and π is a *function*

$$\pi : P \rightarrow \mathbb{N}.$$

Definition 16. Let $G = (V, T, P, S, D, R, \rho, \pi)$ be an a(P)SCGP. We say that y directly derives z in a(P)SCG G according to the production p, $y \Rightarrow_G z\ [p]$ (or simply $y \Rightarrow_G z$), if and only if:

- $y = u_1 A^1_{w_1} u_2 \ldots u_n\ A^n_{w_n} u_{n+1} \in V^*$,
- $z = u_1 x^1_{@} u_2 \ldots u_n x^n_{@} u_{n+1} \in V^*$,
- $p = (A^1_{w_1}, \ldots, A^n_{w_n}) \to (x^1_{@}, \ldots, x^n_{@}) \in P$,
- there is no $p' = ({A'}^1_{w_1}, \ldots, {A'}^n_{w_n}) \to ({x'}^1_{@}, \ldots, {x'}^n_{@}) \in P$, such that:
  1. $y = u'_1 {A'}^1_{w_1} u'_2 \ldots u'_n {A'}^n_{w_n} u'_{n+1} \in V^*$, and
  2. $\pi(p') > \pi(p)$;
- and conditions of Definition 14 for attributes must hold.

Language generated by a(P)SCGP is defined similarly as for a(P)SCG.

To give an order of rules when several options could be used, we use priority. For demonstration, we define the attributed SCG with priority

$$G_3 = (V, T, P, S, D, R, \rho, \pi).$$

The grammar is the same as G_2 up to priority mapping, which is defined as:

$$\begin{aligned}
\pi((S) \to (X_{<'a','b'>} C_{<'c'>})) &= 1 \\
\pi((X_{<q,w>}, C_{<e>}) \to (a_{<q>} X_{<q,w>} b_{<w>}, c_{<e>} C_{<e>})) &= 1 \\
\pi((X_{<q,w>}, C_{<e>}) \to (a_{<q>} b_{<w>}, c_{<e>})) &= 1
\end{aligned}$$

Then, the sentence $a_{<'a'>} a_{<'a'>} b_{<'b'>} b_{<'b'>} c_{<'c'>} c_{<'c'>}$ is obtained by the following derivation:

$$\begin{aligned}
S &\Rightarrow X_{<'a','b'>} C_{<'c'>} & [p_1] \\
&\Rightarrow a_{<'a'>} X_{<'a','b'>} b_{<'b'>} c_{<'c'>} C_{<'c'>} & [p_2] \\
&\Rightarrow a_{<'a'>} a_{<'a'>} b_{<'b'>} b_{<'b'>} c_{<'c'>} c_{<'c'>} & [p_3]
\end{aligned}$$

that represents the string $aabbcc$.

7.2. Regulated Pushdown Automata

As has been illustrated above, the a(P)SCGP can be easily used for description of context-sensitive languages. However, we still need a formal model for parsing such description. For this reason, we use a *Regulated Pushdown Automata*.

In [15], it is presented, how regulated pushdown automata can be exploited for building context parsers derived from scattered context grammars of particular features. Basic concept of regulated pushdown automata can be found in [17,27] — papers especially present definition and expressive power of various versions of regulated pushdown automata.

Consider a pushdown automaton (PDA)

$$M = (Q, \Sigma, \Omega, R, s, S, F),$$

where

– Q is a *finite set of states*;
– Σ is an *input alphabet*;
– Ω is a *pushdown alphabet*;
– R is a *set of productions* of the form

$$Apa \rightarrow wqb,$$

where $A \in \Omega$, $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $w \in \Omega^*$ and $b \in \{a, \varepsilon\}$ (if $b \neq \varepsilon$ then the production "tests" the value under the reading head, the head is not shifted, the symbol is not read);
– $s \in Q$ is the *start state*;
– $S \in \Omega$ is the *start symbol*;
– $F \subseteq Q$ is a *set of final states*.
– Without a loss of generality, we require that Q, Σ, and Ω are pairwise disjoint.

Now, consider a control language, Ξ (formally defined below), over M's productions. Informally, with Ξ, M accepts a word, x, if and only if Ξ contains a control word according to which M makes a sequence of moves so it reaches a final configuration after reading x.

Let Ψ be an alphabet of *production labels* such that $card(\Psi) = card(R)$, and ψ be a bijection from R to Ψ. For simplicity, to express that ψ maps a production, $Apa \rightarrow wq \in R$, to ρ, where $\rho \in \Psi$, this paper writes $\rho.Apa \rightarrow wq \in R$; in other words, $\rho.Apa \rightarrow wq$ means $\psi(Apa \rightarrow wq) = \rho$.

Definition 17. A *configuration* of M, χ, is any word from $\Omega^* Q \Sigma^*$. For every $x \in \Omega^*$, $y \in \Sigma^*$, and $\rho.Apa \rightarrow wq \in R$, M makes a move from configuration $xApay$ to configuration $xwqy$ according to ρ, written as $xApay \vdash xwqy\ [\rho]$.

Let χ be any configuration of M. M makes *zero moves from* χ *to* χ *according to* ε, symbolically written as $\chi \vdash^0 \chi\ [\varepsilon]$. Let there exist a sequence of configurations $\chi_0, \chi_1, \dots, \chi_n$ for some $n \geq 1$ such that $\chi_{i-1} \vdash \chi_i\ [\rho_i]$, where $\rho_i \in \Psi$, for $i = 1, \dots, n$, then M makes n *moves from* χ_0 *to* χ_n *according to* $\rho_1 \dots \rho_n$, symbolically written as $\chi_0 \vdash^n \chi_n\ [\rho_1 \dots \rho_n]$.

Definition 18. Let Ξ be a *control language* over Ψ; that is, $\Xi \subseteq \Psi^*$. With Ξ, M defines the following three types of accepted languages:

$L(M, \Xi, 1)$—*the language accepted by final state*
$L(M, \Xi, 2)$—*the language accepted by empty pushdown*
$L(M, \Xi, 3)$—*the language accepted by final state and empty pushdown*

defined as follows. Let $\chi \in \Omega^* Q \Sigma^*$. If $\chi \in \Omega^* F$, $\chi \in Q$, $\chi \in F$, then χ is a *1-final configuration*, *2-final configuration*, *3-final configuration*, respectively. For $i = 1, 2, 3$, define $L(M, \Xi, i)$ as $L(M, \Xi, i) = \{w \mid w \in \Sigma^*,$ and $Ssw \Rightarrow^* \chi\ [\sigma]$ in M for an i*-final configuration*, $\chi,$ and $\sigma \in \Xi\}$.

Definition 19. *Regulated pushdown automata* (RPDA). For any family of languages, X, set $RPDA(X, i) = \{L \mid L = L(M, \Xi, i)$, where M is a PDA and $\Xi \in X$, where $i = 1, 2, 3\}$.

Namely, pushdown automata regulated by linear languages have the same power as Turing machine

$$RE = RPDA(LIN, 1) = RPDA(LIN, 2) = RPDA(LIN, 3),$$

where RE stands for the set of all recursively enumerable languages and LIN stands for the set of all linear languages [26] — proof can be found in [17].

Thus, such automata are powerful enough for analysis of context languages. Nevertheless, we need a deterministic version of such automata. Their detailed description and a way, how the automaton can be built from a SCG of certain features, can be found in [15].

Definition 20. Let $M = (Q, \Sigma, \Omega, R, s, S, F)$ be a regulated pushdown automaton, with set of labels Ψ, bijection ψ from labels Ψ to productions R, and with control language Ξ. Such an RPDA is *deterministic* (DRPDA) if being in a state q, $q \in Q$, the appropriate action, which should be performed, can always be deterministically selected. This can only be due to the following two circumstances:

(1) For the given state, there is only one production $r \in R$ that is applicable in a given situation (state, symbol on the top of the pushdown or under the reading head) and, moreover, control language admits such a production.

(2) If there are more than one productions that are applicable in a given situation then the production can be deterministically denoted according to the actual context of sentential form of the control language applicable to the current state of operation performed by RPDA.

To give a rough idea from another viewpoint: in a center, there is non-deterministic pushdown automaton; all of its operations are encoded as a symbols of the control-language alphabet; successful operation of the PDA must be verified by the control language, which means that operation of PDA produces a string of symbols (step-by-step operations of the automaton are encoded to string of symbols) and if the string is a sentence of the control language then the operation of regulated PDA is successful; if PDA fails during its operation or the produced string is not in the control language then it means that analyzed input is not accepted.

7.3. Context Parser Construction

Relation between automata presented above and implementation is quite simple. We can build appropriate automaton from a given grammar (see [14,15,18]) automatically. Moreover, usage of Haskell programming language enables to build a kind of domain specific language on the top of Haskell. Thus, it is necessary to define the wanted grammar inside Haskell using supporting predefined constructs and the parser is done. Lexical analysis is done in the same way as in any other parser (i.e. definition of lexemes and their transformation to tokens, see [1]). Also manipulation of the output of the parser is done in a traditional way. The key feature is that just a simple modification of the grammar allows to dramatically modify the parsed input. Thus, any change is much faster then using any other technique.

Furthermore, we can apply the same principles as in SCG parsing (see [15]):

– regulated PDA can be made deterministic;
– having SCG of suitable features (LL SCG, see [15]), we can algorithmically derive a deterministic regulated pushdown automaton, which accepts (decides) the language generated by the SCG.

To achieve full flexibility and big expressive power, so that changes in a language can be efficiently handled on the grammar level, we need to introduce attributes and priorities to LL SCG parsers.

Attributes Introduction of attributes is not difficult at all — grammar (omitting attributes) must satisfy the same conditions as LL SCG grammars, plus the following one — $\forall (A^1_{w_1}, \ldots, A^n_{w_n}) \to (x^1_{@}, \ldots, x^n_{@}) \in P$ it must hold:

– $\rho(A^1) = <>$ and
– let $x^1_{@} = X^1_{v_1} \ldots X^m_{v_m}$ then $\forall X_i \in (V - T) : \rho(X_i) = <>, i \in \{1 \ldots m\}$

Priorities Fortunately, priorities are not a problem of construction parsing tables and automaton as such. They are problem of saving automaton configuration and its restoration — from a formal point of view.

The situation is such, priorities can cause that we have several grammar productions for expansion for the same automaton configuration (symbol under the reading head and top of the pushdown) — we say the productions are overlapping. In the traditional notion of deterministic PDA it means a conflict and no automaton can be built.

If we have priorities for grammar productions introduced then this situation is conflict if two or more such overlapping productions have the same priority assigned.

If the priorities for overlapping productions are different then we have to order such set of productions from the highest priority to the lowest one. When the automaton configuration gets to the point when some of these productions could be applied, the production with the highest priority is applied at first. If it fails then the original configuration is restored and the next production is applied and so on and so on, until some production succeeds. If none of the productions succeeds then the analysis fails with an error.

The problem is about storing the configuration and restoration, especially, how we can recognize that some production expansion fails. Fortunately, as it can be seen in [14,15,18], during expansion, when we search for suitable nonterminals on the pushdown, we use the control language to save the content of the pushdown that is popped out of the pushdown. Thus, when we reach bottom of the pushdown it means that the production cannot be expanded in the situation, so that we should apply another one. In such a situation the content of the pushdown is saved in the context of the control language and we can restore it to its original content before trying to expand the production. It is used the same technique with a small difference, when right hand sides of the so far expanded part are not pushed to the pushdown, but the original content is.

7.4. Experimental Results

In present, the context parser of the OFF language is in the prototype phase. Therefore, we are unable to give any experimental results in a deeper detail yet. On the other hand, the concept of the parser can be described using the simple context-sensitive language.

Performance measurement of our approach is not easy. The reason is that there is no context parser based on grammar input available. General approaches are well known to be inefficient. Thus, it was quite difficult to find simple use case for comparison.

Our implementation language is Haskell due to ease of use for our purposes. Re-implementing our parser in C/C++ would be time consuming, so we decided to implement competitive parser of some suitable language in Haskell directly, without using our grammar based context parser.

We have decided to use parser of the aforementioned language $a^n b^n c^n$ based on the presented grammar, but without any attributes and priority — firstly, they are not necessary for such a simple case; secondly, it would be quite complicated to implement something similar in the other program for comparison.

The comparison is unfair for the SCG-based solution, though. We compare parser based on complex SCG with straightforward ”C-like” implementation of analysis of the language $a^n b^n c^n$. There are several reasons, why it is unfair:

- The grammar based parser uses stack to create contextual information and its consumption is proportional to input size.
- The ”C-like” implementation is very much Haskell syntax of C approach, on the other hand the grammar based parser is very much of the Haskell.
- ”C-like” implementation is constant space so it provokes for better performance.

In other words, we compared something incomparable, in a fact. The comparison of speed is depicted in Figure 11. The tests were limited on size due to stack utilization and application size limitation in Windows 32-bit application.

Surprisingly, the time complexity according to input size is almost the same. Thus, we can state that our approach is not only very efficient in change incorporation both on user and implementation side, but is is even quite efficient from the evaluation speed viewpoint.

The evaluation of this concept on a more complex examples (such as the OFF language) is marked as our future research but unavailable yet.

8. Conclusion and Future Work

This paper was focused on handling of OFFs and its usage in retargetable tools. Several existing solutions were presented, and their limitations were discussed.

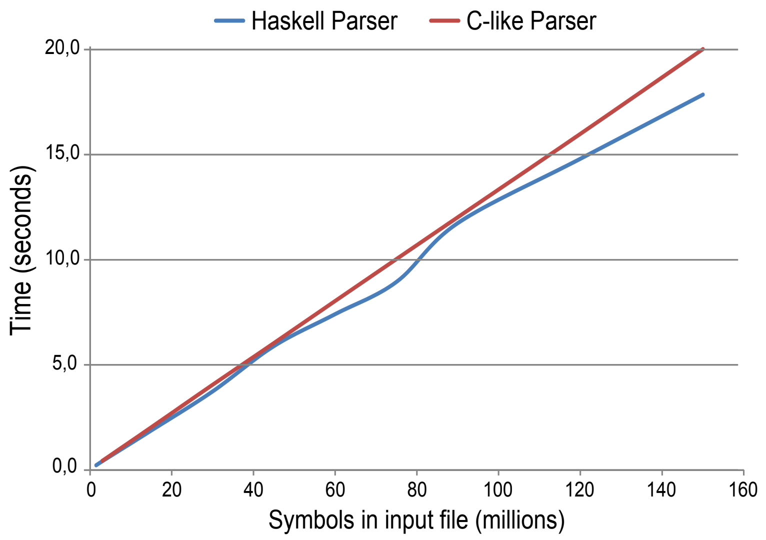


Fig. 11. Speed comparison between two parsers of the $a^nb^nc^n$ language.

The main contribution of this paper is a presentation of the language for OFF description and concept of its parser. The major advantage of this concept is its ability to describe and parse context-sensitive properties. The parser is based on the formal models that were designed for this purpose.

A prototype of this context parser is under development. The Haskell programming language is used for this purpose because it is well-suited for our needs (lazy evaluation [13], type inference, etc.). According to the preliminary experimental results, which were focused on simple languages like $a^nb^nc^n$, this approach is faster than other parsers of the same language.

The language can be used for OFF parsing and manipulation. Its main usage is within an existing retargetable decompiler, where it will be used for conversion from platform-dependent OFFs into an internal COFF-based file format. However, this is not a limitation because the language can be used in other retargetable tools, such as disassemblers, loaders, or debuggers.

In the future research, we would like to use the context parser in other areas. For example it can be used for natural language processing, description of other binary file formats (i.e. not just OFF), or parsing of HLL programming languages, such C, where it will be able to automatically check consistency of declaration, definition, and usage of variables, see [28,41,37] for details.

Acknowledgments. This work was supported by the project TA ČR TA01010667 System for Support of Platform Independent Malware Analysis in Executable Files, BUT grant FEKT/FIT-J-13-2000 Validation of Executable Code for Industrial Automation Devices using Decompilation, BUT FIT grant FIT-S-11-2, by the project CEZ MSM0021630528

Security-Oriented Research in Information Technology, and by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison-Wesley, Boston, 2nd edn. (2006)
2. Apple Inc.: Macintosh application environment. `http://www.mae.apple.com` (1994)
3. Apple Inc.: Mac OS X ABI Mach-O file format reference (2009)
4. Chamberlain, S.: The Binary File Descriptor Library. Iuniverse Inc. (2000)
5. Cifuentes, C.: Reverse Compilation Techniques. Ph.D. thesis, School of Computing Science, Queensland University of Technology, Brisbane, QLD, AU (1994)
6. Digital: Freeport express. `http://www.novalink.com/freeport-express` (1996)
7. Emmerik, M.J.V.: Static Single Assignment for Decompilation. Ph.D. thesis, University of Queensland, Brisbane, QLD, AU (2007)
8. Faase, F.: BFF: A grammar for binary file formats. `http://www.iwriteiam.nl/Ha_BFF.html` (2012)
9. Fernández, M.F.: Simple and effective link-time optimization of Modula-3 programs. SIGPLAN Not. 30(6), 103–115 (1995)
10. Gircys, G.R.: Understanding and Using COFF. O'Reilly & Associates, Inc., Sebastopol, US-CA (1988)
11. Greibach, S., Hopcroft, J.: Scattered context grammars. Journal of Computer and System Sciences 3(3), 233–247 (1969)
12. Hohensee, P., Myszewski, M., Reese, D.: Wabi cpu emulation. Hot Chips VIII (1996)
13. Jirák, O., Kolář, D.: Derivation in scattered context grammar via lazy function evaluation. In: 5th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'09). OpenAccess Series in Informatics (OASIcs), vol. 13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, DE (2009)
14. Kolář, D.: Pushdown Automata: Another Extensions and Transformations. Habilitation thesis, Brno University of Technology, Faculty of Information Technology (2005)
15. Kolář, D.: Scattered context grammars parsers. In: 14th International Congress of Cybernetics and Systems of WOCS. pp. 491–500. Wroclaw University of Technology, PL, Wroclaw, PL (2008)
16. Kolář, D., Husár, A.: Output object file format for assembler and linker. Internal document, Brno University of Technology, Faculty of Information Technology, Brno, CZ (2012)
17. Kolář, D., Meduna, A.: Regulated pushdown automata. Acta Cybernetica 2000(4), 653–664 (2000)
18. Kolář, D., Meduna, A.: Regulated automata: From theory towards applications. In: 8th International Conference on Information Systems Implementation and Modelling (ISIM'05). pp. 34–48. MARQ, Ostrava, CZ (2005)
19. Křoustek, J., Matula, P., Ďurfina, L.: Generic plugin-based convertor of executable file formats and its usage in retargetable decompilation. In: 6th International Scientific and Technical Conference (CSIT'11). pp. 127–130. Ministry of Education, Science, Youth and Sports of Ukraine, Lviv Polytechnic National University, Institute of Computer Science and Information Technologies (2011)

20. Křoustek, J., Přikryl, Z., Kolář, D., Hruška, T.: Retargetable multi-level debugging in HW/SW codesign. In: 23rd International Conference on Microelectronics (ICM'11). p. 6. Institute of Electrical and Electronics Engineers (2011)
21. Křoustek, J., Židek, S., Kolář, D., Meduna, A.: Scattered context grammars with priority. International Journal of Advanced Research in Computer Science (IJARCS) 2(4), 1–6 (2011)
22. Levine, J.R.: Linkers and Loaders. Operating Systems, Morgan Kaufmann Publishers (2000)
23. Lissom: `http://www.fit.vutbr.cz/research/groups/lissom/` (2013)
24. LLVM Assembly Language Reference Manual: `http://llvm.org/docs/LangRef.html` (2013)
25. Masařík, K.: System for Hardware-Software Co-Design. VUTIUM, Brno University of Technology, Faculty of Information Technology, Brno, CZ, 1st edn. (2008)
26. Meduna, A.: Automata and Languages: Theory and Applications. Springer-Verlag, London, GB (2005)
27. Meduna, A., Kolář, D.: One-turn regulated pushdown automata and their reduction. Fundamenta Informaticae 2001, 1001–1007 (2001)
28. Meduna, A., Techet, J.: Scattered Context Grammars and their Applications. WIT Press, Southampton, GB (2010)
29. Microsoft Corporation: Microsoft portable executable and common object file format specification. `http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx` (2013), version 8.3
30. Muchnick, S.S.: Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers, San Francisco, US-CA (1997)
31. Nokia: E32Image. `http://www.developer.nokia.com/Community/Wiki/E32Image` (2012)
32. Ramsey, N., Fernández, M.: Specifying representations of machine instructions. ACM Transactions on Programming Languages and Systems 19(3), 492–524 (1997)
33. Ramsey, N., Fernandez, M.F.: The New Jersey Machine-Code Toolkit. In: USENIX Technical Conference. pp. 289–302 (1995)
34. Ramsey, N., Hanson, D.R.: A retargetable debugger. Tech. rep., Princeton University, Princeton, US-NJ (1992)
35. Roberts, D.M.: Earley parsing for context-sensitive grammars. `http://danielmattosroberts.com/earley/context-sensitive-earley.pdf` (2009)
36. Rodriguez-Cerezo, D., Cabezuelo, A.S., Sierra, J.L.: A systematic approach to the implementation of attribute grammars with conventional compiler construction tools. Computer Science and Information Systems (ComSIS) 9(3), 983–1017 (2012)
37. Rychnovský, L.: Parsing of context-sensitive languages. In: 2nd International Workshop on Formal Models (WFM'07). pp. 219–226. Opava, CZ (2007)
38. The LLVM Compiler Infrastructure: `http://llvm.org/` (2013)
39. TIS Committee: Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification (1995), `http://refspecs.freestandards.org/elf/elf.pdf`
40. Troshina, K., Chernov, A., Derevenets, Y.: C decompilation: Is it possible? In: International Workshop on Program Understanding (IWPU'09). pp. 18–27 (2009)
41. Ung, D., Cifuentes, C.: SRL - a simple retargetable loader. In: The Australia Software Engineering Conference. pp. 60–69. IEEE Computer Society (1997)
42. UQBT - A Resourceable and Retargetable Binary Translator: `http://itee.uq.edu.au/~cristina/uqbt.html` (2012)

43. Ďurfina, L., Křoustek, J., Zemek, P., Kábele, B.: Detection and recovery of functions and their arguments in a retargetable decompiler. In: 19th Working Conference on Reverse Engineering (WCRE'12). pp. 51–60. IEEE Computer Society, Kingston, ON, CA (2012)
44. Ďurfina, L., Křoustek, J., Zemek, P., Kolář, D., Hruška, T., Masařík, K., Meduna, A.: Design of a retargetable decompiler for a static platform-independent malware analysis. In: 5th International Conference on Information Security and Assurance (ISA'11). Communications in Computer and Information Science, vol. 200, pp. 72–86. Springer-Verlag, Berlin, Heidelberg, DE (2011)
45. Ďurfina, L., Křoustek, J., Zemek, P., Kolář, D., Hruška, T., Masařík, K., Meduna, A.: Design of a retargetable decompiler for a static platform-independent malware analysis. International Journal of Security and Its Applications (IJSIA) 5(4), 91–106 (2011)

Jakub Křoustek is a Ph.D. student at the Faculty of Information Technology, Brno University of Technology, Czech Republic. He received the MSc. degree from the same university in 2009. He is currently working on the Lissom research project as the leader of the generic decompiler and debugger development team. His current research interests include the reverse engineering, malware detection, and compiler design, with special focus on the code analysis and reverse translation.

Dušan Kolář went to Brno University of Technology, Czech Republic, where he studied computer science and cybernetics and obtained his degrees in 1994 and 1998. Since then, he has been working at the university, presently at the Faculty of Information Technology. His main research interests are formal languages and automata and formal models with focus on their exploitation in compilers and formal models transformation.

Received: January 20, 2013; Accepted: September 2, 2013.

DOI: 10.2298/CSIS121202065H

An evaluation of keyword, string similarity and very shallow syntactic matching for a university admissions processing infobot

Peter Hancox[1] and Nikolaos Polatidis[2]

[1] School of Computer Science, University of Birmingham
Edgbaston, Birmingham, B15 2TT, United Kingdom
pjh@cs.bham.ac.uk

[2] Department of Applied Informatics, University of Macedonia
156 Egnatia Street, 54006, Thessaloniki, Greece
npolatidis@uom.edu.gr

Abstract. "Infobots" are small-scale natural language question answering systems drawing inspiration from ELIZA-type systems. Their key distinguishing feature is the extraction of meaning from users' queries without the use of syntactic or semantic representations. Three approaches to identifying the users' intended meanings were investigated: keyword-based systems, Jaro-based string similarity algorithms and matching based on very shallow syntactic analysis. These were measured against a corpus of queries contributed by users of a WWW-hosted infobot for responding to questions about applications to MSc courses. The most effective system was Jaro with stemmed input (78.57%). It also was able to process ungrammatical input and offer scalability.

Keywords: chatbot, infobot, question-answering, Jaro string similarity, Jaro-Winkler string similarity, shallow syntactic processing.

1. Introduction

University student recruitment administration is an application where there is potential for a large volume of enquiries of a fairly routine and predictable nature from a world-wide pool of applicants. The costs of call centres (both in terms of running the centres and recruiting and retaining a knowledgeable workforce) make such ventures unattractive. On the other hand, it should be possible to implement a technological solution beyond adding over-large FAQs to web pages. The amount and breadth of information required to answer the applicants' questions would require a large number of long FAQs with quite possibly a complex net of interrelations.

Student recruitment, particularly at graduate level, is international in outlook: in UK postgraduate computing degrees, it is not unusual for international students to outnumber UK students by two to one. Communicating with international applicants brings with it all the problems of understanding versions of

English that can vary from the excellent to the less than adequate. Most applicants have a strong commitment to accessing the information that they need to be able to make a choice of what and where to study. Varying language capabilities notwithstanding, many applicants are very articulate (and so able to frame precise requests for information) and discriminating in making their choices based on that information. Although universities work hard at providing prospectus information that ranges from detail of the structure of courses to the accommodation packages and sports facilities available, many applicants still find themselves wanting to ask highly targeted questions about, for instance, module choice deadlines and visa requirements. Given the wide range of time zones that applicants come from and that 24 hour access to staff is not a possibility, the work reported here emanated from projects to implement a web-based infobot application for answering applicants' enquiries.

The aim was to provide a service where users could ask questions in English about making applications. (The system was not intended to be able to answer highly specific questions such as the reason for rejection of a particular application.) In doing this, the system would have to be able to process input in the syntactic form of questions over the relatively narrow semantic domain of applications for graduate study asked in versions of English that are not always those of a native speaker.

A number of organizations (for instance in the UK, the Alliance & Leicester bank and the O_2 mobile phone company) have made chatbots available on their websites. Chatbots have a long history and the first implementation, ELIZA [30], is particularly famous. Recent bots owe much to the ALICE software which consists of a matching engine and an XML-based language for writing rules to transform input [1].

We argue that the use of traditional chatbots for informational systems is inappropriate. Writing of the first extensive script he prepared for ELIZA which parodied "the responses of a nondirective psychotherapist", Weizenbaum [30, p. 188] stated "I chose this script because it enabled me to temporarily sidestep the problem of giving the program a database of real-world knowledge." ELIZA was a *conversational* agent: whatever happened and wherever the dialogue went, the aim was to keep the conversation going. Being a nondirective psychotherapist, information content was low, the main tactic being to use keywords from the user's input to fill slots in the system's output. In the following, ELIZA [30, p. 2] has used *boyfriend* and *depressed* to fill pre-prepared templates:

```
User:   Well, my boyfriend made me come here.
ELIZA:  Your boyfriend made you come here.
User:   He says I'm depressed much of the time.
ELIZA:  I am sorry to hear you are depressed
```

To avoid the conversation becoming too repetitive, ELIZA used two ways of introducing variety into its responses. For each keyword there could be a number of possible responses; each was used in turn to introduce some variation. Also previous topics were stored on a stack (thus keeping a record of the conversation) so that, should it be impossible to match a keyword with a template, a previous keyword could be revisited. This had the significant effect of making it seem as if there was some larger dialogue management taking place.

The ELIZA/ALICE model is essentially conversational: the chatbot attempts to maintain a dialogue exchange above all else. The communication of information is very much a secondary objective; hence Weizenbaum's choice of a nondirective psychotherapist.

Both the Alliance & Leicester and O_2 chatbots try to communicate information about products while trying to maintain a dialogue. In particular, they use an avatar figure to represent the computer partner in the chatbot dialogue. Although it might seem attractive from a marketing point of view to present the user with a "chatbot friend" in the hope they will bond with it, many users must be sufficiently ICT-literate and the chatbots so limited that the illusion of a conversational friend is shattered. However, behind such systems, the information content is equivalent to an over-large FAQ. This paper focuses on providing a natural language interface to a set of FAQ-like topics where the number of topics is too large for a conventional WWW-based FAQ and too small for a full database natural language interface system. While a small FAQ list ranging over a very limited topic area is usually an ideal way of presenting information, a larger FAQ list ranging over a broader topic area or areas is less effective. For the information seeker, the organization of the question list may seem unfamiliar or unintuitive and the length of the list makes is difficult to locate the perhaps small piece of information. It may seem that the FAQ writer has not predicted the user's question or the information being sought is given as the part answer to several questions. For the work presented here, the user may not find their question expressed in a form they recognize, perhaps because of differing levels of competence in the language of the FAQ [25, p. 97].

More specifically, the aims of the natural language interface investigated here can be stated as:

1. *robustness* - capable of processing well-formed English or ill-formed either because the user's command of English is poor or because of ellipsis;
2. *low cost* - such a system should use relatively simple techniques to extract meaning from input and to return outputs, thus reducing the cost of implementation and maintenance;
3. *low-skilled maintenance* - it is essential that adding to and modifying the knowledge base of the application should be as simple as possible, allowing changes to be made by IT literate rather than computer science trained colleagues.

As explained above, the context of this investigation was a system for responding to natural language enquiries about applications to MSc courses. Such a system would consist of a WWW interface to a bank of 50-100 topics (i.e. too many for a manageable unhierarchically structured FAQ). Two main ways of accessing the bank of topics were chosen:

– *keywords* - keywords were manually assigned to each topic, together with a weight in the range 1...5 (where 1 was relatively insignificant and 5 extremely significant);

– *sentences* - one or more stereotypical interrogative sentences were assigned to each topic. No weights were assigned to these sentences. (These are referred to in the remainder of the paper as “stereotypical queries”.)

In both cases, it would be relatively easy for non-computer scientists to annotate the topic banks. This system is termed an “infobot” to distinguish its informational and non-conversational functionality from that of chatbots.

Experiments were designed to assess the effectiveness of a number of methods of matching queries with either sets of keywords or stereotypical queries. The latter were also used as the source for syntactically selected sets of keywords.

2. Claims

The main claim made as a result of the experiments is that:

– A Jaro-based string similarity algorithm [10] is at least as effective as the less complex keyword-based methods tested and offers better scalability.

Sub-claims are:

– Abbreviated, terse queries (e.g. “cost of courses”) and lengthy inputs have no significant effect on the performance of the best-performing matching algorithms.
– The best performing matching algorithms are robust when processing “non-native” English.
– Matching with keywords extracted using shallow syntactic techniques offers no improvement in performance.

The methodology was first to establish a corpus of queries from users. This was used as the basis for building the keyword and sentence indexes. Then, each matching method was applied to the corpus to provide a basis of comparison.

3. Preparing a Corpus

To collect a sample of inputs, a simple keyword-based infobot for delivering admissions-related information in response to natural language queries was mounted on the WWW.

This infobot was implemented in SICStus Prolog with a PrologBeans interface to the Java front-end. Users’ inputs were delivered to the Prolog application which extracted keywords or key-phrases and used these to match with keywords or key-phrases associated with “chunks” of informational text (Fig. 1). These informational texts were created after a study of a log of email enquiries received from MSc applicants in the previous of the academic year.

The system was made accessible via the WWW to applicants for MSc courses in the School of Computer Science, University of Birmingham [23] in two phases.

Informational text	Keywords
Our programmes begin on 4th October 2010. Next academic year begins on 26th September 2011.	begin beginning 'academic year' 'starting date'
The on-line application form is at: http://apply.bham.ac.uk/cp/home/loginf.	'online application'

Fig. 1. Rules and keywords from the simple chatbot

3.1. Phase 1: Initial Testing

This was a feasibility study designed to assess whether there were informational texts missing from the system or if extra keywords needed to be added to existing information texts. A subset of about 15% of current MSc applicants were contacted by email, inviting them to use the system. Taking a random sample of the set of current applicants would have been possible but unduly complex, given that the set of applicants changed dynamically as some applications were rejected and new applications were received. Rather, all applicants with surnames beginning with 'S' or 'T' were included in the subset.[3] 121 queries were submitted by members of this subset of applicants. These were analysed with two extra informational texts being added and extra keywords added to some existing informational texts. This resulted in the infobot system that was used in the second phase to produce the corpus used in the experiments described in the remainder of this paper.

3.2. Phase 2: Corpus Collection

The second phase was used to collect a reference sample of queries that might be used to evaluate later systems, to analyse the behaviour of users and to analyse the performance of this simple system. 573 applicants were invited by email to use the system (being applicants with surnames beginning with other than 'S' or 'T'). 357 queries were recorded of which 70 were repeats[4].

All inputs and responses were logged. Each input was manually annotated as one of:

– *Correct* - the input was judged to be grammatical, correctly spelled and the question appropriate to the domain.
– *Correct/spelling error* - an otherwise correct input that contains at least one spelling error.

[3] This subset of surnames was chosen because the spread of nationalities of, and languages spoken by, applicants was better than other subsets of surnames, e.g. 'A' and 'Z'.

[4] A repeat is defined as a user immediately entering an input identical to their previous input.

Examples: How long it takes to finish the *porgram*? How do I know if my online registration is *finnished*?

– *Correct/grammar error* - an otherwise correct input that contains at least one grammatical error.
  Examples: Do i require to attend an interview? Is there any part time programs?
– *Abbreviated* - an input that was too brief (usually lacking a verbal component) for keywords to be reliably identified.
  Examples: Registration? FAQ? why Birmingham?
– *Inappropriate* - the input was either judged to be grammatical, correctly spelled but the question inappropriate to the domain or the input was not English or not natural language.
  Examples: What time is it now? What is your name? Das ist ein scholarship! MumbleJumble,ISupposeThisIsATest, ????, "; OR 1=1".

3.3. Analysis of Users' Inputs

In the email inviting applicants to take part in the trial, it was explained to them that this was a system under development that needed testing. An analysis of the input shows that a substantial number of the enquiries were well-formed and relevant English questions. Some applicants chose to use abbreviated enquiries such that they might use in a general search engine. Inevitably, in the context of a test where there was no identification of individual users, some chose to enter completely irrelevant (and thus inappropriate) queries (Fig. 2).

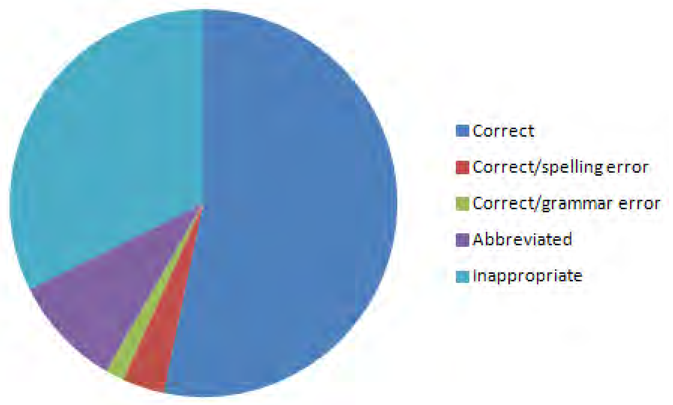


Fig. 2. Classification of inputs

From the log of inputs it could be seen that some users immediately followed their original query with one or more repetitions of the input as if they believed

Table 1. Classification of inputs

Label	Original input		Repeated input		Total input	
	n	%	n	%	n	%
Correct	105	76.64	32	23.36	137	100.00
Correct/incorrect spelling	7	77.78	2	22.22	9	100.00
Correct/incorrect grammar	4	100.00	0	0.00	4	100.00
Abbreviated	22	91.67	2	8.33	24	100.00
Inappropriate	49	59.04	34	40.96	83	100.00

that a repetition would, for some reason, return an alternative response (Table 1). It is very noticeable that users' willingness to repeat input was determined by the nature of their original input. 23.36% of correct inputs were repetitions, whereas only 8.33% of abbreviated queries were repeated, suggesting users realised that their input was too brief. The number of repetitions of inappropriate input was particularly large at 40.96%, perhaps suggesting that such users had a poor initial model of the system and were struggling to refine that model.

3.4. From Input to Corpus

To build a corpus as a tool for testing alternative designs, the inputs were selected as follows. All correct inputs were kept as were correct/grammar errors inputs. Correct inputs with spelling errors were corrected and (unless already present in the corpus) included. The inappropriate inputs were not included in the corpus. Abbreviated inputs were included where it was possible to glimpse some intended meaning. The corpus consisted of 154 queries, including well-formed and less well-formed questions as well as terse non-grammatical queries. Thus the corpus could claim to represent a real-life variety of English performance. The mean length of queries was 6.19 words and the mode was 5 words.

A "response class" set of 68 interpretations was formed. Each query in the corpus was assigned to one of the infobot's response class interpretations. For instance, the input "how long does it take to pursue a master program?" was labelled as a "duration" so that the query would be given the response "Our MSc programmes last for one year". A few response class interpretations were very closely related, for instance "birmingham_location" ("Where is Birmingham") and "location_university" ("Where is Birmingham University"). Such similarity would make the task of retrieval more difficult but reflected the practical difficulties of responding to some queries. Two topics dominated others in the corpus: the cost of tuition fees and the availability of scholarships. There was a noticeable difference between the contents of emails previously sent to admissions tutors and infobot queries: when applicants realised they were communicating with a machine, they felt sufficiently uninhibited to ask about money issues.

4. Experiments on Matching Methods

The matching methods used fell into three groups:

1. Those that used keywords extracted from the query matched against keywords assigned to interpretations from the response class (Section 4.1).
2. Those that matched the whole text of the user's query with one or more stereotypical queries assigned to interpretations from the response class (Section 4.2).
3. Shallow syntactic extraction of keywords from the user's query. The Stanford Parser [13] was used to analyse the stereotypical queries assigned to interpretations drawn from the response class, giving dictionary entries which also included information about keyword co-occurrence *and the ordering of keywords* (Section 4.3).

The results of each experiment were classified into one of three categories:

1. *Correct* - the outcome matched the expected outcome given in the corpus;
2. *Incorrect* - the outcome did not match the expected outcome given in the corpus;
3. *No response* - there was no outcome, for instance because no match was made by the current matching algorithm.[5]

4.1. Keyword-based Matching

Words judged to be significant were manually added to the keyword set.[6] In the following queries from the corpus, the keywords have been underlined:

how many modules
what is the last date of submitting the recommendations

Weights were manually assigned to each keyword, with low weight attached to meaningful but commonly used keywords ("how many" = 1) and high weight to those keywords thought to carry the main content of their queries ("recommendations" = 4). As explained above, each keyword was associated with one or more *interpretations* from the response class; an interpretation here meaning the label of a particular response, for instance the duration example (Sec. 3.4). There were 152 keywords indexing 68 topics.

Simple Keyword Matching This method of matching was not expected to be effective but was used to provide a baseline method against which all other methods could be compared. (It should be viewed as a keyword equivalent

[5] In these experiments, the use of a corpus that excluded irrelevant queries meant that "no response" would be indicative of system failure rather than irrelevant input.
[6] Here "keyword" in understood to mean both single word and multi-word keywords, e.g. "part time".

of the bag-of-words model in document classification.) In the first experiment, weights were ignored. Competing interpretations were judged solely by the number of keywords found in the input. So, if the underlined words are keywords that shared the same interpretation (*deadline_application*):

what is the last date of submitting the recommendations

the score for the *deadline_application* interpretation was 3. Where there was a tie between two or more interpretations, the first occurring interpretation was selected.[7] Results are given in Table 2.

Weighted Keyword Matching Here the weights were summed. So, if the underlined words are keywords that shared the same interpretation (*deadline_application*):

what is the last date of submitting the recommendations

and their weights were:

what - deadline_application - 1
last date - deadline_application - 3
recommendations - deadline_application - 1

the sum was 5. Where there was a tie, the first occurring interpretation was selected. Results are given in Table 3.

Simple/Weighted Keyword Matching The sum of the weights and the number of keywords found were summed. Again, using the example:

what is the last date of submitting the recommendations

where the simple keyword score was 3 and the weighted keyword score was 5, the simple weighted keyword was 8. Where there was a tie, the first occurring interpretation was selected. Results are given in Table 4. (It might seem more reasonable to calculate the mean weight of keywords by dividing the summed weight by the number of keywords but this gave a slightly worse performance.)

4.2. Sentence-based Matching (String Similarity)

One or more stereotypical queries were written for each interpretation. For instance, for the "duration" interpretation, the stereotypical queries were:

how long does a masters degree take?
how long does the program take?
how long does the programme take?

[7] In a practical system, it would be necessary to employ some principled way of choosing between tied interpretations, for instance by allowing the user to choose the response best suited to their query. This, however, is an evaluation where the emphasis is on mechanically selecting the most appropriate interpretation.

Table 2. Simple keyword matching: results

Outcome	n	%
Correct	105	68.18
Incorrect	49	31.82
No response	0	0.00
Total	154	100.00

Table 3. Weighted keyword matching: results

Outcome	n	%
Correct	118	76.62
Incorrect	36	23.38
No response	0	0.00
Total	154	100.00

Table 4. Simple and weighted keyword matching: results

Outcome	n	%
Correct	119	77.27
Incorrect	35	22.73
No response	0	0.00
Total	154	100.00

how long is the msc?
what is the duration of the course?
what is the duration of the program?
what is the duration of the programme?

The matching process was to compute the string similarity between input (here drawn from the corpus) and the stereotypical queries. There are a number of string similarity algorithms that could be used [7]. Those selected were:

– *Jaro proximity*[8] (comparing inputs/stereotypical questions forwards and backwards);
– *Jaro-Winkler proximity* (forwards and backwards).

These algorithms were devised tor comparing strings such as personal names where strings would be short and errors likely to be transpositions over fairly short distances.

The Jaro algorithm compares two strings such as ‘Martha’ and ‘Marhta’. One string is scanned, character-by-character. (In this example, ‘Martha’ is taken as the first string.) A moveable window is placed over the second string. The width of the windows is computed as half the length of the longer string - 1. The window moves in synchrony with the scanning of the first string. A match between a character in the first string can only occur within the window. In the example, the emboldened characters are matches while underlined characters are within the current window:

Martha M**a**rtha Ma**r**tha Mar**t**ha Mart**h**a Marth**a**
Marhta M**a**rhta Ma**r**hta Marh**t**a Mar**h**ta Marht**a**

[8] Confusingly, “proximity” and “distance” seem to be used interchangeably in the literature.

In a second scan, the number of transpositions is counted. The calculation of Jaro proximity is:

$$\frac{1}{3} \times \frac{matches}{length(string_1)} + \frac{matches}{length(string_2)} + \frac{matches - (transpositions//2)}{matches} \quad (1)$$

(It should be said that the detailed implementation of transposition matching is not intuitive: “The number of transpositions . . . is computed somewhat differently from the obvious manner.” [32, p. 10].)

The Jaro-Winkler algorithm is founded on the observation that transposition errors are less likely to occur in names or addresses within the initial *n* character positions (usually *n* = 4). Winkler extended the Jaro algorithm by adding a threshold of similarity (usually 0.70). For two strings with a Jaro proximity of 0.7 or more, the initial *n* characters are matched for absolute similarity (giving a “match length”). Thus, Jaro-Winkler proximity is calculated as:

$$JaroProximity + (length(match) \times position \times (1.0 - JaroProximity)) \quad (2)$$

Jaro [10] and Jaro-Winkler [31] algorithms have a record of good performance [7]. Whilst developed for character-by-character processing of names, in these experiments the comparison was word-by-word and thus inputs in these experiments were relatively short and had a number of words comparable to the number of letters in names. The rationale was that only a very limited domain of words could be reasonably used to request information on any particular topic. Also, the form of queries could be very standardised with only minor variations, for instance because of choice of function words (e.g. “a”, *v.* “the”) or that there would be minor variations caused by an applicant’s imperfect command of English. In both cases, a Jaro-based algorithm would seem to offer a way of pairing a stereotypical query with a closely related user query. It should be noted that the proportion of matching words (either directly aligned or transposed) was lower than the proportion of matching characters in a personal name [16].

Jaro Proximity String Similarity The standard Jaro algorithm uses a matching window defined as:

$$\frac{max(length(string_1), length(string_2))}{2} - 1 \quad (3)$$

A number of runs were tried to investigate the effect of longer window sizes, leading to the conclusion that Jaro’s original window size was optimal.

Two experiments were carried out: searching from beginning to end of input/stereotypical queries (Table 5); searching from end to beginning (Table 6).

Jaro-Winkler Proximity String Similarity This modification of the Jaro algorithm rewards matches at the beginning of the two strings, specifically in the first four positions. It was used in these experiments because it seemed that the beginning of a query (e.g "how many ...", "are there any ...") was significant in the query's meaning. It was hypothesised that it would be more significant still for comparing the endings of queries because many questions in English begin with the same sequence of words, thus the endings of queries should be more discriminating. The results are presented in Tables 7 and 8.

Table 5. Jaro (forward): results

Outcome	n	%
Correct	118	76.62
Incorrect	26	23.38
No response	0	0.00
Total	154	100.00

Table 6. Jaro (backwards): results

Outcome	n	%
Correct	105	68.18
Incorrect	49	31.82
No response	0	0.00
Total	154	100.00

Table 7. Jaro-Winkler (forward): results

Outcome	n	%
Correct	117	75.97
Incorrect	37	24.03
No response	0	0.00
Total	154	100.00

Table 8. Jaro-Winkler (backwards): results

Outcome	n	%
Correct	104	67.53
Incorrect	50	32.47
No response	0	0.00
Total	154	100.00

Jaro/Jaro-Winkler with Stemming Both the forward and backwards versions of the Jaro and Jaro-Winkler algorithms were supplemented by stemming the input and stereotypical queries. The stemming algorithm used was the Porter algorithm [17]. The query:

what is the last date of submitting the recommendations

would be reduced to:

what i the last dat of submit the recommend

Results are given in Tables 9 to 12.

4.3. Sentence-based Matching (Shallow Syntax)

The hypothesis was that relative order of keywords is intrinsically important over and above mere co-occurrence of keywords. However, choice of keywords should not be left to human assignment (as in Section 4.1) but chosen using syntactic information. Additionally, the order of keywords relative to other keywords is significant as may be the distance between any two keywords.

Table 9. Jaro (forward-stemmed): results

Outcome	n	%
Correct	121	78.57
Incorrect	33	21.43
No response	0	0.00
Total	154	100.00

Table 10. Jaro (backwards-stemmed): results

Outcome	n	%
Correct	106	68.83
Incorrect	48	31.17
No response	0	0.00
Total	154	100.00

Table 11. Jaro-Winkler (forward-stemmed): results

Outcome	n	%
Correct	119	77.27
Incorrect	35	22.73
No response	0	0.00
Total	154	100.00

Table 12. Jaro-Winkler (backwards-stemmed): results

Outcome	n	%
Correct	106	68.83
Incorrect	48	31.17
No response	0	0.00
Total	154	100.00

The aim was to build a matching algorithm that, for any given keyword, had associated with it an ordered list of keywords that could (and should) occur in the query *before* the given keyword and an ordered list of keywords that could (and should) occur in the query *after* the given keyword. For the query:

what is the last date of submitting the recommendations

when the algorithm selected "submitting" as the given keyword, the ordered list of prior keywords would be [last, date] and the subsequent keywords would be [recommendations].

This method of matching required more pre-processing than the keyword-based matching and Jaro-based matching. To "learn" a set of possible keyword combinations, the Stanford Parser [13] was used to analyse each of the stereotypical queries. This produced a syntactic structure:

```
(ROOT
  (SBARQ
    (WHNP (WP what))
    (SQ (VBZ is)
      (NP
        (NP (DT the) (JJ last) (NN date))
        (PP (IN for)
          (S
            (VP (VBG submitting)
              (NP (NNS recommendations)))))))
    (. ?)))
```

from which keywords were extracted. Three sets of syntactic classes were chosen:

1. nouns and adjectives (JJ, NN, NNS)[9] giving from the example above the keywords {last, date, recommendations}.
2. verbs, nouns and adjectives (JJ, NN, NNS, VRB, VRG, VRB, VRP) giving from the example above the keywords {last, date, submitting, recommendations}.
3. WH-adverbs, verbs, nouns and adjectives (JJ, NN, NNS, VRB, VRG, VRB, VRP, WRB) giving from the example above the keywords {what, last, date, submitting, recommendations}.

For each keyword, a dictionary entry was formed giving the keyword and the ordered "before" and "after" keyword lists thus enforcing a very shallow amount of (linear) syntactic structure:

dictionary(date, [what, last], [submitting, recommendations])

The matching algorithm scanned the user's query. Each word in the input having a dictionary entry was identified as a "main keyword". The whole query was them matched as follows:

1. Each keyword in the "before" list was sought in the user's query before the occurrence of the current main keyword. If a "before" keyword was found, then any subsequent "before" keyword had to occur afterwards in the query but before the "main keyword". For instance, with the "before" keyword list [what, last], there would be a complete match with:

   $\underline{\text{what}}$ is the $\underline{\text{last}}$ date . . . ?

   but would be an incomplete match of:

   $\underline{\text{last}}$ what is the date for submitting recommendations?

   In this second example, there is an incomplete match because the algorithm requires the keywords to occur in order.

2. Each keyword in the "after" list was sought in the user's query after the occurrence of the current main keyword. The same requirement of ordering was enforced.

Keyword matches were scored. Each valid occurrence of a keyword was given a point, so

$\underline{\text{what}}$ is the $\underline{\text{last}}$ $\underline{\underline{\text{date}}}$ for $\underline{\text{submitting}}$ $\underline{\text{recommendations}}$?

scored 5, whereas:

$\underline{\text{last}}$ $\underline{\underline{\text{date}}}$ is what for $\underline{\text{submitting}}$ $\underline{\text{recommendations}}$?

scored 4 as would:

$\underline{\text{last}}$ what is the $\underline{\underline{\text{date}}}$ for $\underline{\text{submitting}}$ $\underline{\text{recommendations}}$?

[9] The Stanford Parser uses the Penn Treebank tagset.

In addition, a mean distance was calculated so that queries with fewer non-keywords between keywords would be rewarded. Given competing interpretations, the interpretation with the highest keyword score and (in the case of interpretations with the same keyword score) then with the lowest mean difference between keywords was ranked first (with the first found being arbitrarily chosen amongst equal scoring interpretations).

As stated above (page 1715), three slightly differing syntactic classes were used to construct the keyword dictionary. One of these (nouns and adjectives) was used in conjunction with the Porter stemming algorithm, so that all dictionary keywords (including those in the before and after lists) and the stereotypical queries from the test corpus were stemmed. The hypothesis was that stemming would increase the number of matching keywords and thus increase accuracy. Results for all four experiments are given in Tables 13 to 16.

Table 13. Shallow syntactic (nouns and adjectives): results

Outcome	n	%
Correct	104	67.53
Incorrect	50	32.47
No response	0	0.00
Total	154	100.00

Table 14. Shallow syntactic (verbs, nouns and adjectives): results

Outcome	n	%
Correct	105	68.18
Incorrect	49	31.82
No response	0	0.00
Total	154	100.00

Table 15. Shallow syntactic (WH-adverbs, verbs, nouns and adjectives): results

Outcome	n	%
Correct	84	54.55
Incorrect	64	41.56
No response	6	3.90
Total	154	100.00

Table 16. Shallow syntactic (nouns and adjectives) stemmed: results

Outcome	n	%
Correct	91	59.09
Incorrect	60	38.96
No response	3	1.95
Total	154	100.00

5. Interpretation of Results

The shallow syntactic matching algorithm including WH-adverbs, verbs, adjectives and nouns was the worst-performing method. The range between the worst (54.55%) and the best-performing methods (78.57%) is not particularly narrow but disappointingly poor at the high end where only four out of five queries would be correctly answered.

There is little to choose between Jaro (forward-stemmed) (78.57%), simple/weighted keywords (77.27%) and Jaro-Winkler (forward-stemmed) (77.27%).

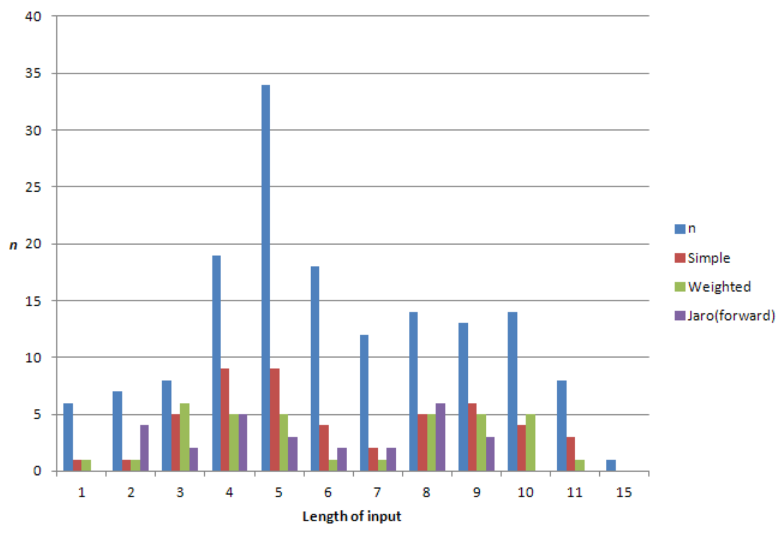


Fig. 3. Incorrect interpretations by method of matching

5.1. Simple Matching

The simple method (here used for baseline comparison) is almost the least reliable because its only strategy of choice is the number of keywords. So from Fig. 3, it can be seen that when word length rises beyond three, the simple method tends to perform less well. Essentially, given the restricted domain (and hence vocabulary of the application) the more words a query such as:

by when do I need to accept a course offer?

contains, the greater probability there is that the query contains keywords for an inappropriate interpretation. The likelihood of incorrect interpretations was compounded by the lack of a principled way of selecting between alternatives.

5.2. Simple/Weighted Matching

The performance of simple/weighted matching was almost as good as the best method. There were some queries which it processed incorrectly but which the other methods processed correctly. Examples are:

why Birmingham University?
why study at Birmingham University?

In both cases, it provided an interpretation for the very closely related query:

why should I come to Birmingham?[10]

[10] Where this sentence is understood to mean the city of Birmingham.

It seems at this level of subtlety, the simple/weighted matching method was unable to distinguish satisfactorily between competing interpretations.

5.3. Jaro (Forward-Stemmed)

Of the eight Jaro-based methods, Jaro (forward-stemmed) was most effective (78.57%). It might have been expected to work even better. Fig. 3 fails to record any particular pattern to failures amongst three similarly scoring methods, for instance they did not mainly occur amongst queries of shorter lengths. Neither did the errors occur amongst longer queries. The stemming algorithm worked to reduce variability between users' expressions. Thus users' variability of expression became less significant and one stereotypical query would match with a larger number of users' queries. This is supported by an examination of queries which keyword methods could resolve correctly but which Jaro (forward-stemmed) failed. The most extreme was:

> when do I need to finalize my course optional modules?[11]

Without a sufficiently similar stereotypical query, it would be more a matter of luck if the nearest matching stereotypical query had an appropriate interpretation.

5.4. Shallow Syntax

All four variants of the very shallow syntactic search algorithm produced consistently poor results. It would be reasonable to expect the inclusion of verbs to increase reliability as they have a crucial role in specifying complements which, in turn are realized primarily by adjectives and nouns within noun phrases. In fact, it produced no better results than the simplest (and crudest) keyword matching algorithm. The addition of stemming decreased accuracy still further. This was because it increased the number of candidate interpretations of a query without in any way contributing to an improvement in their ranking. Thus it increased the likelihood that the matching algorithm would be unable to select the current interpretation from a larger set of candidates.

5.5. Scalability

The corpus was, at 154 entries, small-scale. Nonetheless, it is possible to discern some problems of scalability even at this size. Simple/weighted matching failed where it had to choose between two closely related interpretations. An attempt to increase the success rate from 77.27% would require, in part, more keywords. This evaluation suggests that increasing keywords in a limited domain (with the likelihood that one keyword will index multiple interpretations) would bring a decrease in accuracy.

[11] A simpler way of expressing this might have been: "what is the deadline for choosing optional modules?"

While it was difficult to see a consistent pattern of failure for the best performing Jaro (forward-stemmed) (78.57%), there is some evidence that a lack of stereotypical queries was a cause of failure. Thus an increase in the coverage would, unlike simple/weighted matching, improve performance. In summary, Jaro (forward-stemmed) has the potential for scalability; simple/weighted matching does not.

5.6. Lack of Input *v.* Correctness

It was hypothesised that it would be more difficult to answer shorter queries correctly. Fig. 3 gives only very limited evidence of this. At a query length of two, Jaro (forward) did badly; at query length of three words, keyword-based matching did less well. However, there is no clearly significant evidence and so the hypothesis can be neither confirmed nor denied.

5.7. Processing Ungrammatical Inputs

It was hypothesised that simple/weighted matching would outperform Jaro (forward-stemmed) in processing ungrammatical inputs. The proportion of ungrammatical inputs[12] (less than 10%) was small. Errors of grammar (usually number/person agreement failures) were either very local (“a courses”) or longer distance:

when does the university starts?

For the keyword-based systems, there was no notion of agreement: each keyword was independent and so number/person agreement could not be enforced, even if desirable. Agreement is explicit in Jaro-based methods because, assuming stereotypical queries will be well-formed, there would be no complete match between the users’ inputs and the stereotypical queries. However, for longer distance ungrammaticality to be possible, there has to be a relatively long input and so the Jaro score would be less reduced than it would be with very local ungrammaticality in short inputs.

Adding stemming worked against any effects of agreement. By reducing word forms to their stems, morpho-syntactic information was removed and so it played no part in the matching process. This had the effect of improving matching.

6. Related Work

ELIZA [28] was one of three well-recognized natural language processing systems developed at much the same time. Raphael’s SIR system [19] and Bobrow’s STUDENT [4] answered mathematical questions. That they were both based on very formal domains of knowledge contributed to their success. As

[12] Not to be confused with abbreviated input e.g. “registration deadline?”

Weizenbaum himself noted “from the purely technical programming point of view, the psychiatric interview [which ELIZA modelled] has the advantage that it eliminates the need for storing explicit information about the real world.” [29, p. 474], [14, pp. 28-41; 110-111].

ELIZA has a long-enduring popularity. Implementations are still widely available[13] and its reputation has outlasted STUDENT and SIR. Much of ELIZA’s reputation is attributable to its domain: a psychiatric therapist was novel and still is very accessible (and even attention catching) to the less-than-expert AI practitioner. This alone does not account for its reputation and longevity: it was an early example of robust processing in that it could deal with ungrammatical input and conversations where topics were abandoned and returned to later, or the changing themes of the conversation were unrelated. Most of all, ELIZA offered an early prospect of a system that could pass the popularized notion of the Turing Test. Weizenbaum himself (incidentally rather than intentionally, it seems) raised this prospect in reporting that, although his secretary knew that in using it she was “talking to a machine”, she asked “‘Would you mind leaving the room, please?’ I [i.e. Weizenbaum] believe the anecdote testifies to the success with which the program maintains the illusion of understanding.” [29, p. 478].

Weizenbaum attempted to show the potential for question-answering systems using a refinement of the ELIZA system[14], i.e. by developing it into what this paper terms an infobot. He chose to demonstrate his ideas by providing another system to answer maths problems, which necessitated the addition of an expression evaluator. Changes were made to the store of templates. It was divided into what Weizenbaum likened to a routine (i.e. controlling set of templates which he termed a “script”) and subroutines (i.e. groups of closely related templates on very narrow topics). The reason for this hierarchical organization seems to be both practical (in that it allowed larger dialogues to be handled in small memories) and theoretical (reflecting a concern with being able to distinguish between alternative word senses). This development did not have the impact of the first ELIZA, failing to develop the ELIZA techniques in any important way, and remains a curiosity.

Shapiro and Kwasny’s 1975 development of an ELIZA infobot [24] proved more influential. In part, their success can be attributed to the development of real-time time-sharing interactive computing. Newly developing operating systems and databases were applications which allowed greater user interaction but also required quite detailed knowledge of command languages. A common theme amongst the approaches being developed to help systems was how the user could find what they needed without first knowing the appropriate technical vocabulary or command language. Shapiro and Kwasny demonstrated a straightforward development of ELIZA to provide help for the DECsystem-10.

[13] For instance: `http://www.chayden.net/eliza/Eliza.html` (Java).
[14] Confusingly Weizenbaum also called his new system ELIZA [29, p. 478, col. 1].

Their evaluation was slight by modern standards but their work was highly regarded for providing access to naïve or casual users[15] [9], [18], [20].

1991 saw the beginning of the Loebner Prize contests in which competing chatbots attempt to persuade a jury of their ability to pass the Turing Test. To succeed, it is necessary for a chatbot to chat, irrespective of topic or quality of responses. In this respect, the Loebner Prize has not directly contributed to the development of infobots. The attachment of a virtual character, an avatar, to a chatbot has become fairly common [2]. There has been a return to the chatbot as therapist with avatars being added to systems for education (e.g. [21]) and psychiatric therapy and counselling [26]. Some organizations, such as Ikea, Alliance & Leicester bank and the O_2 mobile phone company (see p. 1704 above) have used chatbots and avatar interfaces as product advisors. The development of such systems has been encouraged in part by the availability of the ALICE system with the AIML (Artificial Intelligence Markup Language) [1]. This provides a more formal way of specifying ELIZA-like templates together with a slightly more sophisticated matching algorithm which allows for some non-determinism. One addition beyond that of the original ELIZA is the "predicate" feature which allows the "botmaster" to write rules that contain is/or facts, for instance: "Samuel Clemens is Mark Twain". This provides a method, albeit unsophisticated, to store factual information.

In common with any developer of a natural language-interfaced information retrieval system, infobot developers have the twin problems of ensuring the coverage of their information resource is correct and complete and their interface covers the range of inputs users wish to employ. The capability for the user to add novel ways of expressing queries was introduced in Weizenbaum's second ELIZA [29, p. 479] where there is an impressive learning of German queries. This kind of learning was included in CSIEC, where it stored all inputs and used them in its responses [11]. The difficulty for infobots is that extending the language coverage alone might not be sufficient: it may also be necessary to extend or improve the information content. CSIEC allowed the user to add new information (i.e. by adding new facts which would be matched to existing templates and so were analysable, such as "Australia is in the Pacific"). Learning additional information by the kinds of infobot discussed in this paper would be problematic because the information to be added would usually be beyond simple facts (e.g. "Australia is in the Pacific") and the quality of newly learned information would have to be assured by the system operators.

Some infobots have been part of larger systems that include, amongst other components, a database. Sammut's system [22] provided an infobot for a museum collection. The pattern matching of natural language inputs to rules (written in production rule form) did not extend the capabilities of ELIZA (or even ALICE) and it is not clear that the incorporation of a database made any differ-

[15] Cuff [8, p. 168] offered a more rigorous analysis of what was meant by "casual user", perhaps from a less favourable standpoint when he stated: "...the author's [sic] discussion is a piece of special pleading for a natural language understanding program which will explain unfamiliar parts of a computer system."

ence to the processing of natural language inputs. Similar comments apply to a system that provides information about student loans [15]. Regretfully, there is often a lack of rigorous evaluation in this work, the notable exception being Carroll and McKendree's evaluation of three types of interface (including ELIZA) to expert systems [6].

Some researchers have, like the work reported here, tried to find alternative ways of matching inputs with FAQ-type information sources. Banchs and Li developed IRIS, a system they described as "a chat-oriented dialogue system." [3] Rather than using templates, they used a vector space model, searching over previous dialogues. Information retrieval systems for FAQs that made no claim to chatting (i.e. are less infobot and more conventional information retrieval system) have used similar, statistically-based matching techniques [12], [5]. Both of these systems identify questions within FAQs and use these to match with the users' queries. This is similar to the use of "stereotypical queries" in the current work.

Shallowness in language analysis is, it seems, a vague term. The work described here is "very shallow" in that, where is uses syntactic processing, it does so only to isolate keywords of particular grammatical classes. Wang, Ming and Chua use slightly less shallow parsing in that they (more conventionally) isolate phrase groups (e.g. VP) to find similar questions which have been asked of services such as Yahoo. They are able to claim that their technique offers robustness when presented with ungrammatical inputs [27]. At the opposite extreme, Sneiders' interpretation of shallow is, if anything, as shallow as the weighted keyword matching presented here. "Shallow language understanding" is implemented in what he terms "prioritized keyword matching" where he divides keywords into four groups: required, optional, forbidden and stop-list (i.e. high-frequency function words). There is no syntactic analysis. Perhaps to solve problems of conflicting word senses, Sneiders uses multiple lexicons, as many as one per FAQ text [25].

The work surveyed shows the tension between chatting – the need to keep the conversation going whatever the topic of ungrammaticality of the input – and the desire to provide users with information. Infobots (as opposed to chatbots) are very much rarer in the literature. While they can respond robustly to all inputs, the accuracy of their responses is disappointing. Work on developing template matching, such as reported here, has been rare. Those systems that have provided information retrieval for the kind of FAQ system used in this work have tended to use statistical techniques with limited robustness.

7. Conclusions and Further Work

A best correctness rate of 78.57% is not high enough for an effective system. The Jaro (forward-stemmed) method offers the possibility of further improvement because it is scalable, thus allowing more stereotypical queries to be used. In particular, it performed well on closely related sentences and less well on longer sentences not closely represented in the stereotypical query store.

The target application of a postgraduate application enquiry system would be used by native and non-native English speakers. There is no evidence that ungrammatical queries led to serious deterioration of performance of the Jaro (forward-stemmed) string similarity algorithm.

The problem with the use of the Jaro (forward-stemmed) method is acquiring stereotypical queries. To this end it is proposed that, in the target application, users be allowed to decide if the system has answered their question or not. If their response in positive, their query could be added to the store of stereotypical queries. Thus the system would, in a limited way, be capable of learning. In this way, it would have a more limited learning capability than those systems (e.g. CSIEC [11]) that seek knowledge from the user.

There is further work that could explore the capabilities of keyword-based searches. First, a limited dictionary of synonyms could be used to normalise queries. So, instances of “MSc”, “master”, “masters”, “MSc in”, “MSc of”, etc. could be normalised to one chosen form. This would reduce the number of keywords to be stored and make it easier to keep keywords and their weights consistent with other keywords and weights. Second, as very shallow syntactic techniques decrease effectiveness, it would seem sensible to investigate more ELIZA-like techniques by, for instance, returning to templates for matching where the system has a number of patterns of the form:

```
what is the deadline for KEYWORD(S)?
```

However, this could overcomplicate the system, leading to poorer performance. It would require a more sophisticated keyword system, perhaps of a predicate/argument structure (e.g. deadline(option_choice)). This in turn would be more difficult to use with abbreviated (“Google-like”) queries.

Acknowledgments. We wish to thank those anonymous applicants who tested the systems for us and Professor Achim Jung and Dr Alberto Simões for their support.

References

1. ALICE Artificial Intelligence Foundation: AIML: Artificial intelligence markup language, www.alicebot.org/aiml.html, [Accessed 31 May 2013].
2. Allen, C.: Artificial life, artificial agents, virtual realities: technologies of autonomous agency. In: Floridi, L. (ed.) The Cambridge Handbook of Information and Computer Ethics, chap. 13, pp. 219–233. Cambridge University Press, Cambridge (2010)
3. Banchs, R.E., Li, H.: IRIS: a chat-oriented dialogue system based on the vector space model. In: ACL 2012: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, Jeju, Jeju Island, South Korea, 8-14 July 2012. pp. 37–42. Association for Computational Linguistics, Stroudsburg, PA (2012)
4. Bobrow, D.G.: Natural language input for a computer problem solving system. In: Minsky, M.L. (ed.) Semantic Information Processing, pp. 146–226. MIT Press, Cambridge, MA (1968)
5. Burke, R.D., Hammond, K.J., Kulyukin, V., Lytinen, S.L., Tomuro, N., Schoenberg, S.: Question answering from frequently asked question files: experiences with the FAQ finder system. AI Magazine 18(2), 57–65 (1997)

6. Carroll, J.M., McKendree, J.: Interface design issues for advice-giving expert systems. Communications of the ACM 30(1), 14–32 (1987)
7. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string distance metrics for name-matching tasks. In: IIWeb-03: proceedings of the IJCAI-2003 Workshop on Information Integration on the Web, Acapulco, 9-10 August 2003. pp. 73–78. IJCAI Press, Palo Alto, CA (1993)
8. Cuff, R.N.: On casual users. International Journal of Man-Machine Studies 12(2), 163–187 (1980)
9. Houghton, R.C.: Online help systems: a conspectus. Communications of the ACM 27(2), 126–133 (1984)
10. Jaro, M.: Advances in record linkage methodology as applied to the 1985 census of tampa florida. Journal of the American Statistical Society 84(406), 414–420 (1989)
11. Jia, J.: CSIEC: a computer-assisted English learning chatbot based on textual knowledge and reasoning. Knowledge-based Systems 22(4), 249–255 (2009)
12. Jijkoun, V., de Rijke, M.: Retrieving answers from frequently asked questions pages on the web. In: CIKM '05: proceedings of the 14th conference on Information and Knowledge Management, Bremen, 31 October-5 November 2005. pp. 76–83. ACM, New York (2005)
13. Klein, D., Manning, C.: Accurate unlexicalized parsing. In: Proceedings of the 41st Meeting of the Association for Computational Linguistics. pp. 423–430. Association for Computational Linguistics, Stroudsburg, PA (2003)
14. Nilsson, N.J.: The quest for artificial intelligence: a history of ideas and achievement. Cambridge University Press, Cambridge (2010)
15. Owda, M., Bandar, Z., Crockett, K.: Conversation-based natural language interface to relational databases. In: WI-IATW '07: proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - workshops, Silicon Valley, CA, 5-12 Nov. 2007. pp. 363–367. IEEE Press, New York (2007), presented at Workshop on Communication between Human and Artificial Agents (CHAA-07)
16. Polatidis, N.: Chatbot for admissions. School of Computer Science, University of Birmingham (2011), unpublished MSc dissertation
17. Porter, M.: An algorithm for suffix stripping. Program 14(3), 130–137 (1990)
18. Price, L.A.: Thumb: an interactive tool for accessing and maintaining text. IEEE Transactions on Systems, Man and Cybernetics 12(2), 155–161 (1982)
19. Raphael, B.: SIR: a computer program for semantic information retrieval. In: Minsky, M.L. (ed.) Semantic Information Processing, pp. 33–145. MIT Press, Cambridge, MA (1968)
20. Relles, N., Price, L.A.: A user interface for online assistance. In: ICSE 1981: proceedings of the 5th International Conference on Software Engineering, San Diego, CA, 9-12 March 1981. pp. 400–408. IEEE Press, New York (1981)
21. Rothkrantz, L.: E-learning in virtual communities. Communication & Cognition 42(1 & 2), 35–52 (2009)
22. Sammut, C.: Managing context in a conversational agent. Linkoping Electronic Articles in Computer & Information Science 3(7) (2001)
23. Satterthwaite, S.: Prolog-Java chatbot for postgraduate admissions in the School of Computer Science. School of Computer Science, University of Birmingham (2010), unpublished MSc dissertation
24. Shapiro, S.C., Kwasny, S.C.: Interactive consulting via natural language. Communications of the ACM 8(8), 459–462 (1975)

25. Sneiders, E.: Automated FAQ answering: continued experience with shallow language understanding. In: Chaudhri, V., Fikes, R. (eds.) Question Answering Systems: papers from the 1999 AAAI Fall Symposium, North Falmouth, MA, 57 November 1999. pp. 97–107. AAAI Press, Palo Alto, CA (1999), Technical Report FS-99-02
26. Tantam, D.: The machine as psychotherapist: impersonal communication with a machine. Advances in Psychiatric Treatment 12(6), 416–426 (2006)
27. Wang, K., Ming, Z., Chua, T.S.: A syntactic tree matching approach to finding similar questions in community-based QA services. In: SIGIR '09: proceedings of the 32nd International ACM SIGIR conference on research and development in Information Retrieval Boston, MA, 19-23 July 2009. pp. 187–194. ACM Press, New York (2009)
28. Weizenbaum, J.: ELIZA: a computer program for the study of natural language communication between man and machine. Communications of the ACM 9(1), 36–45 (1966)
29. Weizenbaum, J.: Contextual understanding by computers. Communications of the ACM 10, 474–480 (1967)
30. Weizenbaum, J.: Computer power and human reason: from judgement to calculation. Penguin, Harmondsworth (1984)
31. Winkler, W.: String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. In: Proceedings of the Section on Survey Research Methods. pp. 354–359. American Statistical Association, Washington, D.C. (1990)
32. Winkler, W.: Overview of record linkage and current research directions. Research report series Statistics 2006-2, Statistical Research Division, U.S. Census Bureau, Washington, D.C. (2006)

Peter Hancox is a Senior Lecturer in the School of Computer Science at the University of Birmingham. He holds a BA and was awarded a PhD for work on the machine translation of limited texts. His research interests are in both natural language processing and parallel implementations of constraint logic programming languages.

Nikolaos Polatidis is a PhD student at the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. He received his Master's degree in Internet Software Systems from the University of Birmingham, UK, having previously completed his BSc in Computer Science at Heriot-Watt University, Edinburgh. His research interests include recommender systems, mobile technologies and natural language processing.

Received: December 2, 2012; Accepted: August 12, 2013.

DOI: 10.2298/CSIS121130060L

Using proximity to compute semantic relatedness in RDF graphs

José Paulo Leal

CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
Porto, Portugal
zp@dcc.fc.up.pt

Abstract. Extracting the semantic relatedness of terms is an important topic in several areas, including data mining, information retrieval and web recommendation. This paper presents an approach for computing the semantic relatedness of terns in RDF graphs based on the notion of *proximity*. It proposes a formal definition of proximity in terms of the set paths connecting two concept nodes, and an algorithm for finding this set and computing proximity with a given error margin. This algorithm was implemented on a tool called Shakti that extracts relevant ontological data for a given domain from DBpedia – a community effort to extract structured data from the Wikipedia. To validate the proposed approach Shakti was used to recommend web pages on a Portuguese social site related to alternative music and the results of that experiment are also reported.

Keywords: semantic similarity, semantic relatedness, ontology generation, web recommendation, processing Wikipedia data

1. Introduction

Searching effectively on a comprehensive information source as the Web or just on the Wikipedia usually boils down to using the right search terms. Most search engines retrieve documents where the searched terms occur exactly. Although stemming search terms to obtain similar or related terms (e.g. synonyms) is a well known technique for a long time [15], it is usually considered irrelevant in general and search engines of reference no longer use it [1].

Nevertheless, there are cases where semantic search, a search where the meaning of terms is taken in consideration, is in fact useful. For instance, to compare the similarity of genes and proteins in bio-informatics, to compare geographic features in geographical informatics, and to relate multiword terms in computational linguistics.

The motivation for this research in semantic relatedness comes from another application area, recommendation. Most recommenders use statistical methods, such as collaborative filtering, to make suggestions based on the choices of users with a similar choice pattern. For instance, an on-line library may recommend a book selected by other users that also bought the books already in the shopping basket. This approach has a cold start issue: what should be recommended to someone that was not yet bought or searched anything? to whom recommend a book that was just published and few people have bought?

An alternative approach is to base recommenders on an ontology of recommend items. An on-line library can take advantage from the structure of an existing book classification, such as the Library of Congress Classification system. However, in many cases such classification does not exist and the cost of creating and maintaining an ontology would be unbearable. This is specially the case if one intends to create an ontology on a unstructured collection of information, such as a folksonomy.

Consider a content-based web recommendation system for a social network, where multimedia content (e.g. photos, videos, songs) is classified by user-provided tags. One could simply recommend content with common tags but this approach would provide only a few recommendations since few content items share the exact same tags. In this case, to increment the number of results, one could search for related tags. For instance, consider that your content is related to music that users tag with names of artists and bands, instruments, music genres, and so forth. To compute the semantic relatedness among tags in such a site one needs a specific ontology adapted to this type of content.

It should be noticed that, although several ontologies on music already exist, in particular the Music Ontology Specification[1], they are not adjusted to this particular use. They have a comprehensive coverage of very broad music genres but lack most of the sub-genres pertinent to an alternative music site. The same would happen with lexical thesaurus, such as WordNet. To create and maintain an ontology adjusted to a very specific kind the best approach is to extract it from an existing source. The DBpedia[2] is a knowledge base that harvests the content of the Wikipedia and thus covers almost all imaginable subjects. It is based on an ontology that classifies Wikipedia pages and on mapping rules that convert the content of Wikipedia info-boxes and tables into Resource Description Framework (RDF) triplets available from a SPARQL endpoint (SPARQL is a recursive acronym for SPARQL Protocol and RDF Query Language).

In this paper we present Shakti, a tool to extract an ontology for a given domain from DBPedia and use it to compute the semantic relatedness of terms defined as labels of concepts in that ontology. One of the main contributions of this paper is the algorithm used for computing relatedness. Most ontologies based algorithms for computing relatedness assume that ontologies are taxonomies or at least direct acyclic graphs, which is not generally the case of an ontology extracted from DBpedia. Also, these algorithms usually focus on a notion of distance. Instead the proposed algorithm is based on a notion of proximity. Proximity measures how connected two terms are, rather than how distant they are. A term may be at the same distance to other two terms but have more connections to one than the other. Terms with more connections are in a sense *closer* and thus have an higher proximity.

[1] `http://musicontology.com/`
[2] `http://dbpedia.org/About`

The rest of this paper is organized as follow. The following section presents related work on semantic relatedness algorithms and on the use of knowledge bases such as DBpedia. Section 3 is the main section as it introduces the concept of proximity, provides a formal definition of this concept in terms of sets of paths, and presents an algorithm for computing proximity based on the proposed definition. Section 4 presents the design and implementation of Shakti, a tool implementing the proposed algorithm. The following section describes a use of Shakti to populate a proximity table of a recommender service that was used as validation of the proposed approach. The final section summarizes the contributions of this paper and highlights future directions of this research.

2. Related Work

This section summarizes the concepts and technologies that are typically used as basis for the computation of semantic relatedness of terms in the Web.

2.1. Knowledge representation

Currently, the Web is a set of unstructured documents designed to be read by people, not machines. The semantic web — sponsored by W3C - aims to enrich the existing Web with a layer of machine-interpretable metadata on Web resources so that computer programs can predictably exchange and infer new information. This metadata is usually represented in RDF. Its specification [2] includes a data model and a XML binding. The RDF data model is a collection of triples – subject, predicate and object — that can be viewed as a labeled directed multigraph; a model well suited for knowledge representation. Ontologies formally represent knowledge as a set of concepts within a domain, and the relationships between those concepts. Ontology languages built on top of RDF provide a formal way to encode knowledge about specific domains, including reasoning rules to process that knowledge [4]. In particular, RDF Schema [3] provides a simple ontology language for RDF metadata that can be complemented with the more expressive constructs of OWL [12]. The triplestores can be queried and updated using SPARQL.

2.2. Knowledge bases

Knowledge bases are essentially information repositories that can be categorized as machine or human-readable information repositories. A human-readable knowledge base can be coupled with a machine-readable one, through replication or some real-time and automatic interface. In that case, client programs may use reasoning on computer-readable portion of data to provide, for instance, better search on human-readable texts. A great example is the machine-readable DBpedia extraction from human-readable Wikipedia.

Wikipedia articles consist mostly of free text. However, the joint efforts of human volunteers have recently obtained numerous facts from Wikipedia, storing them as machine-harvestable triplestores in Wikipedia infoboxes [17]. The

DBpedia project extracts this structured information and combines this information into a huge, cross-domain knowledge base. DBpedia uses RDF as the data model for representing extracted information and for publishing it on the Web. Then, SPARQL can be used as the query language to extract information allowing users to query relationships and properties associated with many different Wikipedia resources.

2.3. Semantic similarity

Extracting the semantic relatedness of terms is an important topic in several areas, including data mining, information retrieval and web recommendation. Typically there are two ways to compute semantic relatedness on data:

1. by defining a topological similarity using ontologies to define the distance between words (e.g. in a directed acyclic graph the minimal distance between two term nodes);
2. by using statistical means such as a vector space model to correlate words from a text corpus (co-occurrence).

Semantic similarity measures have been developed and applied in several domain ontologies such as in Computational Linguistics (e.g. Wordnet[3]) or Biomedical Informatics (e.g. Gene Ontology[4]). In order to calculate the topological similarity one can rely either on ontological concepts (edge-based or node-based) or ontological instances (pairwise or groupwise). A well-known node-based metric is the one developed by Resnik [13] which computes the probability of finding the concept (term or word) in a given corpus. It relies on the lowest common subsumer which has the shortest distance from the two concepts compared. This metric is usually applied on WordNet [6] a lexical database that encodes relations between words such as synonymy and hypernymy. A survey [14] between human and machine similarity judgments on a Wordnet taxonomy reveal highest correlation values on other topological metrics such the ones developed by Jiang [9] and Lin [10].

Statistical computation of semantic relatedness relies on algebraic models for representing text documents (and any objects, in general) as vectors of identifiers. Comparing text fragments as bags of words in vector space [1] is the simplest technique, but is restricted to learning from individual word occurrences. The semantic sensitivity is another issue where documents with similar context but different term vocabulary won't be associated, resulting in a ”false negative match”. Latent Semantic Analysis (LSA) [5] is a statistical technique, which leverage word co-occurrence information from a large unlabelled corpus of text [8].

Currently, Wikipedia has been used for information retrieval related tasks [16], [18], [7] and [11]. This is due to the increasing amount of articles available and the associated semantic information (e.g. article and category links).

[3] http://wordnet.princeton.edu/
[4] http://www.geneontology.org/

One of these efforts is the Explicit Semantic Analysis(ESA), a novel method that represents the meaning of texts in a high-dimensional space of concepts derived from Wikipedia and the Open Directory Project (ODP). It uses machine learning techniques to represent the meaning of any text as a weighted vector of Wikipedia-based concepts. The relatedness of texts in this space is obtained by comparing the corresponding vectors using conventional metrics (e.g. cosine) [7].

3. Proximity

This section presents an approach to compute semantic relatedness using ontological information in RDF graphs. The first subsection provides the motivation for using proximity, rather than distance, as the underlying concept for computing semantic relatedness between two nodes. The following subsection presents a formal definition of the proximity based on sets of paths connecting the nodes. The final subsection outlines the algorithm for computing proximity using the proposed definition.

3.1. Motivation

Concepts on DBPedia are represented by nodes. Take for instance the music domain used for the case study presented in section 5. Singers, bands, music genres, instruments or virtually any concept related to music is represented as a node in DBpedia. These nodes are related by properties, such as `has genre` connecting singers to genres, and thus form a graph. This graph can be retrieved in RDF format using the SPARQL endpoint of DBpedia.

The core idea in the research presented in this paper is to use the RDF graph to compute the similarity between nodes. Actually, the goal is the similarity between terms, but each node and arc of this graph has a label — a string representation or stringification — that can be seen as a term.

At first sight relatedness may seem to be the inverse of the distance between nodes. Two nodes far apart are unrelated and every node is totally (infinitely) related to itself. Interpreting relatedness as a function of distance has an obvious advantage: computing distances between nodes in a graph is a well studied problem with several known algorithms. After assigning a weight to each arc one can compute the distance as the minimum length of all the paths connecting the two nodes.

On a closer inspection this interpretation of relatedness as the inverse of distance reveals some problems. Consider the graph in Fig. 1. Depending on the weight assigned to the arcs formed by the properties `has type` and `has genre`, the distances between Lady Gaga, Madonna and Queen are the same. If the `has genre` has less weight than `has type`, this would mean that the band Queen is as related to Lady Gaga as Madonna, which obviously should not be the case. On the other hand, if `has type` has less weight than `has`

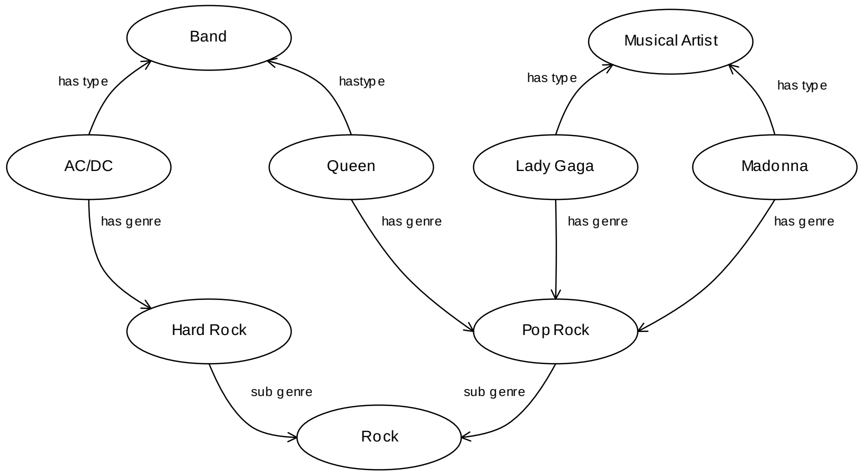


Fig. 1. RDF graph for concepts in music domain

`genre` then Queen is more related to AC/DC than Lady Gaga or Madonna simply because they are both bands, which also should not be the case.

In the proposed approach we consider *proximity* rather than distance as a measure of relatedness among nodes. By definition[5], proximity is closeness; the state of being near as in space, time, or relationship. Rather than focusing solely on minimum path length, proximity balances also the number of existing paths between nodes. As a metaphor consider the proximity between two persons. More than resulting from a single common interest, however strong, it results from a collection of common interests.

With this notion of proximity, Lady Gaga and Madonna are more related to each other than with Queen since they have two different paths connecting each other, one through `Musical Artist` and another `Pop Rock`. By the same token the band Queen is more related to them than to the band AC/DC.

An algorithm to compute proximity must take into account the several paths connecting two nodes and their weights. However, paths are made of several edges, and the weight of an edge should contribute less to proximity as it is further away in the path. In fact, there must be a limit in number of edges in a path, as RDF graphs are usually connected graphs.

3.2. Definition

To be of practical use the notion of proximity among RDF nodes needs to be formalized. Proximity must be a function of graph nodes returning their amount

[5] https://en.wiktionary.org/wiki/proximity

of proximity. Given a graph with a set of nodes V the objective of this subsection is thus to define a function

$$p : V \times V \rightarrow [0, 1]$$

The proximity function p must take two nodes and return the "percentage" of proximity between them. That is, proximity of related nodes must be close to 1, with $\forall_{v \in V}\ p(v, v) = 1$, and the proximity of unrelated nodes must be close to 0.

An RDF graph is actually a typed multigraph, meaning that any pair of nodes can be connected by several edges, known as properties. Nodes and specially edges (properties) in RDF graphs have an URI that can be interpreted as a *type*[6]. For the purpose of defining a proximity function only edge types are relevant. Moreover, this approach requires weights associated with edge types, rather then directly to edges as is usual in graph theory.

Consider a direct[7] typed multigraph $G = (V, E, T, W)$ where V is a set of nodes or vertices, E is a set of edges, T is a set of edge types and W is a mapping of types to positive integers. Each edge in E is an ordered triplet (u, v, t) where $u, v \in V$ and $t \in T$.

The set W defines a mapping $w : T \rightarrow \mathbb{N}^{+}$ and the lower upper bound of weights for all types is

$$\Omega(G) \equiv \max_{t_i \in T}\ w(t_i)$$

The *degree* of a node is the number of edges connecting to it, $deg(u) = \#\{(u', v', t') \in E : u' = u\}$ and the degree of a graph G, denoted $\Delta(G)$, is usually defined as the maximum of the node degrees

$$\Delta(G) = \max_{v \in V}\ deg(v)$$

Given the multigraph G, an *acyclical path* p of size $n \in \mathbb{N}^{+}$ is defined as a sequence of unrepeated nodes $u_0 \ldots u_n \forall_{0 \leq i,j \leq n} u_i \neq u_j$ connected by edges with type t_i in either direction, that is $\forall_i (u_{i-1}, u_i, t_i) \in E \vee (u_i, u_{i-1}, t_i) \in E$, as follows.

$$p = u_0 \xrightarrow{t_1} u_1 \xrightarrow{t_2} u_2 \ldots u_{n-1} \xrightarrow{t_n} u_n$$

An acyclical path must have at least one edge and cannot have loops. In the remainder of this section an acyclical path is simply referred as a *path*.

The weight function defined above can be extended to paths. The weight of path p is the sum of weights of each edge's type, $w(p) = w(t_1) + w(t_2) + \ldots + w(t_n)$. Since $w(t_i) \leq \Omega(G)$ it results that $w(p) \leq \Omega(G)n$, where n is the size of the path.

The set of all paths connecting vertices u and v with exactly $n \geq 1$ edges is defined as follows.

[6] A type in the usual sense of graph theory, not an RDF Schema or OWL type.

[7] An RDF graph is a direct graph. However, edge direction is irrelevant for the purpose of this definition of proximity.

$$P^n_{u,v} = \{u_0 \xrightarrow{t_1} u_1 \ldots u_{n-1} \xrightarrow{t_n} u_n : u = u_o \wedge v = u_n \wedge \forall_{0 \leq i,j \leq n}\, u_i \neq u_j\}$$

The weight of $P^n_{u,v}$ can be computed using the path weight function defined above simply by adding the contribution of each path. The reader should note that $\sum_{p \in P^n_{u,v}} w(p) \leq \Omega(G) n \Delta(G)^n$ since $\forall_{p \in P^n_{u,v}} w(p) \leq \Omega(G) n$ and $\#P^n_{u,v} \leq \Delta(G)^n$.

A proximity function can be defined in terms of these sets of paths. The proximity of a node to itself must be taken as a special case given that $\forall_{n \in \mathbb{N}^+} P^n_{u,u} = \emptyset$. For the general case where the two nodes are different, proximity must take into account each path in $P^n_{u,v}$, for all values of n. However, shorter paths must weight more that longer paths. That is, paths in $P^n_{u,v}$ for smaller values of n must contribute more to proximity than those of larger values of n. Having this in mind the proposed proximity function p is defined as follows.

$$p(u,v) = \begin{cases} 1 & \leftarrow u = v \\ \frac{1}{\Omega(G)} \sum\limits_{n=1}^{\infty} \frac{1}{2^n n \Delta(G)^n} \sum\limits_{p \in P^n_{u,v}} w(p) & \leftarrow u \neq v \end{cases}$$

Since this definition relies on an infinite series one must ensure that it converges. Given that $\sum_{p \in P^n_{u,v}} w(p) \leq \Omega(G) n \Delta(G)^n$, if $u \neq v$, $p(u,v) \leq \sum_{n=1}^{\infty} \frac{1}{2^n} = 1$, and thus the series converges absolutely. It is trivial that the proximity function is non-negative, since all its terms and factors are natural numbers. Hence this also proves that the image of the proposed function is defined within the intended codomain ($[0,1]$).

3.3. Algorithm

The proximity function as defined in the previous subsection requires computing an infinite series. However, since the series defining this function converges absolutely, the first n terms compute the proximity within a known error margin.

The proposed proximity algorithm is formalized in Algorithm 1. It takes a multigraph and two strings, and starts by creating initial sets of paths for each of the given terms. If these sets are equal then proximity is set to its maximum value (1). Otherwise the algorithm computes the sets of paths linking the two nodes with a size under a predefined limit.

To compute the set of paths of size n the algorithm expands half-paths starting on both ends. The set `PathSetA` contains paths starting in the node with label `A` and the set `PathSetB` contains paths ending in the node with label `B`. Paths of size n are those with semi-paths in `PathSetA` and `PathSetB` with a common ending. The contribution of these paths to proximity is computed by summing their weights, using the `PathWeight` function, and divide it by a `denominator` that depends on the value of n. Before proceeding to the next value of n, first the semi-paths from sets `PathSetA` and `PathSetB` are alternately expanded. If both sets were expanded at once only paths with an even number of nodes would be generated.

Algorithm 1: Proximity function

```
Input  : G = (V, E, T, W), A, B
Output: Proximity

$\Omega \leftarrow$ MaxWeight$(T, W)$
$\Delta \leftarrow$ MaxDegree$(V, E)$
PathSetA ← NodeSetWithLabel (A,V,E,T)
PathSetB ← NodeSetWithLabel (B,V,E,T)

Proximity ← $0$
ExpandLeft ← $true$

if PathSetA = PathSetB then
    Proximity = $1$
else
    for N ← $0$ to MaxLevel do
        Denominator ← $2^{N}\Omega N\Delta^{N}$
                                  ▷ Process all paths of size N
        for PathA ∈ PathSetA do
            for PathB ∈ PathSetB do
                if LastNodeInPath(PathA) = LastNodeInPath(PathB) then
                    Weight ← PathWeight(PathA) + PathWeight(PathB)
                    Proximity ← Proximity + Weight/Denominator

                        ▷ Expand paths one level alternately in each side
        if ExpandLeft then
            PathSetA ← ExpandPaths(PathSetA, E)
        else
            PathSetB ← ExpandPaths(PathSetB, E)
        ExpandLeft ← Not(ExpandLeft)
```

Function ExpandPaths(PathSet,Edges)

```
Data: PathSet,Edges
Result: NewPathSet

NewPathSet ← $\emptyset$
for Path ∈ PathSet do
    LastNode ← LastNodeInPath(Path)
    for NextNode ∈ $\{$n : (LastNode, n) ∈ Edges ∨ (n, LastNode) ∈ Edges$\}$ do
        if NextNode ∉ Path then
            NewPathSet ∋ (Path, NextNode)
```

The function `ExpandPaths` expands each path in the given set of paths using a set of edges. Paths are expanded at their end with nodes for which there is an edge starting (or ending) at their last node. If the new node already occurs in the selected path then this is not a valid expansion as it would contain a cycle and it is not added to the expanded path set. Note that this function expands the size of the paths rather than the cardinality of the set. The expanded path set contains paths of size $n+1$ where n is the size of the paths in the original path set. The cardinality of the expanded path set may either increase, decrease, or remain unchanged by this expansion.

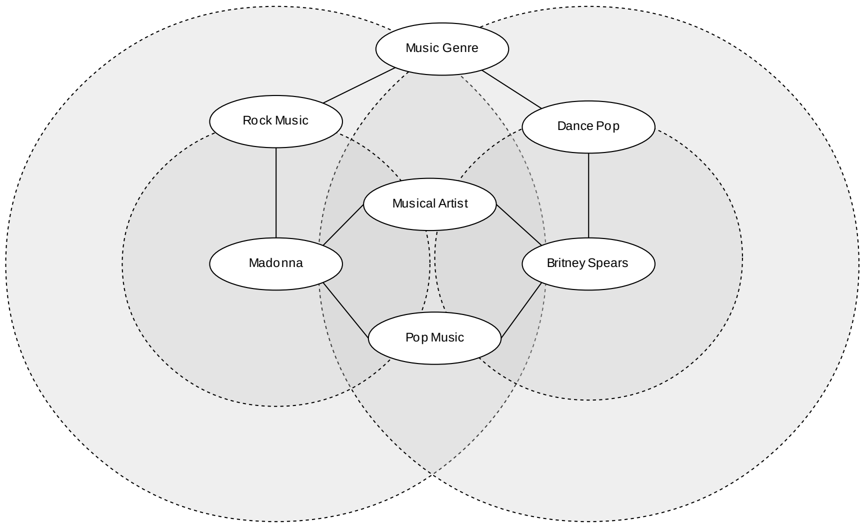


Fig. 2. Using the proximity algorithm to relate "Madonna" and "Britney Spears"

Figure 2 shows how the proximity algorithm proceeds to relate the nodes "Madonna" and "Britney Spears". This example omits the label nodes and starts with concept nodes associated with the relevant terms. We can see that each node is at the center of a pair of concentric circles. Each circle intersects a set of nodes that are reached from the center with a certain number of path segments. For instance, "Rock Music", "Musical Artist" and "Pop Music" are all a path segment away from "Madonna". A similar situation occurs with "Britney Spears" and some nodes are common to both circles, in this case "Musical Artist" and "Pop Music". These two intermediary nodes contribute with two independent paths connecting the original modes. The remaining nodes, "Rock music" for "Madonna" and "Dance Pop" for "Britney Spears" are used to continue unfolding the sets of nearby nodes connected to the original ones. In this case the node “Music genre” is common to both circles on the second level. This path is longer than the previous ones (i.e. has more path segments) and thus contributes less to proximity. At each level the contribution of new paths diminishes, although they are usually in greater number. After a few levels (typically 5) the algorithm stops.

The proximity algorithm can be extended to compare groups of concepts. This is relevant to relate two web pages, for instance. For this purpose a web page is represented by a bag-of-words, where each word occurs in the web page and is also a label of a graph node. The proximity between the two bags-of-words can be defined as the average, or the maximum, of all proximity pairs.

4. Shakti

The algorithm described in the previous section is implemented by a system called Shakti. This system is responsible for extracting data relevant to a given domain from DBpedia, and to provide a measure of the proximity among concepts in that domain. This system is implemented in Java using an open-source semantic web toolbox called Jena[8] including application interfaces for RDF and OWL, a SPARQL engine, as well as parsers and serializers for RDF in several formats such as XML, N3 and N-Triples.

The overall architecture of a Shakti use case is described in the diagram in Figure 3. It shows that Shakti mediates between a client system and DBpedia, that in turn harvests its data from the Wikipedia. The system itself is composed of three main components:

controller is parametrized by a configuration file defining a domain and provides control over the other components;
extractor fetches data related to a domain from the DBpedia, pre-processes it and stores the graph in a local database;
proximity uses local graph to compute the proximity among terms in a pre-configured domain.

The purpose of the controller is twofold: to manage the processes of extracting data and computing proximity values by proving configurations to the modules; and to abstract the domain required by client applications. For instance, to use Shakti in a music domain it is necessary to identify the relevant classes on concepts, such as *musical artist*, *genre* or *instrument*, as well as the properties that connect them, such as *type*, *has genre* or *plays instrument*. To use Shakti in a different domain, say movies, it is necessary to reconfigure it.

The controller is parametrized by an XML configuration file formally defined by an XML Schema definition as depicted in Figure 4. The top level attributes in this definition configure general parameters, as the URL of the SPARQL endpoint, the natural languages of the labels (e.g. English, Portuguese), the maximum level used in the proximity algorithm, among others. The top level elements are used for defining prefixes, types and properties. XML prefixes are

[8] https://jena.apache.org/

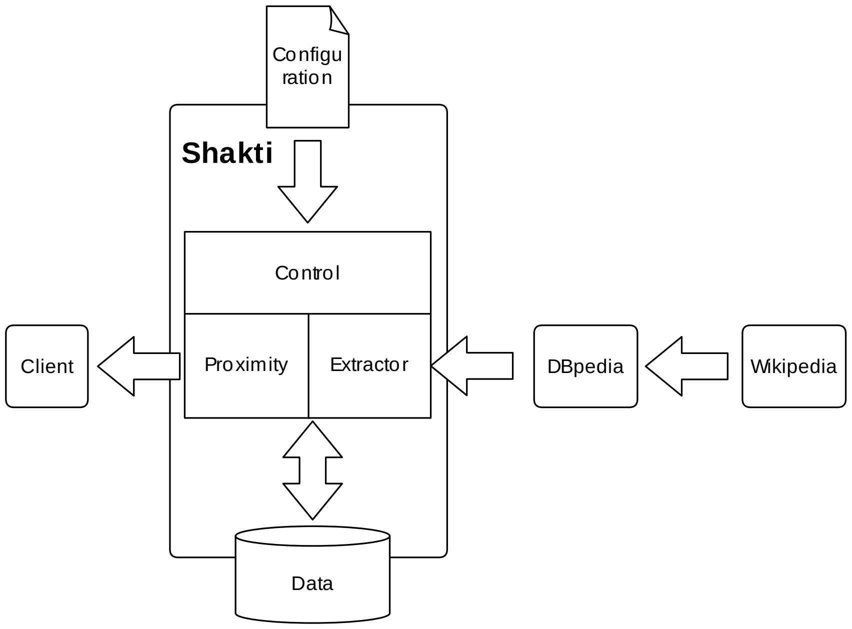


Fig. 3. The architecture of Shakti

routinely used in RDF to shorten the URLs used to identify nodes. This configuration enables the declaration of prefixes used in SPARQL queries. The configuration file also enumerates the types (classes) of concepts required by a domain. This ensures that all the concepts with a declared type, having a label in the requested language are downloaded from DBpedia. The declaration of properties has a similar role but it also provides the weights assigned to path segments required by the algorithm. Each property definition includes a set of occurrences since the same name may be used to connect different types. That is, each property occurrence has a domain (source) and a range (target) and these must be one of the previously defined types. These definitions ensure that only the relevant occurrences of a property are effectively fetched from DBpedia.

The *extractor* retrieves data using the SPARQL endpoint of DBpedia. The extractor processes the configuration data provided by the controller and produces SPARQL queries that fetch a DBpedia sub-graph relevant for a given domain. Listing 1.1 shows an example of a SPARQL query to extract a type declared in the configuration file, where the string “[TYPE]” is replaced by each declared type. Similar queries are used for extracting properties.

Part of the data extracted this way, namely the labels, must be preprocessed. Firstly, multiword labels are annotated incorrectly with language tags and must

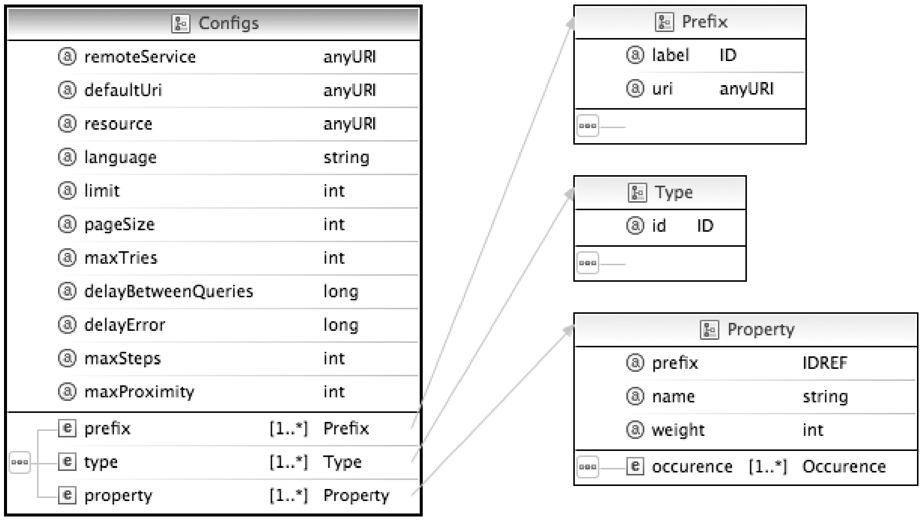


Fig. 4. The XML Schema definition of Shakti configuration

Listing 1.1. SPARQL query for extracting a type

```
SELECT    ?R ?L
WHERE {
        ?R      rdf:type    dbpedia:[TYPE];
                rdfs:label ?L.
}
```

be fixed. For instance, a label such as ``` ``Lady Gaga@en'' ``` must be converted into ``` ``Lady Gaga''@en ```. Secondly all characters between parentheses must be removed. The Wikipedia, and consequently DBpedia, use parentheses to disambiguate concepts when needed. For instance, ``` ``Queen (Band)''@en ``` is a different concept from ``` ``Queen''@en ``` but in a music setting the term in brackets is not only irrelevant but would disable the identification with the term ``` ``Queen'' ``` when referring to the actual band. Also, concepts with short labels (less than 3 characters) or solely with digits (e.g. “23”) are simply discarded.

The *proximity* module is responsible for computing the relatedness between two terms, or two bags-of-terms, from the graph extracted from DBpedia and already preprocessed. This module maintains a dictionary with all labels in the graph, implemented using a prefix tree, or *trie*. This data structure enables an efficient screening of terms, discarding those for which relatedness cannot be computed. Following this step, the implementation follows Algorithm 1.

5. Evaluation

This section presents a use of Skati in the implementation of a recommender developed as part of the project Palco 3.0. This project was targeted to the re-development of an existing Portuguese social network — Palco Principal — whose main subject is alternative music.

The goals of this project include the automatic identification, classification and recommendation of site content. The recommendation service developed for this project is structured around recommenders — pluggable components that generate a recommendation for a certain request based on a given model. Most of the recommenders developed for this service use collaborative filtering. For instance, a typical recommender suggest songs to users in Palco Principal based on the recorded activity of other users. If a user shares a large set of songs in his or her playlist with other users then it is likely that he or she will enjoy other songs in their playlist.

This approach is very effective and widely used but its main issue is cold start. If the system has no previous record of a new user then it will not be able to produce a recommendation. An alternative is to produce a content-based recommender. To implement such a recommender Shakti was used to find related content on the web site. This recommender can be used on words extracted from the web page itself, such as news articles or interviews, or on tags used to classify web pages, such as musics, photos of videos.

The remainder of this section describes the main steps to define a content recommender for Palco Principal using Shakti and how this experiment was used to evaluate this approach.

5.1. Proximity based recommendation

Palco Principal is a music website hence this is the domain that must be used in Shakti. This required selecting DBpedia classes and properties relevant to this domain, preparing DBpedia for extracting data from the Portuguese Wikipedia to populate these classes, and configuring Shakti with the relevant types and properties to compute proximity values.

DBpedia already has an extensive ontology covering most of the knowledge present in Wikipedia. This is certainly the case with the music domain and all the necessary classes and properties were already available. The DBpedia uses a collection of mappings to extract data present in the info boxes of Wikipedia. Unfortunately these mappings were only available for the English pages of Wikipedia and they had to be adapted for the pages in Portuguese. The DBpedia uses a wiki to maintain these mappings and new mappings of some classes had to be associated with the language label ”pt”.

In the Shakti it was necessary to configure the XML file to extract the selected classes and properties from DBpedia. These classes, whose mappings were created on DBpedia wiki for Portuguese pages, are:

MusicalArtist solo performers (e.g. Madonna, Sting);

Band groups of musicians performing as a band (e.g. Queen, Bon Jovi);
MusicGenre musical genres (e.g. rock, pop).

The properties associated with these classes that were considered relevant were also inserted in the configuration file and are enumerated in Table 1. This table defines also the weights assigned to properties, with values ranging from 1 to 10, needed for computing proximity values. These weights were assigned based on the subjective perception of the authors on the proximity of different bands and artists. A sounder approach to weight calibration was left for future work.

Table 1. Properties of a music domain

Property	**Domain**	**Range**	**Weight**
Genre	Band and MusicalArtist	MusicGenre	7
Instrument	Band and MusicalArtist	*label*	2
StylisticInfluences	MusicGenre	*label*	4
AssociatedBand	Band	Band	10
AssociatedMusicaArtist	MusicalArtist	MusicalArtist	10
CurrentMember	Band	*label*	5
PastMember	Band	*label*	5

To integrate Shakti with the recommender it was necessary to implement a client application. This application is responsible for populating a table with proximity values among web pages recorded on the recommender service database. For each page this client application extracts a bag-of-words, either the words on the actual page or its tags. For each pair of bags-of-words it computes a proximity using methods provided by Shakti.

5.2. Results analysis

Shakti is currently being used in an experimental recommender. Thus, the recommendations are not yet available on the site the of Palco Principal. For this reason a comprehensive analysis is not yet possible. This subsection presents some experimental results that are possible to obtain from the current setting.

For this experiment the recommender system computed proximity values among news and events pages, which took about a day. In total 57,982 proximity relations among news pages were calculated, plus 59992 among event pages, performing a grand total of 69604805 relations.

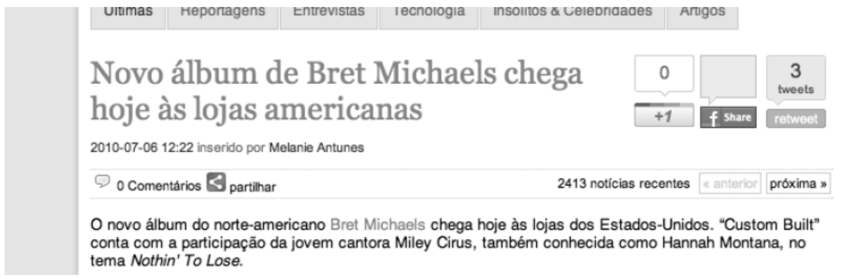
Table 2 displays the proximity table for news pages ordered by decreasing proximity. Each id code is a news item in the web site. For this particular entity the recommender searched for content regarding both terms from its text and tags.

To analyze the performance of Shakti the contents of the 2 most related pages — id 3540 (resource A) and id 2623 (resource B) — were compared in

Table 2. Proximity between pairs of news pages.

Resource ID	Resource ID	Proximity
3540	2623	0.22
3540	2431	0.21
3540	3000	0.15
3540	4115	0.15
3540	2691	0.15
3540	1892	0.15
3540	2676	0.14
3540	760	0.14
3540	3189	0.14
3540	4397	0.14

detail. The text and tags of this resource can be viewed in Figure 5. In order to calculate proximity values, Shakti merge both fields and generates a group of concepts present in the RDF graph. Thus, from all the words of *text* and *tags* fields only the following bag-of-words are actually used to compute proximity: 38 Special, Lynyrd Skynyrd, Bret Michaels. For resource B the bag-of-words considered for computing compute proximity is: Lemmy, Myles Kennedy, Andrew Stockdale, Dave Grohl, Fergie, Ian Astbury, Kid Rock, M. Shadows, Rock, The Sword, Adam Levine, Ozzy Osbourne, Chris Cornell, Duff McKagan, Slash, Iggy Pop. Using these two bags-of-words Shakti computes a proximity of 0.22. The concepts are names of the bands appearing in news text, so the approach of using the this field to determine proximity seems promising.



Novo álbum de Bret Michaels chega hoje às lojas americanas

2010-07-06 12:22 inserido por Melanie Antunes

2413 notícias recentes

O novo álbum do norte-americano Bret Michaels chega hoje às lojas dos Estados-Unidos. “Custom Built” conta com a participação da jovem cantora Miley Cirus, também conhecida como Hannah Montana, no tema *Nothin' To Lose*.

Fig. 5. News piece generated from resource A.

Analyzing these news items one notices that they are on two musician artists with a musical genre in common, and both playing the guitar. This shows that the two news items are in fact related and a 0.22 proximity seems a reasonable figure. Note that proximity values range between 0.0 (unrelated) to 1.0 (the same).

The proximity values computed for all pages vary between 0.1 and 0.22 and the average value of is 0.2. This value is lower than expected. Of course that these figures can be modified simply by reconfiguring the property weights. On the other hand, Shakti determined a non null proximity in 24,401 of a total of 33,616,804 possible relationships, about 0.07%, which is an unsatisfactory figure for a recommendation system.

One of the culprits for these poor results is the text encoding using HTML entities in the database of Palco Principal. For instance, the term ”Guns N’ Roses” (which is part of the text and tags of resource B) is written in the database in the format ”Guns N&amp;#039 Roses”. This value is sent to Shakti. As Shakti is not prepared to receive this type of formatting, it does not detect the word in the dictionary.

This experiment suggest that algorithm is producing the expected results. For the pairs of pages that produced a non null proximity the obtained measure is consistent with their degree of relatedness. However, the number of pages that the algorithm was able to relate is insufficient for a recommender system. The problems with text encoding alone do not justify the low number recommendations obtained in this experiment. Most probably the words contained in those pages are not labels in the sub-ontology extracted from DBpedia and it does not not cover satisfactory the domain of Palco Principal. It should be noted that this sub-ontology deals only with artists and bands that have sufficient recognition to have their own entry in Wikipedia. Other concepts related to music, such as musical instruments or music event venues, were not covered. Thus, pages on music festivals featuring garage bands, for instance, or advertising used guitars for sale, would be difficult to relate. In any event, further experimentation is needed to validate both the algorithm itself and the approach of using semantic relatedness a basis for recommendation.

6. Conclusions and future work

The goal of the research described in this paper is to measure the relatedness of two terms using the knowledge base of BDPedia. The motivation for this research is to use semantic relatedness in content-based recommenders, in particular in tags provided by users in social networks.

This paper proposes proximity, rather than distance, as a means to compute semantic relatedness on RDF nodes. It provides a formal definition of the proximity in terms of the sets of paths connecting the nodes, and an algorithm to determine these sets and compute proximity. The algorithm ponders the collection on paths connecting the two terms using the weights associated to properties on the ontological graph. This algorithm was implemented in a system called Shakti. This system fetches a sub-graph of the ontology in DBpedia relevant to a certain domain and computes the relatedness of terms assigned as labels to concepts. To validate the proposed approach Shakti was used to populate a proximity table on a web recommender service of Palco Principal, a Portuguese social network whose subject is alternative music. The results are promising, although the ontology extracted from DBpedia is not yet covering satisfactory the terms contained on the pages of Palco Principal.

Part of the future work in this research includes the experimentation with larger ontologies, providing better coverage of the underlying domain and validating scalability of Shakti. At this stage most of the effort of using Shakti is configuring this tool. We plan the development of a graphical user interface for assisting the tool users in defining the classes and properties to extract from DBpedia. There are two approaches being considered for this task. On the first approach a seed class is typed in and other related classes and properties in that domain are suggested for possible inclusion. On the second approach Shakti is fed with a collection of example terms and DBpedia is searched for related classes and properties. Independently from the selected approach, the graphical user interface will also assist in the definition of property weights and other general configurations required by Shakti.

The validation of the algorithm itself is perhaps the most important part of the future work. It is necessary to compare it experimentally with the results obtained by similar algorithms using standard benchmarks. A testbed for computing similarity and visualizing relatedness among any sets of terms, based on the full DPpedia ontology, is currently being developed. This testbed is expected to be instrumental in the validation of the proposed algorithm.

The fact that the algorithm currently relies on weights being assigned to properties is an obstacle to use it with multiple domains. This issue can be overcome by assigning weights to properties according to their role on the ontology, independently of the domain: `is-a` properties with the maximum weight, `part-of` properties with an intermediary weight, and all other properties with a minimum weight. The testbed will be used to fine-tune these generic weights and to validate this approach to weight assignment.

Acknowledgments.

This work is in part funded by the ERDF/COMPETE Programme and by FCT within project FCOMP-01-0124-FEDER-022701. The author wishes to thank to Vânia Rodrigues for her collaboration on the implementation o Shakti, and to Ricardo Queirós, João Delgado and the anonymous reviewers for their helpful comments.

References

1. Baeza-Yates, R.A., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
2. Beckett, D., McBride, B.: Resource description framework (RDF) model and syntax specification (revised). Tech. rep., W3C (2004), http://www.w3.org/TR/REC-rdf-syntax/
3. Brickley, D., Guha, R.: RDF vocabulary description language 1.0: RDF schema. Tech. rep., W3C (2004), http://www.w3.org/TR/rdf-schema/
4. Corcho, O., Gómez-Pérez, A.: A roadmap to ontology specification languages. In: Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management. pp. 80–96. EKAW '00, Springer-Verlag, London, UK, UK (2000), http://dl.acm.org/citation.cfm?id=645361.650838

5. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science 41, 391–407 (1990)
6. Fellbaum, C. (ed.): WordNet An Electronic Lexical Database. The MIT Press, Cambridge, MA ; London (May 1998), http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=8106
7. Gabrilovich, E.: Feature generation for textual information retrieval using world knowledge. SIGIR Forum 41(2), 123–123 (Dec 2007), http://doi.acm.org/10.1145/1328964.1328988
8. Gabrilovich, E., Markovitch, S.: Overcoming the brittleness bottleneck using wikipedia: enhancing text categorization with encyclopedic knowledge. In: proceedings of the 21st national conference on Artificial intelligence - Volume 2. pp. 1301–1306. AAAI'06, AAAI Press (2006), http://dl.acm.org/citation.cfm?id=1597348.1597395
9. Jiang, J., Conrath, D.: Semantic similarity based on corpus statistics and lexical taxonomy. In: Proc. of the Int'l. Conf. on Research in Computational Linguistics. pp. 19–33 (1997), http://www.cse.iitb.ac.in/ cs626-449/Papers/WordSimilarity/4.pdf
10. Lin, D.: An information-theoretic definition of similarity. In: Proceedings of the Fifteenth International Conference on Machine Learning. pp. 296–304. ICML '98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998), http://dl.acm.org/citation.cfm?id=645527.657297
11. Milne, D., Witten, I.H.: Learning to link with wikipedia. In: Proceedings of the 17th ACM conference on Information and knowledge management. pp. 509–518. CIKM '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1458082.1458150
12. Nilsson, M., Palmer, M., Brase, J.: The LOM RDF binding - principles and implementation. In: 3rd Annual ARIADNE Conference (2003)
13. Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. In: Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1. pp. 448–453. IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995), http://dl.acm.org/citation.cfm?id=1625855.1625914
14. Seco, N., Veale, T., Hayes, J.: An intrinsic information content metric for semantic similarity in WordNet. Proc. of ECAI 4, 1089–1090 (2004)
15. Smirnov, I.: Overview of stemming algorithms. Mechanical Translation (2008)
16. Strube, M., Ponzetto, S.P.: Wikirelate! computing semantic relatedness using wikipedia. In: proceedings of the 21st national conference on Artificial intelligence - Volume 2. pp. 1419–1424. AAAI'06, AAAI Press (2006), http://dl.acm.org/citation.cfm?id=1597348.1597414
17. Wu, F., Weld, D.S.: Automatically refining the wikipedia infobox ontology. In: Proceedings of the 17th international conference on World Wide Web. pp. 635–644. WWW '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1367497.1367583
18. Zesch, T., Gurevych, I.: Automatically creating datasets for measures of semantic relatedness. In: Proceedings of the Workshop on Linguistic Distances. pp. 16–24. LD '06, Association for Computational Linguistics, Stroudsburg, PA, USA (2006), http://dl.acm.org/citation.cfm?id=1641976.1641980

José Paulo Leal is assistant professor at the department of Computer Science of the Faculty of Sciences of the University of Porto (FCUP) and associate researcher of the Center for Research in Advanced Computing Systems

(CRACS). His main research interests are eLearning system implementation, structured document processing and software engineering. He has a special interest on automatic exercise evaluation, in particular on the evaluation of programming exercises, on ontology processing and on web adaptability. He has participated in several research projects in his main research areas, including technology transfer projects with industrial partners. He has over 60 publications in conference proceedings, journals and book chapters.

Received: November 30, 2012; Accepted: May 15, 2013.

DOI: 10.2298/CSIS121130061M

Managing experiments on cognitive processes in writing with HandSpy

Carlos Monteiro[1] and José Paulo Leal[2]

[1] CRACS & INESC-Porto LA
Faculty of Sciences, University of Porto
Porto, Portugal
carlosmonteiro@dcc.fc.up.pt
[2] CRACS & INESC-Porto LA
Faculty of Sciences, University of Porto
Porto, Portugal
zp@dcc.fc.up.pt

Abstract. Experiments on cognitive processes require a detailed analysis of the contribution of many participants. In the case of cognitive processes in writing, these experiments require special software tools to collect gestures performed with a pen or a stylus, and recorded with special hardware. These tools produce different kinds of data files in binary and proprietary formats that need to be managed on a workstation file system for further processing with generic tools, such as spreadsheets and statistical analysis software. The lack of common formats and open repositories hinders the possibility of distributing the workload among researchers within the research group, of re-processing the collected data with software developed by other research groups, and of sharing results with the rest of the cognitive processes research community.
This paper describes the development of HandSpy, a collaborative environment for managing experiments in the cognitive processes in writing. This environment was designed to cover all the stages of the experiment, from the definition of tasks to be performed by participants, to the synthesis of results. Collaboration in HandSpy is enabled by a rich web interface. To decouple the environment from existing hardware devices for collecting written production, namely digitizing tablets and smart pens, HandSpy is based on the InkML standard, an XML data format for representing digital ink. This design choice shaped many of the features in HandSpy, such as the use of an XML database for managing application data and the use of XML transformations. XML transformations convert between persistent data representations used for storage and transient data representations required by the widgets on the user interface. Despite being a system independent from a specific collecting device, for the system validation, a framework for data collection was created. This framework has also been highlighted in the paper due to the important role it took in a data collection process, of a scientific project to study the cognitive processes involved in writing.

Keywords: InkML, experiments management, collaborative environment, XML data processing.

1. Introduction

Writing is a basic tool for a successful personal and academic growth. Given the importance of this subject social scientists are actively researching the cognitive processes involved in writing. Writing studies can focus on different writing forms, such as keyboard logging and handwriting. The writting action can be complemented with other indicators, such as eye movements and speech made during the production. The collected data focus on the complementary concepts of burst and pause [7]. A *burst* is a time span in which text was produced without interruptions. A *pause* is a non-writing time span between two writing bursts. These two moments are linked to distinct cognitive processes. The duration of a pause is related to the writing task being performed. During the *pause* period the working memory used in the writing process is freed. Therefore the time spent pausing is used for planning and revising the written production.

The development of HandSpy is embedded in the research project DAAR, being held at the Psychology Faculty of Porto University. The object of study in this research is the relation between the cognitive processes involved in writing and the quality of the writing productions. As the goal of this research is in general to determine the factors that influence the development of writing skills, the participants are school children. The object of these research studies are writing productions on different tasks such as narratives, copies, dictations and alphabet transcriptions. Different tasks influence the way the idea of the text is processed. The study results may be used to detect learning problems related to the ability of creating good quality writing productions. This can then be used to define new strategies and interventions on writing teaching.

The development of HandSpy was inspired on the existing, state of the art, software for collecting and analysing written productions. Although these tools offer a good handwritten analysis system they lack of a simple way to collect and organize the data. With the desire to innovate and improve the way the writing studies are processed, HandSpy was designed to be a web based system offering typical features of writing research tools. The system covers the entire experimental process filling the existing gap on the experiment management. By using a common repository, researchers can set up an experiment for storing all the entities involved. An entity is an abstract concept to define the different components of project such as tasks definitions, trait information on participants involved in the experiment and the generated data. Using a web server for data storing, the system follows a collaborative paradigm where various researchers can work on the same experiment simultaneously. HandSpy uses a standard XML format for data files which enables users to collect data from various hardware devices. XML files enable the data persistence over time. Being based on a web system avoids the complexity of installation processes, as one installation can be shared by several users. The collected data files need to be uploaded just once, and thereafter are accessible to all researches, even to those not involved in the collection process.

A collecting framework was developed to collect data for system validation. This framework uses a fairly inexpensive hardware device in a shape of a normal pen, enabling a less intrusive collecting method in the writing process.

The present paper is organized as follows. Section 2 describes the state of the art with regard to the platforms used to conduct scientific experiments on writing productions and describes the principles of experiment management systems in which HandSpy was based. Section 3 describes the main technologies used to develop HandSpy and describes existing devices to record handwriting productions. Section 4 is the main description of the design and implementation methods. Section 5 describes a collecting framework created to validate HandSpy usage. Section 6 is an evaluation of the usability of HandSpy to prepare future modifications. Section 7 concludes this paper and identifies opportunities for future work on HandSpy.

2. Related Work

This section covers background topics related to the development of a collaborative environment for managing experiments on cognitive processes in writing. Studying cognitive processes in writing involves the detailed analysis of written productions, therefore the analysis component is the essential feature on a software for that purpose. HandSpy analysis engine owes credit to two mature systems used in the study of cognitive processes in writing. These systems are described in the first section of this paper.
The proposed environment complies with the requisites for an Experiment Management System (EMS) [3] thus the second section is devoted to introduce this kind of system.

2.1. Collecting and Analysing Tools

Studies on cognitive processes in writing are mainly conducted by social scientists. In the last decade this subject was supported by the availability of new devices to digitally record the writing productions and complemented with new software to analyze those productions. The two most proeminent tools currently available to conduct studies on cognitive processes in handwriting are Eye And Pen [1] and Ductus [2]. The following subsections are a description of these systems.

Eye And Pen The Eye and Pen system was originally design to study reading while performing a writing production. The system is composed by three parts. A collecting system composed by a digitizing tablet and a eye tracker, a software for data analysis and experiment control system. The digitizing tablet recordings and the eye tracker signals are synced in the begining of the experiment. The digitizing tablet records the position of the pen and the pressure made in every point throughout collection. This data is used by the Eye and

Pen analysis software to reconstruct the written text and display the point of regard on the tablet surface related to the pen position on that specific moment. The point of regard is the spot in the paper that the participant is looking at a particular moment. The text reconstruction can be played and controlled with a media player style control set. The *pauses* are displayed on the reconstructed text image and are represented by a circle centered on the place where the pen stopped, its diameter is defined by the *pause* duration time.
The experiment control system consists on a scripting language used to define the tasks to be performed. The tasks are displayed on the computer connected to the tablet. Particular regions on the tablet can be assigned to a function. When these regions are reached they act as control buttons of the experiment and sets the end of a task.

Before Eye and Pen, the eye tracking devices were mostly used to study reading processes. The first known use of eye trackers for studying writing was made on computer typing. Studies on computer typed tasks are limited by the expertise of the participants using a computer keyboard. The tasks that can be performed with a computer keyboard are also limited to the typing action.
Any study that makes use of technological equipment to collect the data is subject to errors and mishandling of the devices. For instance the eye tracker depends on specifications of the manufacture and some eye trackers require the participant to hold the head still in order to work properly. Using an eye tracker while writing may distract and alter the normal text production. These factors may invalidate the text production.

Ductus Ductus is a software to study the processes involved on handwritten productions. The system is composed by two modules, a *Stimulus Presentation Module* and a *Data Analysis Module*. The *Stimulus Presentation Module* encompasses two parts, the stimulus presentation and a data acquisition module. The stimulus is displayed on a computer screen in front of the participant and consists on a series of images, words or texts for transcription. The stimulus module supports plain text (.txt) and bitmap (.bmp) file formats. The visual stimulus are preceded by a sound to signal the begining of a stimulus.
The data acquisition module works with any model of digitizer from Wacom, a recognized tablets manufacturer. The sample rate is limited by the digitizer model. The elements recorded by the acquisition module are:

– pen postion - the position of the pen on the digitizer.
– pressure - the pressure made on the digitizer, some digitizers enable the recording of hovering movements on the tablet.
– latency - the time between the apearence of the stimulus and the pen touching the digitizer.
– event landmark - is an event defined by the experimenter to signal some occurence during the recording.

The recorded data is stored in a plain text file and is used by the *Data Analysis Module* to produce the calculations on kinematic and geometrical parameters of the handwriting. The *Data Analysis Module* interface displays information on several windows. There are two windows that display information on the writing and are the most used during the data analysis.

The first window is divided in four parts. A list with data on the points that constitute the text, such as time, position, absolute velocity, absolute acceleration and pressure, an image with the reconstruction of the text, hovering movements are displayed in a gray light tone, a graph with the variations on the trajectory made with the pen and finally a graph with the pressure made on the tablet and an image of the text and the graphs have a vertical line that syncs the position in the text with the positions in the trajectory and pressure graphs.

The second window is used to segment the text for a thorough analysis. The text can be segmented in a hierarchical way, the text can be divided into paragraphs and each paragraph into words and the words into letters. These segments are made to limit the calculations to a precise area. The results are presented in a table and can be exported through the clipboard or can be saved in a plain text file.

2.2. Experiment Management Systems

The growth of data collected during scientific experiments, leveraged by the use of digital devices, created the need for systems to manage this data. Multiple fields of scientific research require the analysis of large amounts of data. Usually, researchers in these areas do not have the necessary knowledge to manage this information in an automated basis by using a digital database system.
An Experiment Management System is composed by two parts, user interface and data storage system. These systems aim to abstract experimentation procedures, offering a consistent data management system replicated by different experiment stages and entities. An entity is a flexible abstract format to represent information regarding some aspect of the experiment, for instance the list of participants, tasks involved in the experiment and actual collected data [3].

In the Figure 1 is depicts the life cycle of an experiment with data abstraction on entities and its relation with data transfers. The stages of an experiment process depicted on the Figure 1 are described on the following.

Experiment Design is the first stage of the experimentation process. In this stage, data files containing information about the experiment are defined. These definitions can be updated in the course of the experiment. These definitions are stored in the database alongside with the collected data.

Data Collection stage can be repeated several times during the experiment, if there is a need to collect more data. The need to collect or recollect data, may arise due to the invalidity of the data or the failure to produce conclusive results.

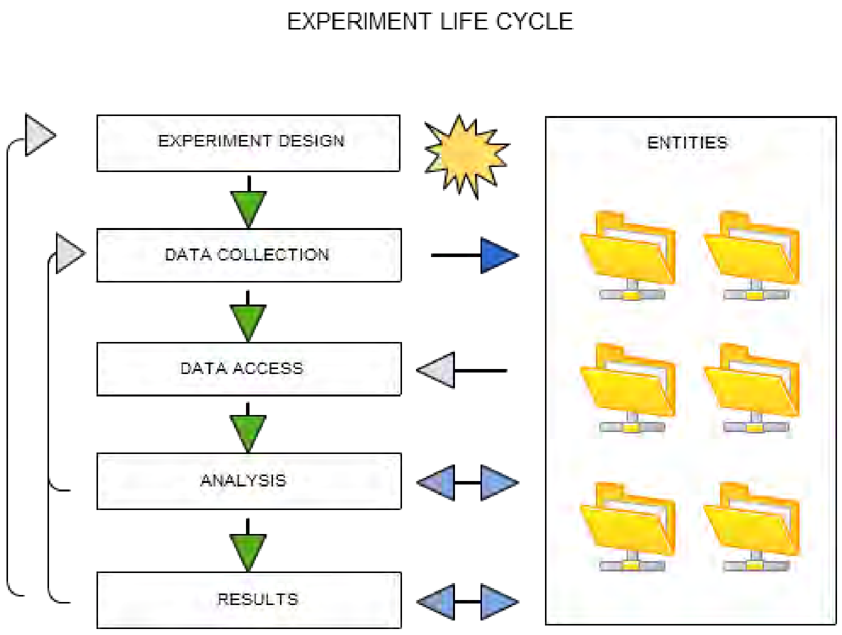


Fig. 1. Experiment Life Cycle

Data Access stage is set to retrieve the data, for analysis, validation or sharing.

Analysis is the main stage of every experiment. The data analysis is done by researchers that assess its validity and generate the results of their studies. In this step design modifications and the need to collect more data may arise.

Results is the final stage of the experiment cycle where the results are generated. The experiment success is validated by the results. At this point the experiment can be terminated, redesigned or more data may be collected.

3. Technology

3.1. InkML

The recent trend of sketching and writing on digital devices capable of recording hand gestures created the need for a standard to describe this kind of data. InkML is a W3C recommendation for storing and exchanging what is commonly called *digital ink*. It is an XML data format to describe a set of strokes digitally representing handwriting and other ink input gestures. It was design to describe ink-based formats but it is flexible enough to store digital interactions such as

keyboard logs and mouse movements.
The ink in InkML is defined by characteristics associated with the act of creating a trace such as the width and color of the trace, the pen orientation while writing, the pen distance to the surface(whether the trace was made with the pen down or hovering the writing surface), among others.

The root element of InkML is `ink` and has the identifier attribute `documentID` with the type *Uniform Resource Identifier (URI)* that uniquely defines each file. The *trace* element is set to describe a continuous trace, i.e. the act of sketching a trace with the pen down on the surface. Each *trace* is a collection of points and their features, separated by commas. These characteristics are defined in the `channel` element. If no `channel` is defined to cast traces, the default `trace` is simply the X and Y coordinates of each point. A set of traces can be grouped in a context, defining optional features such as starting time, writing surface dimensions and characteristics of the trace.

The Listing 1.1 is an example of the ”Hello” word described in a basic InkML file. The word has five letters represented with five `trace` elements, its contents are defined on the `traceFormat` with a set of `channel` elements whose attributes define the `name` and the `type`. The values on the `trace` element separated by commas represent the coordinates (X,Y) and a timestamp defined on the `traceFormat` `channel` elements. This data represents each point depicted on the Figure 2 and the time the point was recorded.

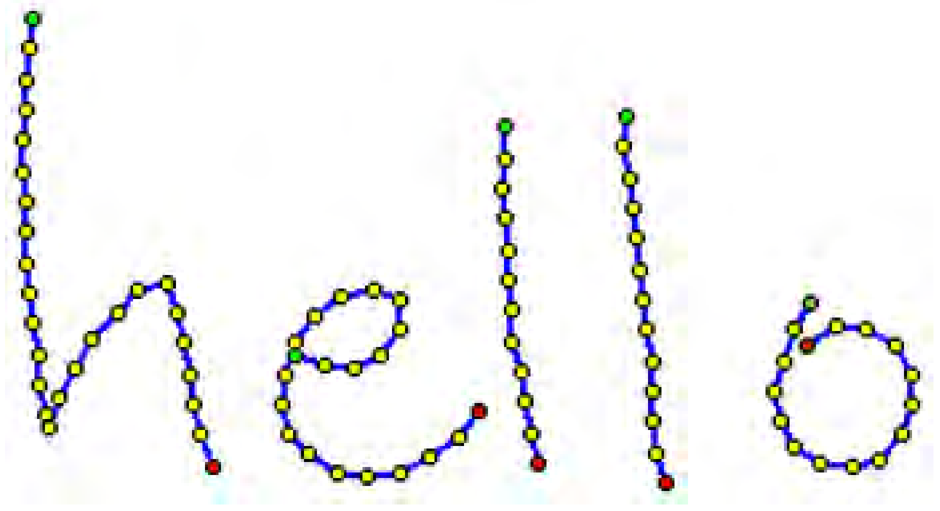


Fig. 2. InkML Hello Example

Listing 1.1. InkML ”hello” example

```
<ink xmlns=”http://www.w3.org/2003/InkML”>
 <context xml:id=”start”>
  <inkSource>
        <traceFormat>
                <channel name=”X” type=”decimal”/>
                <channel name=”Y” type=”decimal”/>
                <channel name=”T” type=”decimal”/>
        </traceFormat>
  </inkSource>
  <timestamp xml:id=”startTime” time=”10000”/>
 </context>
 <trace>
   10   0 11000,  9  14 11200,  8  28 11400,  7  42 11500,  6
56 11600,
   10  70 11700,  8  84 11900,  8  98 12100,  8 112 12200,
9 126 12300,
   10 140 12400, 13 154 12500, 14 168 12600, 17 182 12800, 18 188 12900,
   23 174 13000, 30 160 13100, 38 147 13200, 49 135 13400, 58 124 13600,
   72 121 13700, 77 135 13800, 80 149 13900, 82 163 14000, 84 177 14200,
   87 191 14300, 93 205 14400
 </trace>
 <trace>
   130 155 14500, 144 159 14600, 158 160 14800, 170 154 15000, 179 143 15100,
   179 129 15200, 166 125 15300, 152 128 15400, 140 136 15600, 131 149 15700,
   126 163 15800, 124 177 15900, 128 190 16000, 137 200 16200, 150 208 16300,
   163 210 16400, 178 208 16600, 192 201 16700, 205 192 16900, 214 180 17000
 </trace>
 <trace>
   227  50 17100, 226  64 17200, 225  78 17400, 227 192 17500, 228 106 17600,
   228 120 17800, 229 134 17900, 230 148 18100, 234 162 18200, 235 176 18300
 </trace>
 <trace>
   282 145 18600, 281 159 18700, 284 173 18900, 285 187 19000, 287 101 19100,
   288 115 19200, 290 129 19400, 291 143 19700, 294 157 19900, 294 171 20000,
   294 185 20200, 296 199 20300
 </trace>
 <trace>
   366 130 20400, 359 143 20600, 354 157 20700, 349 171 20800, 352 185 21000,
   359 197 21100, 371 204 21300, 385 205 21500, 398 202 21600, 408 191 21800,
   413 177 21900, 413 163 22000, 405 150 22100, 392 143 22200
 </trace>
</ink>
```

3.2. Digital Sketching Devices

In recent years the digital recording of handwritten data, had significant developments. With smaller process units new writting tools appeared in the shape of normal pens.
This section is an overview on devices that resemble a normal pen and have the capability to digitally record handwritten text.

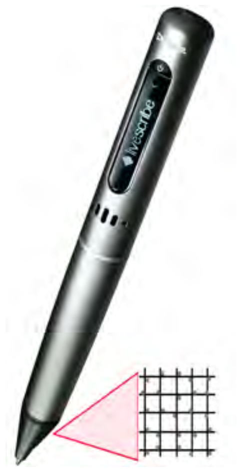


Fig. 3. Livescribe Smartpen and the dotted position system

Livescribe Smartpen The Livescribe Smartpen depicted on Figure 3 is a device with the shape of a normal pen featuring a LCD display, an infrared camera and a microphone. The LCD display is just for status information on the smartpen menu navigation and usage, the infrared camera is the key feature for recording sketched shapes and more specifically handwriting. The smartpen has an internal memory capacity of up to 4GB and a built in battery. The pen has a physical ink cartridge on the tip to sketch on the paper.

The smartpen works on a particular micro dotted paper which gives information about its position on paper. The dots on the micro dotted paper, depicted on Figure 3, are spaced about 0.3mm apart and form an apparently messy square grid. The dots appear on one of four possible positions of an imaginary square grid. The infrared camera captures a area of 6x6 dots on the paper and transform this information into a X and Y coordinate pair. The camera has a frequency of 72 captures per second which gives a sufficient sample rate to record handwritting. The smartpen store the current position when pressure is made on the tip of the pen and it is not correlated to the actual ink left on the paper. Each page on every notebook is unique for that notebook, hence the smartpen can identify the number of the current page.

The smartpen runs a system based on Java Micro Edition and can run external applications known as *Penlets*. Livescribe has a SDK for *Penlets* development but its support has been discontinued, restricting the use of custom made *Penlets*. The *Penlets* can raise events entering on active zones on the dotted paper, for instance to scroll through the main menu. The smartpen has a built in handwriting recognition (HWR) system. The dotted paper can be acquired in the form of notebooks or can be produced and printed on a standard 600 dpi laser printer. Every notebook has a Anoto Functionality Document (AFD) to describe it. This document needs to be installed on the smartpen so the printed paper sheets can be used, all the recording done on a page on a notebook is stored on the AFD structure.

For retrieving the recorded data, updating the software and recharging the battery there is a dock station. It has a desktop application, Livescribe Desktop, that can run both on Windows and Mac OS environments. This application downloads the data files from the smartpen and organizes by notebook. Livescribe has also a Desktop Application SDK for developing applications to extract and process the recorded data on the pen.

The comercial bundle has an average price of 100€ and comes with a smartpen, a dotted notebook and the dock station.

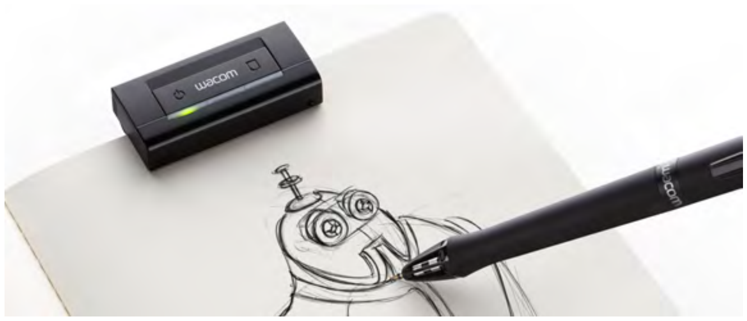


Fig. 4. Wacom Inkling

Wacom Inkling The Wacom Inkling is a sketching recording system composed by a pen and a receiver that can be clipped to the top of a paper sheet or notebook, this arrangement is depicted in Figure 4. The pen can be used to draw on the paper as it has a physical ink cartidge. The system has a memory capacity of 2GB and a built-in battery.

The operation of this tool mimics the functioning of a sonar system. The pen emits an inaudible sound, that is processed by the receiver. The receiver uses this pulse of sound to calculate the pen position and record it. The pen is pressure sensitive, this enhances the digital line weights. The system allows the definition of a new layer on the same sheet, by pressing a button on the receiver.

To transfer the data from the receiver to the computer, the receiver must be connect to the charging case. The Inkling Sketch Manager is the desktop application for downloading data from the receiver. Sketches can be saved as a single image by merging different layers or can be exported as layered files and be used on common image editors. The data can also be exported as an XML format similar to InkML.

The comercial bundle has an average price of 200€ and its composed by the pen, the receiver and a charging case.

4. HandSpy

HandSpy is a web based application to manage distributed and collaborative experiments on cognitive processes in writing. The system has the following distinctive features:

– an experiment management philosophy encompassing all the steps of the research in cognitive processes in writing;
– a multiuser web interface fostering collaboration among researchers and enabling remote work on the experiments;
– a cloud-based data management system providing central storage for all data collected in the experiments;
– an analysis process of the collected data, inspired in the state-of-art systems described in Section 2;
– the ability to select and synthesize collections of data based on different criteria;
– the use of standard XML based data formats to promote interoperability and cooperation among researchers in this community.

HandSpy system is based on a client-server model. The client makes requests to the server and the server processes the request making use of other applications to generate the response. The system follows a 3-tier architectural model as depicted in Figure 5 where the presentation tier (a web interface) is represented by the left box, the logic tier (a web server) is represented by the central box and the data tier (an XML database) is represented by the right box. This diagram represents also in three rows the data flows between these tiers. In the top row marked with number one is represented the process of uploading data files in InkML format to the database through the web interface. On the server side the database manager module is responsible for organizing the uploaded files in collections based on the current user context.

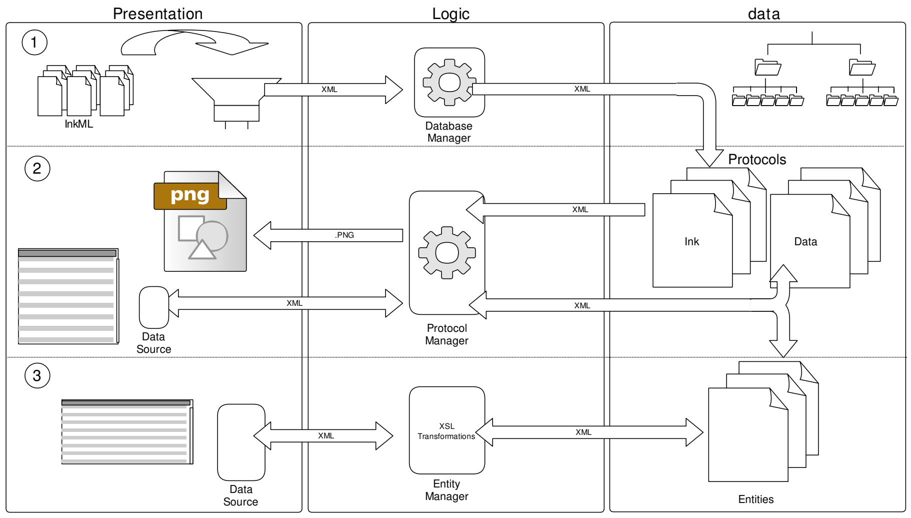


Fig. 5. HandSpy application architecture

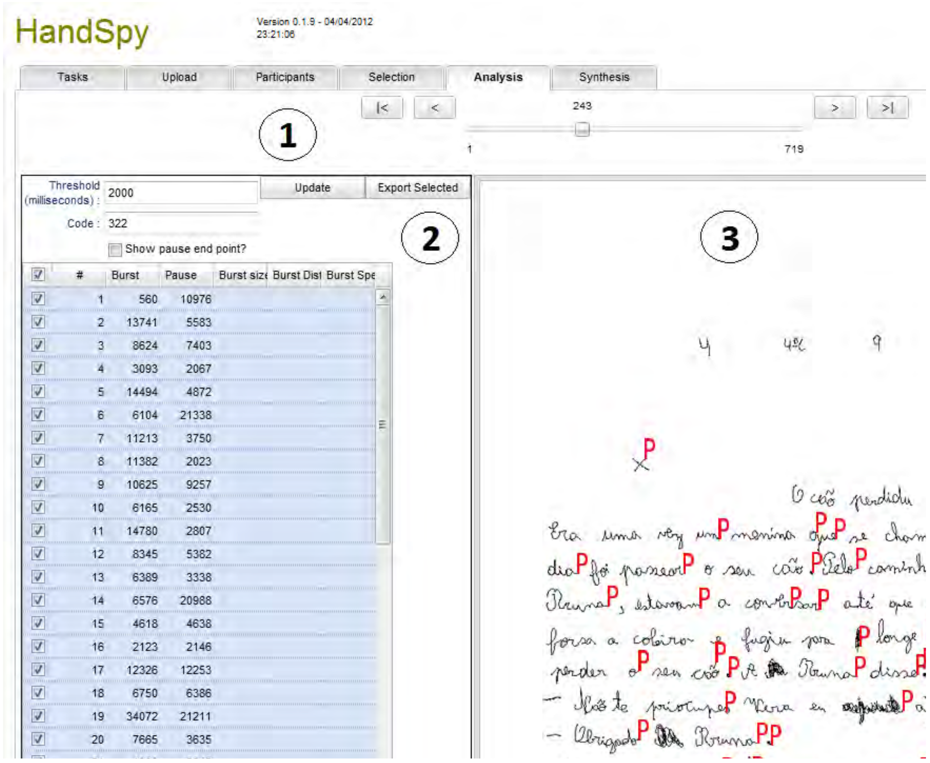
The middle row represents the interaction with ink data. The two main components of HandSpy user interface are depicted in this row, an image viewer to display the written production of the protocol ink and a list of calculations based on the protocol data. The server gets the ink of the selected protocol and generates an image file to feed the image viewer. The list grid is populated with calculations results based on the *pause* concept. To optimize the system, the main definitions on the HandSpy are classified and treated as entities. This generalization of data permits to manage it in the same way. All data showing objects are based on list grids which use XML data sources.

In the third row of the model in Figure 5 is shown the Entity Manager that identifies the entity and uses the respective XSL transformer to transform the data stored in the database into the client specific data source when the fetch operation is made. Adding, updating and deleting entities uses XSL transformations to perform the operations and save the changes to the database.

4.1. Design

This section divides the description of the system design in three parts, the *Application Interface*, *Logic* and *Data Repository*.

Application Interface The graphical user interface of HandSpy relies on a web application. The workspace is divided in six tabs covering the usual work flow of an experiment on cognitive processes in writing.



Threshold (milliseconds) : 2000

Code : 322

☐ Show pause end point?

✓	#	Burst	Pause	Burst size	Burst Dist	Burst Spe
✓	1	560	10976			
✓	2	13741	5583			
✓	3	8624	7403			
✓	4	3093	2067			
✓	5	14494	4872			
✓	6	6104	21338			
✓	7	11213	3750			
✓	8	11382	2023			
✓	9	10625	9257			
✓	10	6165	2530			
✓	11	14780	2807			
✓	12	8345	5382			
✓	13	6389	3338			
✓	14	6576	20988			
✓	15	4618	4638			
✓	16	2123	2146			
✓	17	12326	12253			
✓	18	6750	6386			
✓	19	34072	21211			
✓	20	7665	3635			

Fig. 6. HandSpy interface

Tasks Identification of tasks to be performed by the participants during the experiment. For instance, an experiment may include a task where participants must write as much letters of the alphabet as they can in a fixed amount of time.

Upload Upload of the InkML files collected with specialized hardware (smart pens or digitizing tablets) to the system. The interface displays a collection of thumbnail images of the uploaded files. Thumbnails can be selected to display a real size image for better identification. At this stage the InkML data is associated with a task and a participant.

Participants Manage the participants in the current experiment. Display the features and the tasks completed by each participant. Custom features describing the participants, such as handedness or mother language, can be added to the participants. The participants features are useful for selecting them in a particular study. The list of participants can be imported and exported as a CSV (Comma Separated Values) file.

Selection Selection of protocols based on tasks and on features of the participants such as age, handedness and gender. The selection is a collection of conditions on protocols to be analyzed and synthesized. Selections set by different researchers are independent from each other, enabling researchers to analyze different collections of protocols simultaneously.

Analysis Figure 6 is a screenshot of HandSpy interface with the Analysis tab selected. The area identified as 1 is a slider to browse the current protocol selection (set in the Selection tab). Area 2 has a form to define the parameters to calculate the pauses which are listed in the table below. The main parameter is the threshold, the time elapsed to be a pause. Each row has a pause duration, a burst duration, a burst size is a number of words present on the burst, burst distance and the burst average speed. The footer of the table presents statistics on some of its columns, such as the average and standard deviation of durations, and the count of words. Area 3 displays the written production with red Ps (for Pause) marking the place where the pauses selected in area 2 start. Pause selection allows worthless parts (for instance, a part where the participant erased a word) to be removed from the analysis. The current selection of pauses can be stored on the database using the threshold value for identification, this enables the analysis work already done on this transcript to be retrieved for further analysis.

Synthesis Displays global statistics on the data processed on the Analysis tab and is delimited by the selection criteria defined for the analysis process. The statistics presented in Analysis tab table footer for each protocol are computed on this tab aggregating all the selected protocols. These results can be exported to other systems, such as spreadsheets or statistical analysis packages.

Logic The server side of the system was designed to receive requests, process them and send the response to the client. Figure 7 depicts an image request flux on the server. The client, on the left, sends a request for an image to the server. The *Process* receives the request and authenticates the session based on the *UserContext*. If it is a valid command for that session the command *GetProcotol* is called. The *Protocol* accesses the *Database* and requests for the respective *InkML* file. The resulting image is sent back to the client through the response stream.

This flow describes the behavior of HandSpy upon a request. HandSpy deals with many requests for information in the XML format. The main difference responding to these request is in the creation of the response. Requests for XML files are created using an XSL transformations engine.

Data Repository HandSpy processes data uploaded in XML files and stores it in a native XML database. The database structure model is depicted in Figure 8. This structure keeps all the resources used by the application. Every project has a set of entities that store data on Tasks, Participants, Selection and Configurations. The data files containing the text productions of one experiment are stored in the collection Ink in the InkML format. They remain unchanged and are treated as read-only files. This enables future usage of the collected data

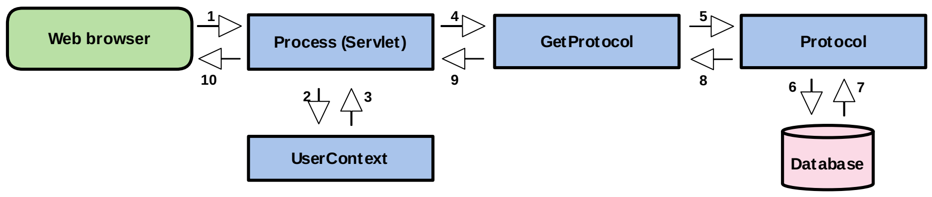


Fig. 7. Image Request Diagram

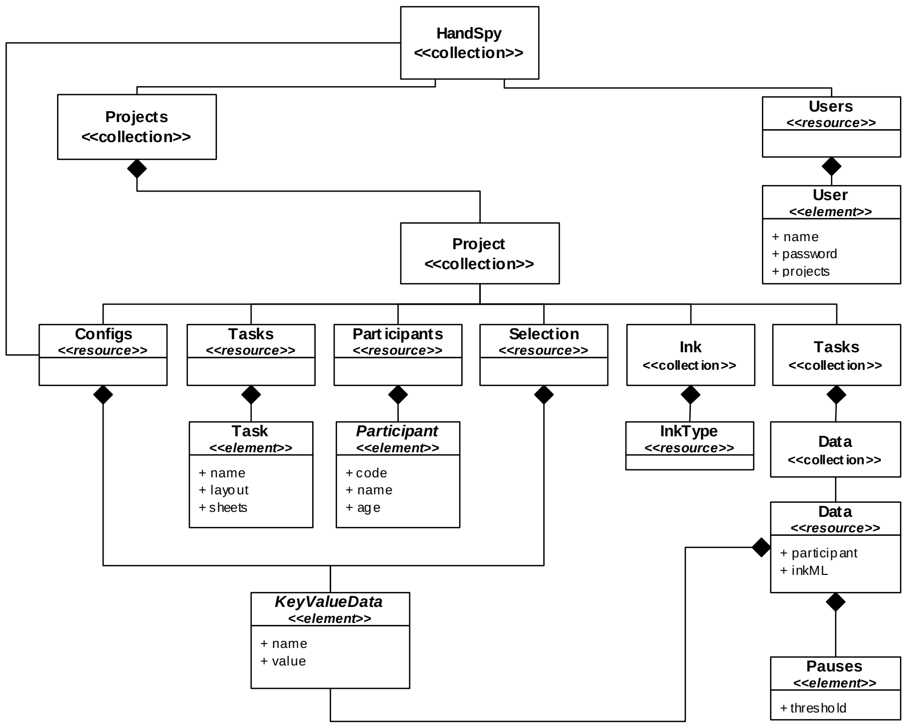


Fig. 8. Database structure diagram

for other purposes, projects or even different analysing systems. For every task added to the project a task collection is created with the name of the task, to store data files containing the calculations and other information obtained by analysing the respective *InkML* file.

The file name is the key to identify and relate the *InkML* files with the respective calculated data. The *InkML* file name is a sequential number, given when the file is uploaded. The data files stored in each task collection have the same name of the respective *InkML* file. For every task a diferent *Data* file is created and associated with the *InkML* by name. Differente studies can be conducted at the same time as every task has an independent *Data* file.

Schema Definitions The entities and the data file associated to the InkML files were specifically defined to work with the proposed architecture. An abstract data format – *KeyValueData*– which represent a mapping of a value to its key was designed and is used in several entities on the system. The following list covers the definitions of the resources presented on Figure 8.

- The *Users* resource is composed by the login name of the user, its password and a list of projects to which it has access.
- The *Configs* and *Selection* resources are composed by *KeyValueData* elements.
- The *Tasks* resource has the attributes to define the name, the layout and the sheets. The sheets value is the page interval on the notebook associated to this task.
- The *Participants* resource have the basic attribute *code*, to identify the participant and a set of *KeyValueData* elements to complete the participant details.
- The *Data* file has two attributes to identify the file. The *participant* which has the code of the participant and the *inkML* that have the name of the inkML file. Has *pauseBurstBlock* element which is a *Pauses*. The *Pauses* element defines a set of *Pause*. Each *Pause* has an attribute *threshold* and a set of *Pause* elements which is a *PauseBurst* format, with calculations for the defined threshold. The *PauseBurst* element has several attributes and a set of *facets* in the *KeyValueData* format. Figure 9 is the *Data* file schema with focus on the attributes of the *PauseBurst* element. This file stores the pauses selected on the interface tab *Analysis* described in the Section 5.

4.2. Implementation

As depicted in Figure 5, HandSpy is composed by presentation, logic and data layers. The presentation layer was implemented on SmartClient JavaScript framework. The logic layer was deployed on the Tomcat servlet container and the data layer on the eXist XML database. The remainder of this section presents the implementation of each layer, describing the implementation methods using these components as platform.

Presentation Layer The Isomorphic SmartClient LGPL platform was the selected web toolkit for the user interface. SmartClient provides sophisticated table editing widgets connected to data sources in XML formats that are appropriate to the data handled in HandSpy. These widgets have many built-in functions, such as sorting and grouping on every column, search fields and column customization. Data operations, such as fetching or querying, are built-in functions

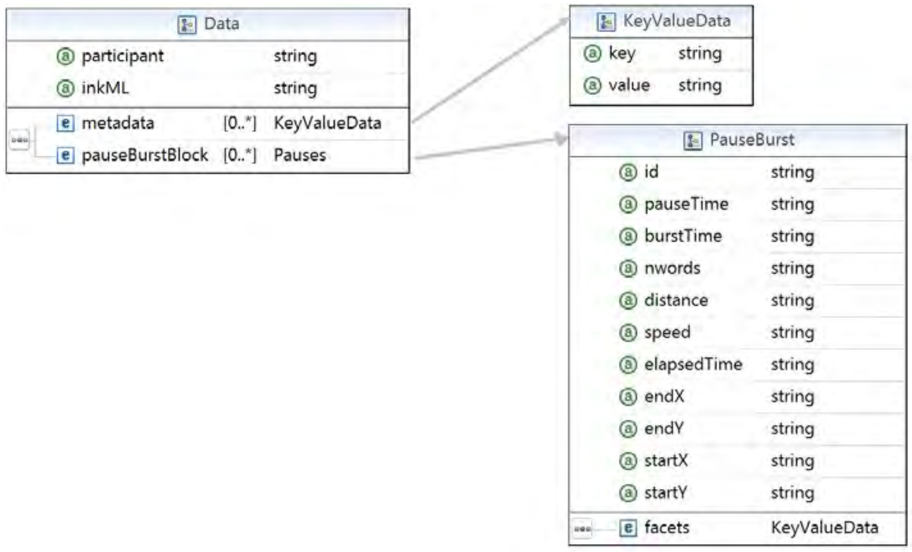


Fig. 9. *Data* schema

of the data source object. As the information can be displayed in a table fashion, the most used widget is the *ListGrid*.

HandSpy analysis gear is the most important feature of the system as it drives the research work. The analysis of the production is made with the visualtization of the text. SmartClient offers the possibility to create a HTML pane. This pane enables the use of a HTML 5 canvas. Images of the written text are generated on the server and displayed as background of HTML 5 canvas object. The use of Javascript functions enables placement of image objects representing the *pauses* starting points on the canvas, overlaying the background image. This also provides flexibility for future costumizations without being tied to widget/function limitations.

Logic Layer The server was deployed on a Tomcat - a Java Servlet container instance. This server, based on a Java Servlet, is responsible for data transactions between the *Data Layer* and *Presentation Layer* objects. The whole data processing is done on this layer as well. The data processing consists on XML transformations, generation of images, calculation of *pauses* and database management. Using eXtensible Stylesheet Language Transformations (XSLT) the information on the database is transformed into the respective data sources on the interface. The InkML files are used to generate the images and calculate the *pauses*. Creating and deleting files on the XML database is made by the server. Maintaining the whole processing on the server side reduces the need of processing power on client machines.

All the data is stored as XML files on the database. Using Java Architecture for XML Binding (JAXB) it is possible to, *marshal* Java objects into XML files and doing the inverse, *unmarshal* XML files to Java objects. This architecture uses the XSD's of the XML files to construct Java objects, with *getters* and *setters* for the XML files *elements* and *attributes*. This provides a faster and efficient generation of Java objects, that are in line with the definition of the XML structure on the XSD. These objects can be used to extract values from XML files and bind them to Java primitive data types.

This architecture is useful on the image generation and calculation process. The InkML files are bound into Java objects. By using the InkML object we can *get* the values of the (X,Y) pair for every point of every stroke in that production and draw the corresponding image. The same object is used to fetch the *timestamp* of every point and calculate the pauses and the rest of the information.

HandSpy server side is composed by a *Servlet* that is the dispatcher for client requests and a set of other functions to create responses. The next list describes some of the main functions that compose the server.

– `Process` - implements a Java Servlet instance. Acts as the single entrance point on the system, managing all client requests. The *HttpSession* and the *UserContext* are requested or created when required. The information is sent as an HTTP POST request and is parsed to retrieve the invoked command. The command is tested by an authentication method to attest its validity, if the command is valid for the authenticated user the *doRequest* method is called. Every command performed is stored on the *UserContext* for an efficient reusage of the same command.
– `Command` - is an *abstract* class to be used by the *CommandFactory*. The method *doRequest* of class Command is implemented by each command. Every command on the system needs an authenticated session to be perfomed. The *Command* have the *HttpServletRequest*, *HttpServletResponse* and the *UserContext* as arguments.
– `Protocol` - is the class for managing the *InkType* and *Data* objects that are the *unmarshalled* representations of the *InkML* and *Data* XSD. The *Protocol* class implements several methods including the *getImage*, that uses the InkType object to generate a *png* image and write it to the *OutputStream* of the *HttpServletResponse*.
– `Selector` - is the class for implementing the selection engine of the system. Makes use of the *Selection* resource described in Section 4.1. The parameters of the *Selection* resource are used to make a *XPath* query and generate a scrollable *LinkedHashMap* with the *Protocol* selection. The slider on the client interface is delimited to this selection and can be used to navigate in the selected elements.
– `UserContext` - as the name suggests, this class stores the information on the user session. For instance, the current working project and the selection

array are accessed through the *UserContext*, therefore all commands using these variables must have a valid instance of *UserContext*.
- `DBconnection` - is a *Singleton* class that implements the connection to the database. This class connects to the database using the XML:DB API, the unique point for database management. When a fresh installation of the system is made the method *createDataBaseStructureIfNeeded* is invoked to create the basic structure of the database.
- `EntityManager` - is the class that implements the *Add*,*Remove*,*List* and*Update* operations on the system resources. Every resource type has a XSL Transformation for each operation. Using a *DBconnection* instance the resource files on the database are requested and used to perfom these operations. The result of the operation is written to the *OutputStream*.

Data Layer The *Data Layer* was implemented using eXist [4] database management system. As the system uses XML files to represent all the information on the system, choosing a native XML database was the most suitable option.

The database system is installed on the same machine as the Tomcat server and is remotely accessed through a socket. The database is exclusively managed by the HandSpy server. The HandSpy database structure described in Section 4.1 is created when the HandSpy starts for the first time.

5. Collecting Framework

As the focus of the project is based on writing productions, a tool for collecting handwritten data was developed. The device used to collect the data for this experiment was the Livescribe Smartpen, already described in Chapter 3.

There are several advantages in using a smartpen instead of digitizing tablets traditionally used for this kind of experiment. The possibility of setting up an experiment in a classroom, a familiar place to the participants and being a writing device similar to the pens normally used by school children. These features make the smartpen less intrusive than digitizing tablets. The cost of running the experiment with smart pens is also relevant because the price of a single digitizing tablet is equivalent to several pens. They are easy to carry, a single researcher can set up and supervise several participants at once. The pens can record several experiment tasks without the need to be connected to download the data to the computer. A single computer can be used to download all the data in every pen.

To generate the data for HandSpy with the Livescribe Smartpen was developed a framework consisting of three parts, a *Penlet* to record the necessary data to calculate the *pause* and *burst* time, a *Paper Application* with specific active regions to control the experiment and finally the *Data File Generator* to extract the collected data and create files in the InkML format. The following sections on this chapter cover the development of the three components of the framework and describes a series of recurring problems of using this kind of smartpen.

5.1. Penlet

The smartpen records data written on the paper on the AFD file of the *paper application*. This data is used by the Livescribe Desktop to organize the downloaded data and render the respective drawing. For the purpose of the experiment the default data recorded on the AFD was not sufficient to calculate the pause and burst time. By default the only information on the strokes that can be retrieved with the AFD file was the starting time.

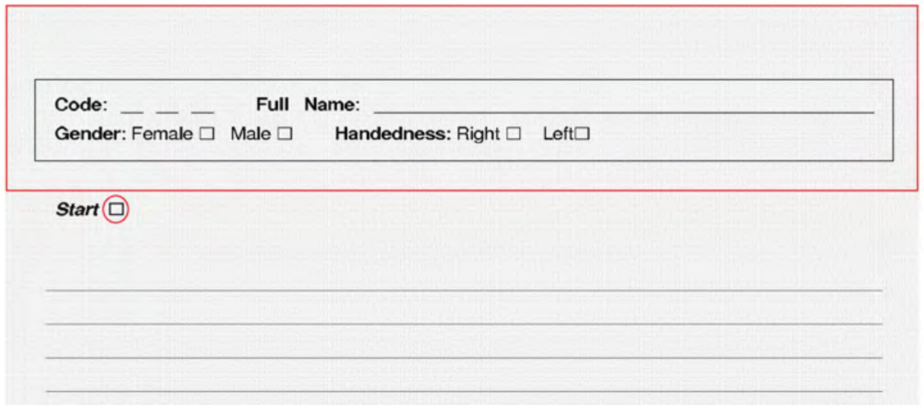
The *penlet* was developed with the Livescribe Pen API. Using the interface – *PenletStorage* – storing a plain *text* file in the internal storage pool. For every stroke, the *timestamp* of every point in the stroke was written on the file. This extra timestamp enables the calculation of pauses occured within a stroke.

To record the moments when each experiment started and ended, active regions were defined in the paper application. The active regions raise events on the *penlet* when the specified regions are entered or exited. These regions set a timestamp for the beginning and the end of the experiment. The timestamp is over overriden if the active zone is repeatedly entered. These timestamps are also written on the same file as extra information on the experiment and are used to calculate the time taken to actually start the task.
A visual feedback on the status of the penlet is given through the display on the smartpen. When the penlet starts its version is shown on the display as well as the interaction with the active regions.
The *penlet* is associated with the specific paper application created for the experiment, the *penlet* starts running when the tip of the pen touches the sheet of paper. On every change of paper sheet the penlet writes a control line on the *text* file to identify a new collection of strokes.

5.2. Paper Application

A *paper application* is an AFD file with the digital description of each sheet of a notebook. Different paper applications were design to perfom the different tasks for the experiment. The paper application was developed in the Integrated Development Environment (IDE), Eclipse. Livescribe provides an Eclipse plugin to create an AFD file and interactively draw the active regions. A background image, on the PostScript(PS) format, defines the page layout, with the page header and a place for the action buttons.
All tasks, follow the same layout. The basic layout of the paper sheet consists of three active regions. A top region to place the header, the start and the end button region. The active regions can be drawn aided by the background image, to ensure their exact position.

Figure 10 shows an example of a paper application with highlighted active regions. In this case the header region acted as a passive region. Even



Code: __ __ __ **Full Name:** ______________________

Gender: Female ☐ Male ☐ **Handedness:** Right ☐ Left☐

Start ☐

Fig. 10. Paper Application - active regions in red

if the participant didn't "touch" the start button, the start timestamp setter was activated by an event raised by any stroke made outside the header or start delimited region.

5.3. Data File Generator

The Livescribe Desktop SDK is a C# API to extract data from the smartpen used for generating the InkML files.
The AFD files and the internal storage pool, where *text* file with the extra information on the strokes is stored, are accessed. For every control line with the identification of a new collection of strokes found in the file, a new InkML file is created. The next lines have the timestamp of the beginning of each stroke preceded by increments of milliseconds of every point within the stroke. The time increments have as reference the beginning of the stroke timestamp. These increments represent the time of each point within the stroke. The timestamps are used to access the AFD file and retrieve the X and Y coordinates of every point given the timestamp. The InkML file is written with the X,Y coordinates and the respective timestamp. The Listing 1.2 is an example of *trace* element generated with information of every point, following the schema [X Y Timestamp] separated by commas. This data was generated from a real collection made with the smartpen.

5.4. Hardware Issues

The collecting framework was used on a real experiment cenario while it was being developed. This experience led us to avoid some features of the smartpen and to reimplement som functions of the penlet. Finally we managed to make a good practical use of the device for an efficient data collection.

Listing 1.2. InkML trace element example

```
...
<trace>
 2534 685 37297520816, 2537 684 37297520829, 2539 681 37297520843,
 2544 678 37297520856, 2546 677 37297520883, 2546 678 37297520896,
 2548 680 37297520909, 2549 682 37297520923, 2554 695 37297520949,
 2559 707 37297520963, 2565 720 37297520976, 2572 732 37297520989,
 2580 749 37297521016, 2582 753 37297521029, 2584 754 37297521043,
 2584 754 37297521056, 2584 754 37297521083, 2584 754 37297521096,
 2584 754 37297521109, 2584 754 37297521123, 2584 753 37297521149,
 2585 751 37297521163, 2586 748 37297521176, 2592 738 37297521189,
 2603 710 37297521216, 2611 694 37297521229, 2615 684 37297521243,
 2620 673 37297521256, 2620 670 37297521283, 2621 670 37297521296
</trace>
...
```

Our first intention was to use smartpen to react to active regions. On the first version of the paper application the header had active regions to define each field. The information written on the fields was processed by the HWR engine to automatically transform the letters and numbers into its character codes. To improve the HWR success different contexts were associated with each field. For the *code* identification of the participant, as it was a numeric field, the context was set to recognize only numeric symbols and for the *name* field, only characters. This entailed a change of context for almost every field and consequently an unexpected overload on the smartpen processing capacity. This overload caused a significant increase in stroke losses, which invalidate an entire collection. The ratio of successful recognitions was not enough to be useful therefore the use of the HWR was discontinued.

The experiment participants are intended to be school aged children. The use of audible signals to prompt entering active regions is also discouraged, as it distracts the children and could led to an active region touching spree, invalidating the experiment timestamps. Children tend to hold a pen close to its tip. As the smartpen makes use of the infrared camera to work and it is located on the tip of the pen, it is necessary to pick the pen in a way so the camera is not blocked by any finger.

The Livescribe Desktop SDK is limited to Windows environment. On the first month of the development of the collecting framework, a surprising business move from Livescribe occurred, they discontinued the development program, ceasing the support and updates on the Livescribe SDK. This led to run system updates more thoroughly.

6. HandSpy Usability Evaluation

The project, *Develop Automate and Auto Regulating cognitive processes in writing composition* (DAAR), focus on the development of cognitive processes in written production. It aims to relate the automation of writing processes and self-regulation of others with the development of this competence.

The plan of the study is divided in two phases. On the first phase, which was the first year of HandSpy development, the participants were children from the second grade to the seventh. The studies characterize the text production and the involved cognitive processes. The second phase will be divided in two interventions. The participants of the first intervention will be children from first to fourth grade and will focus on transcription skills. The second, with children in the fourth grade, will focus on strategies of self-regulation in writing.

In the first year of the development, HandSpy is being used to store the collected data of the first phase of the study. More than two thousand productions were collected on several tasks performed with five hundred and sixty children. The collections were made with the collecting device described in Chapter 5 with groups of fifteen children at a time making five different writting tasks. After collecting the data was uploaded to the HandSpy system and automatically stored on the XML database. HandSpy is currently being used to analyze the data of the first phase of the project.

The social scientists on this project were the users who had more contact with HandSpy therefore they were the main assessors of the system usability. Besides the evaluation that was made throughout the development which identified some problems, an evaluation based on the completion of a questionnaire was also made. The evaluation method is described in the following sections.

6.1. Heuristic Evaluation

Heuristic evaluation is on the most popular methods to identify problems in the user interface design. An heuristic is a set of rules and methods to solve problems. Rolf Molich and Jakob Nielsen [6] describe the heuristic evaluation as ”an informal method of usability analysis where a number of evaluators are presented with an interface design and asked to comment on it”.

After evaluating different heuristics, Nielsen created a list with the best heuristics to identify interface usability problems [5].

- **Visibility of system status** - The system should always give operation status.
- **Compatibility** - The system should use familiar language to the user. Information should appear in a natural order.
- **User control and freedom** - Support undo and redo operations to recover from choosing functions by mistake.
- **Consistency and standards** - The interface should use consistent colors, operations names and layout.
- **Error prevention** - Try to prevent errors from ocurring displaying confirmation on critical operations.
- **Recognition rather than recall** - Minimize the users memory load by making objects, actions, and options always visible.
- **Flexibility and efficiency of use** - Permission for the user to personalize frequent actions.

– **Aesthetic and minimalist design** - The information displayed must be relevant.
– **Help users recognize, diagnose, and recover from errors** - Error messages should be clear, precisely indicate the problem and suggest a solution.
– **Help and documentation** - Help and documentation should always be available.

6.2. Evaluation

The evalution made on HandSpy usability was based on the results of a questionnaire. The questionnaire was based on the heuristic set listed in Section 6.1. The questionnaire was answered by three evaluators. The questionnaire consisted on a multiple choice answer system. In the Figure 11 is the graph with percentage of each heuristic. The results were processed as follows:

– For each group of questions the possible answers were - **Does not apply** – **Never** – **Almost Never** – **Regular** – **Almost Always** – **Always**
– The total number of effective answers is calculated by subtracting the **Does not apply** answers to the total answers.
– The percentage of the answers ”Never/Almost Never”, ”Regular”, ”Always/Almost Always” is calculated based on effective answers.

With the analysis of graph is clear that the critical issues on the interface are the lack of help, documentation and poor flexibility. These problems are reinforced by the evaluators in the comments *”Insufficient help menus and still arise many errors that are not comprehended”*, *”Integrate the help in tutorial format, improve ergonomics and clarity of controls and functions”*. We can verify that the heuristics better accomplish are *”Compatibility”* and *”Recognition rather than recall”*. Despite a better classification, some comments made on these heuristics suggest some improvements on some specific components *”Improve the way to confirm the selection of data”*, *”Improve interactivity with the data from participants” and ”Transparency for the user's project idea and its management”*. The comments clearly show that project management is the feature that deserves more improvements on usability. Despite having quite a few negative points the interface meets satisfactorily the usability heuristics.

The answer to, an overall evaluation of the system, *”Considering all the parameters that you analyzed how would you rank HandSpy?”* was unanimous, all evaluaters answer that the system is *”Merely Adequate”*. Despite some severe faults on the interface this evaluation showed that HandSpy has potential to be a reference in this field, improving some aspects on the user experience.

7. Conclusion

With the use of new devices capable of recording hand gestures, the use of digital handwriting as a transferable data is becoming more common. These new

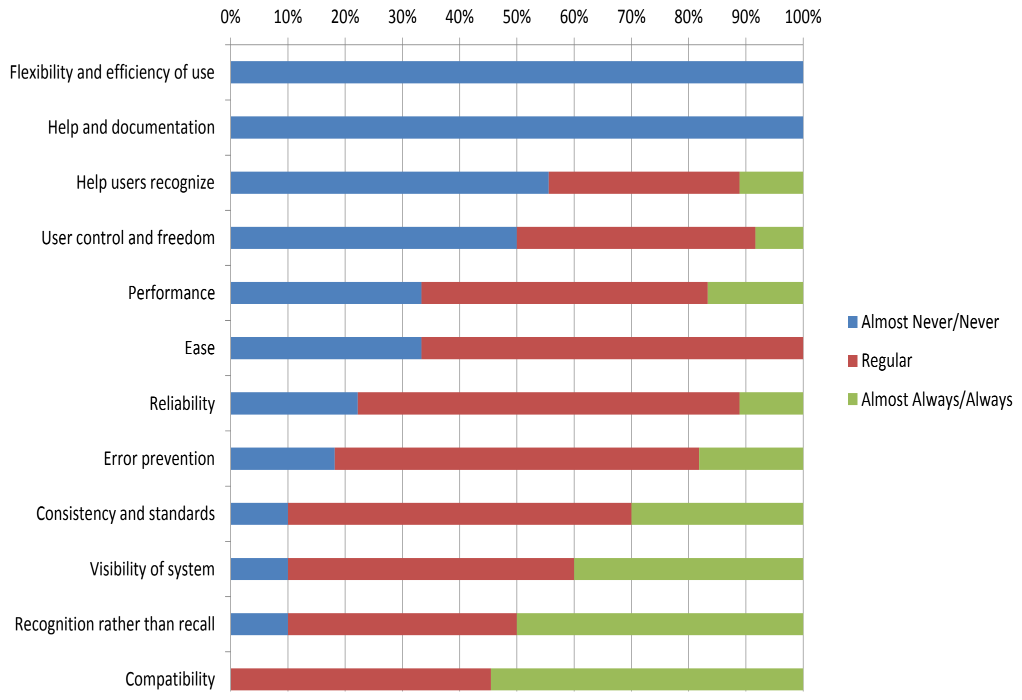


Fig. 11. Heuristic Evaluation

possibilities provide new ways of studying the cognitive processes involved in the handwriting process. This paper describes the design, implementation and evalution of a new tool, HandSpy, to support the study on cognitive processes in writing.
HandSpy aims to manage and support handwriting research studies with large amounts of data and enable collaborative work to speed up the analysis process. Embedded in a web platform, HandSpy is a powerful tool to be used as a cross platform environment. With the use of the web browser as the main working tool, it obviates the need for installing various programs, on various machines.

The collecting framework described in the Chapter 5 is a new tool for recording writing productions. The use of a commercial product such as the Livescribe smartpen to collect written productions results on a affordable, easy to use and less intrusive compared to other tools for this purpose. This tool has already raised interest among the social sciences research community.

7.1. Future Work

As future work the evolution of HandSpy will consist in user interface upgrades and expanding the collection to new data elements. The evaluation of HandSpy defined the next steps in the user interface upgrades. Based on the outcome of the questionnaire along with a series of suggestions made by the evaluators, we present here some of the main future implementations to improve user interaction.

– Incorporate user guides and tutorials on the interface, offering information on the current screens.
– Improve error handling giving specific feedback of the error.
– Optimize interaction in the analysis screen giving a more accurate selection and identification of the pauses.
– Create a real time animation playback of the written text.

The smartpen has a built-in microphone which enables collection extension with audio data. This can be used to record information on what participants are thinking, if they are asked to ”think out loud”, while writing. In this case synchronizing the audio with the writing is eased as they are collected with the same device. Collecting physiologic data such as heart rate or electric conductivity of the skin can be useful to relate with the writing pauses. Video recording the production is also an added value for the research but only if we manage to retrieve the point of regard on the paper during the writing production. Nevertheless synchronizing video and physiologic data with the writing raises new challenges.

Acknowledgments. This work is in part funded by the ERDF/COMPETE Programme and by FCT within the projects FCOMP-01-0124-FEDER-022701 & PTDC/PSI-PCO/110708/2009.

References

1. D. Alamargot, D. Chesnet, C.D., Ros, C.: Eye and pen: A new device for studying reading during writing. Behavior Research Methods (2006)
2. E.Guinet, Kandel, S.: Ductus: A software package for the study of handwriting production. Behavior Research Methods (2010)
3. Ioannidis, Y.E., Livny, M.: Conceptual schemas: Multi-faceted tools for desktop scientific experiment management. Journal of Intelligent and Cooperative Information Systems 1, 451–474 (1992)
4. Meier, W.: e[x]ist: An open source native xml database. Web, Web-Services, and Database Systems (2003)
5. Nielsen, J.: Enhancing the explanatory power of usability heuristics. In: Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence. pp. 152–158. ACM, New York, NY, USA (1994)
6. Nielsen, J., Molich, R.: Heuristic evaluation of user interfaces. In: Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people. pp. 249–256. ACM, New York, NY, USA (1990)
7. Olive, T., Alves, R.A., Castro, S.L.: Cognitive processes in writing during pause and execution periods. European Journal of Cognitive Psychology 21(5), 758–785 (2009)

Carlos Monteiro is currently a Telecommunications Engineer at Inmarsat, a satellite telecommunications company. During his master's degree he was involved in a research project that was focused on digital handwriting formats and analysis, based on XML.

José Paulo Leal is assistant professor at the department of Computer Science of the Faculty of Sciences of the University of Porto (FCUP) and associate researcher of the Center for Research in Advanced Computing Systems (CRACS). His main research interests are eLearning system implementation, structured document processing and software engineering. He has a special interest on automatic exercise evaluation, in particular on the evaluation of programming exercises, on ontology processing and on web adaptability. He has participated in several research projects in his main research areas, including technology transfer projects with industrial partners. He has over 60 publications in conference proceedings, journals and book chapters.

Received: November 30, 2012; Accepted: July 9, 2013.

DOI: 10.2298/CSIS121129066A

Batched Evaluation of Linear Tabled Logic Programs

Miguel Areias and Ricardo Rocha

CRACS & INESC TEC and Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
{miguel-areias,ricroc}@dcc.fc.up.pt

Abstract. Logic Programming languages, such as Prolog, provide a high-level, declarative approach to programming. Despite the power, flexibility and good performance that Prolog systems have achieved, some deficiencies in Prolog's evaluation strategy - SLD resolution - limit the potential of the logic programming paradigm. Tabled evaluation is a recognized and powerful technique that overcomes SLD's susceptibility in dealing with recursion and redundant sub-computations. In a tabled evaluation, there are several points where we may have to choose between different tabling operations. The decision on which operation to perform is determined by the scheduling algorithm. The two most successful tabling scheduling algorithms are *local scheduling* and *batched scheduling*. In previous work, we have developed a framework, on top of the Yap Prolog system, that supports the combination of different *linear tabling strategies* for local scheduling. In this work, we propose the extension of our framework to support batched scheduling. In particular, we are interested in the two most successful linear tabling strategies, the DRA and DRE strategies. To the best of our knowledge, no other Prolog system supports both strategies simultaneously for batched scheduling. Our experimental results show that the combination of the DRA and DRE strategies can effectively reduce the execution time for batched evaluation.

Keywords: logic programming, linear tabling, scheduling.

1. Introduction

Logic programming provides a high-level, declarative approach to programming. Arguably, Prolog is one of the most popular and powerful logic programming languages. Ideally, one would want Prolog programs to be written as logical statements first, and for control to be tackled as a separate issue. In practice, the operational semantics of Prolog is given by SLD resolution [9], an evaluation strategy particularly simple but that suffers from fundamental limitations, such as in dealing with recursion and redundant sub-computations. Unfortunately, the limitations of SLD resolution mean that Prolog programmers must be concerned with SLD semantics throughout program development.

Tabling [4] is a proposal that overcomes SLD limitations in dealing with recursion and redundant sub-computations. Tabling based models are able to reduce the search space, avoid looping, and always terminate for programs with the *bounded term-size property*[1]. In a nutshell, tabling consists of storing intermediate solutions for subgoals so that they can be reused when a similar subgoal appears during the execution of a program and, for that, the calls and the solutions to tabled subgoals are stored in a global data structure called the *table space*. Work on tabling, as initially implemented in the XSB system [11], proved its viability for application areas such as Natural Language Processing, Knowledge Based Systems, Model Checking, Program Analysis, among others.

In a tabled evaluation, there are several points where we may have to choose between continuing forward execution, backtracking, consuming solutions from the table, or completing subgoals. The decision on which operation to perform is determined by the scheduling strategy. Whereas a strategy can achieve very good performance for certain applications, for others it might add overheads and even lead to unacceptable inefficiency. The two most successful strategies are *local scheduling* and *batched scheduling* [7]. Local scheduling tries to complete subgoals as soon as possible. When new solutions are found, they are added to the table space and the evaluation fails. Solutions are only returned when all program clauses for the subgoal at hand were resolved. Batched scheduling favors forward execution first, backtracking next, and consuming solutions or completion last. It thus tries to delay the need to move around the search tree by batching the return of solutions. When new solutions are found for a particular tabled subgoal, they are added to the table space and the evaluation continues.

The main difference between the two strategies is that in batched scheduling, variable bindings are immediately propagated to the calling environment when a solution is found. For some situations, this may result in creating complex dependencies between subgoals and in having more memory space requirements. On the other hand, since local scheduling delays solutions, it does not benefit from binding propagation, and instead, when explicitly returning the delayed solutions, it incurs an extra overhead for copying them out of the table.

Currently, the tabling technique is widely available in systems like XSB Prolog [14], Yap Prolog [12], B-Prolog [15], ALS-Prolog [8], Mercury [13] and Ciao Prolog [5]. In these implementations, we can distinguish two main categories of tabling mechanisms: *suspension-based tabling* and *linear tabling*. Suspension-based tabling mechanisms need to preserve the computation state of suspended tabled subgoals in order to ensure that all solutions are correctly computed. A tabled evaluation can be seen as a sequence of sub-computations that suspend and later resume. Linear tabling mechanisms use iterative computations of tabled subgoals to compute fix-points and, for that, they maintain a single execution tree without requiring suspension and resumption of sub-computations. For that reason, linear tabling mechanisms have less memory space requirements and can be implemented with less disruption of an existing Prolog engine. On the other hand, linear tabling mechanisms can be arbitrarily

[1] A logic program has the bounded term-size property if there is a function $f : N \to N$ such that whenever a query goal Q has no argument whose term size exceeds n, then no term in the derivation of Q has size greater than $f(n)$.

slower than suspension-based tabling. However, they are still very competitive on a large number of examples. In particular, for batched scheduling, they may have an additional advantage since, with suspension-based tabling, some evaluations may require very large amounts of space.

In previous work, we have developed a framework, on top of the Yap Prolog system, that supports the combination of different linear tabling strategies for local scheduling [1, 2]. As these strategies optimize different aspects of the evaluation, they were shown to be orthogonal to each other for local scheduling. In this work, we propose the extension of our framework, to combine different linear tabling strategies, but for batched scheduling. In particular, we are interested in the two most successful linear tabling strategies, the DRA and DRE strategies [2]. To the best of our knowledge, no other Prolog tabling system supports both strategies simultaneously for batched scheduling. Extending our framework from local scheduling to batched scheduling should be, in principle, smooth but, as we will see, there are some relevant details that have to be considered in order to ensure a correct and efficient integration of the DRA and DRE strategies with batched scheduling. In more detail, this integration required changes to the table space data structures, to the tabling operations and a new mechanism to support the propagation of solutions in reevaluation rounds.

Our experimental results show that the combination of the DRA and DRE strategies can effectively reduce the execution time for batched evaluation. When compared with Yap's suspension-based mechanism, the commonly referred weakness of linear tabling of doing a huge number of redundant computations for computing fix-points was not such a problem in our experiments. We thus argue that an efficient implementation of linear tabling can be a good and first alternative to incorporate tabling into a Prolog system without such support.

The remainder of the paper is organized as follows. First, we briefly introduce the basics of tabling and describe the execution model for standard linear tabled evaluation using batched scheduling. Next, we present the DRA and DRE strategies and discuss how they optimize different aspects of the evaluation. We then describe the most relevant implementation details regarding the integration of the two strategies on top of the Yap Prolog system. Finally, we present experimental results and we end by outlining some conclusions.

2. Standard Linear Tabled Evaluation

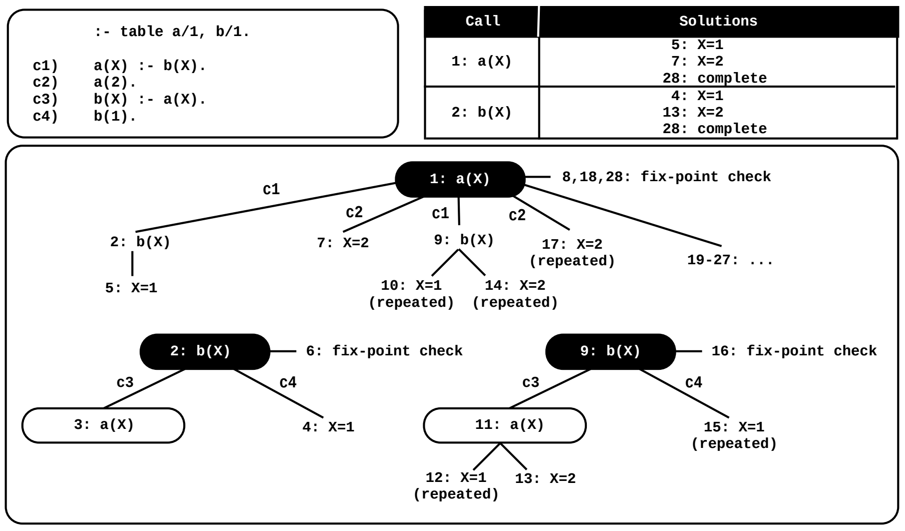
Tabling works by storing intermediate solutions for tabled subgoals so that they can be reused when a similar[2] (or repeated) call appears. In a nutshell, first calls to tabled subgoals are considered *generators* and are evaluated as usual, using SLD resolution, but their solutions are stored in a global data space, called the *table space*. Similar calls to tabled subgoals are considered *consumers* and

[2] For the sake of simplicity, we are assuming a *variant-based tabling* mechanism, where two terms are considered to be similar if they are the same up to variable renaming. Alternatively, *subsumption-based tabling* mechanisms consider that two terms are similar if one term *subsumes* (is more general than) the other [6].

are not reevaluated against the program clauses because they can potentially lead to infinite loops, instead they are resolved by consuming the solutions already stored for the corresponding generator. During this process, as further new solutions are found, we need to ensure that they will be consumed by all the consumers, as otherwise we may miss parts of the computation and not fully explore the search space.

A generator call *C* thus keeps trying its matching clauses until a fix-point is reached. If no new solutions are found during one round of trying the matching clauses, then we have reached a fix-point and we can say that *C* is completely evaluated. However, if a number of subgoal calls is mutually dependent, thus forming a *Strongly Connected Component (SCC)*, then completion is more complex and we can only complete the calls in a SCC together [11]. SCCs are usually represented by the *leader call*, i.e., the generator call which does not depend on older generators. A leader call defines the next completion point, i.e., if no new solutions are found during one round of trying the matching clauses for the leader call, then we have reached a fix-point and we can say that all subgoal calls in the SCC are completely evaluated.

We next illustrate in Fig. 1 the standard execution model for linear tabling using batched scheduling. At the top, the figure shows the program code (the left box) and the final state of the table space (the right box). The program defines two tabled predicates, *a/1* and *b/1*, each defined by two clauses (clauses *c1* to *c4*). The bottom sub-figure shows the evaluation sequence, numbered in order of evaluation, for the query goal *a(X)*. Generator calls are depicted by black oval boxes and consumer calls by white oval boxes.



```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

Call	Solutions
1: a(X)	5: X=1 7: X=2 28: complete
2: b(X)	4: X=1 13: X=2 28: complete

Fig. 1. A standard linear tabled evaluation using batched scheduling

The evaluation starts by inserting a new entry in the table space representing the generator call *a(X)* (step 1). Then, *a(X)* is resolved against its first matching clause, clause *c1*, calling *b(X)* in the continuation. As this is a first call to *b(X)*, we insert a new entry in the table space representing *b(X)* and proceed as shown in the bottom left tree (step 2). Subgoal *b(X)* is also resolved against its first matching clause, clause *c3*, calling again *a(X)* in the continuation (step 3). Since *a(X)* is a repeated call, we try to consume solutions from the table space, but at this stage no solutions are available, so execution fails.

We then try the second matching clause for *b(X)*, clause *c4*, and a first solution for *b(X)*, {*X=1*}, is found and added to the table space (step 4). We then follow a batched scheduling strategy and the evaluation continues with *forward execution* [7]. With batched scheduling, new solutions are immediately returned to the calling environment, thus the solution for *b(X)* should now be propagated to the context of the previous call, which also originates a first solution for *a(X)*, {*X=1*} (step 5).

The execution then fails back to node 2 and we check for a fix-point (step 6), but *b(X)* is not a leader call because it has a dependency (consumer node 3) to an older call, *a(X)*. Remember that we reach a fix-point when no new solutions are found during the last round of trying the matching clauses for the leader call. Then, we try the second matching clause for *a(X)* and a second solution for it, {*X=2*}, is found and added to the table space (step 7). We then backtrack again to the generator call for *a(X)* and because we have already explored all matching clauses, we check for a fix-point (step 8). We have found new solutions for both *a(X)* and *b(X)* in this round, thus the current SCC is scheduled for reevaluation.

The evaluation then repeats the same sequence as in steps 2 to 3 (now steps 9 to 11), but since we are following a batched scheduling strategy, we first consume the solutions already available for *b(X)* (this will be further explained later in section 4), which leads to a repeated solution for *a(X)* (step 10). Tabling does not store duplicate solutions in the table space. Instead, repeated solutions fail. Next, the evaluation moves to the consumer call of *a(X)* (step 11). Solution {*X=1*} is first forwarded to it, which originates a repeated solution for *b(X)* (step 12) and thus execution fails. Then, solution {*X=2*} is also forward to it and a new solution for *b(X)* is found (step 13) and propagated to *a(X)*, which leads to a repeated solution for *a(X)* (step 14).

In the continuation, we find another repeated solution for *b(X)* (step 15) and we fail a second time in the fix-point check for *b(X)* (step 16). Again, as we are following a batched scheduling strategy, the solutions for *b(X)* were already all propagated to the context of *a(X)*, thus we can safely backtrack to the generator call for *a(X)*. Because we have found a new solution for *b(X)* during this last round, the current SCC is scheduled again for reevaluation (step 18). The reevaluation of the SCC does not find new solutions for both *a(X)* and *b(X)* (steps 19 to 27). Thus, when backtracking again to *a(X)* we have reached a fix-point and because *a(X)* is a leader call, we can declare the two subgoal calls to be completed (step 28).

3. Linear Tabling Strategies

The standard linear tabling mechanism uses a naive approach to evaluate tabled logic programs. Every time a new solution is found during the last round of evaluation, the complete search space for the current SCC is scheduled for reevaluation. However, some branches of the SCC can be avoided, since it is possible to know beforehand that they will only lead to repeated computations, hence not finding any new solutions. Next, we present two different strategies for optimizing the standard linear tabled evaluation. The common goal of both strategies is to minimize the number of branches to be explored, thus reducing the search space, and each strategy tries to focus on different aspects of the evaluation to achieve it.

3.1. Dynamic Reordering of Alternatives

The key idea of the *Dynamic Reordering of Alternatives (DRA)* strategy, as originally proposed by Guo and Gupta [8], is to memoize the clauses (or alternatives) leading to consumer calls, the *looping alternatives*, in such a way that when scheduling an SCC for reevaluation, instead of trying the full set of matching clauses, we only try the looping alternatives.

Initially, a generator call *C* explores the matching clauses as in standard linear tabled evaluation and, if a consumer call is found, the current clause for *C* is memoized as a looping alternative. After exploring all the matching clauses, *C* enters the *looping state* and from this point on, it only tries the looping alternatives until a fix-point is reached. Figure 2 uses the same program from Fig. 1 to illustrate how DRA evaluation works.

The evaluation sequence for the first SCC round (steps 2 to 7) is identical to the standard evaluation of Fig. 1. The difference is that this round is also used to detect the alternatives leading to consumers calls. We only have one consumer call at node 3 for *a(X)*. The clauses in evaluation up to the corresponding generator, call *a(X)* at node 1, are thus marked as looping alternatives and added to the respective table entries. This includes alternative *c3* for *b(X)* and alternative *c1* for *a(X)*. As for the standard strategy, the SCC is then scheduled for two extra reevaluation rounds (steps 9 to 15 and steps 17 to 23), but now only the looping alternatives are evaluated, which means that the clauses *c2* and *c4* are ignored.

3.2. Dynamic Reordering of Execution

The second strategy, that we call *Dynamic Reordering of Execution (DRE)*, is based on the original SLDT strategy, as proposed by Zhou et al. [16]. The key idea of the DRE strategy is to give priority to the program clauses and, for that, it lets repeated calls to tabled subgoals execute from the *backtracking clause of the former call*. A first call to a tabled subgoal is called a *pioneer* and repeated calls are called *followers* of the pioneer. When backtracking to a pioneer or a follower, we use the same strategy and we give priority to the exploitation of the

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

Call	Solutions	Looping Alternatives
1: a(X)	5: X=1 7: X=2 24: complete	3: c1
2: b(X)	4: X=1 13: X=2 24: complete	3: c3

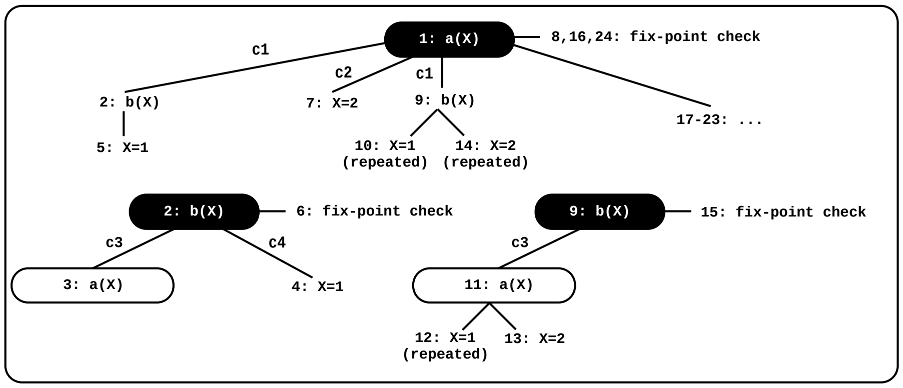
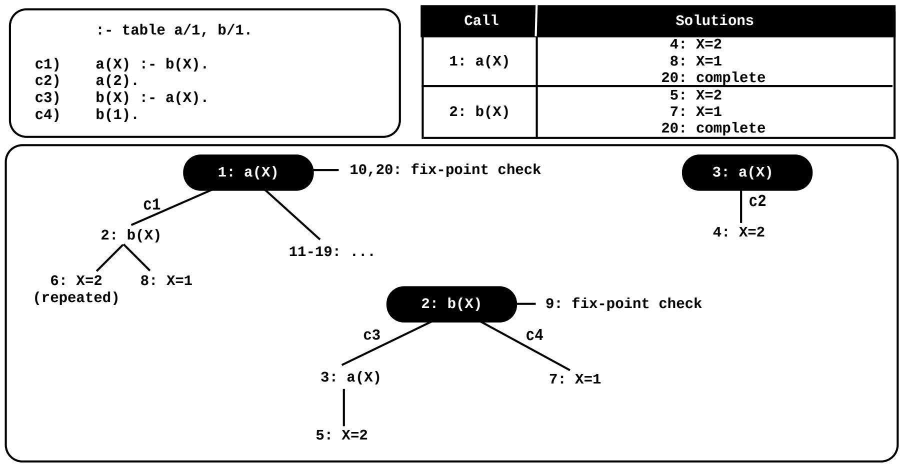


Fig. 2. A linear tabled evaluation using batched scheduling with DRA evaluation

remaining clauses. The fix-point check operation is still performed by pioneer calls. Figure 3 uses again the same program from Fig. 1 to illustrate how DRE evaluation works.

As for the standard strategy, the evaluation starts with (pioneer) calls to *a(X)* (step 1) and *b(X)* (step 2), and then, in the continuation, *a(X)* is called repeatedly (step 3). With DRE evaluation, *a(X)* is now considered a follower and thus we *steal* the backtracking clause of the former call at node 1, i.e., clause *c2*. The evaluation then proceeds as for a generator call (right upper tree in Fig. 3), which means that new solutions can be generated for *a(X)*. We thus try clause *c2* and a first solution for *a(X)*, $\{X{=}2\}$, is found and added to the table space (step 4). Then, we follow a batched scheduling strategy and the solution $\{X{=}2\}$ is propagated to the context of *b(X)*, which originates the solution $\{X{=}2\}$ (step 5), and to the context of *a(X)*, which leads to a repeated solution (step 6).

As both matching clauses for *a(X)* were already taken, the execution backtracks to the pioneer node 2. Next, we find a second solution for *b(X)* (step 7), which is then propagated, leading also to a second solution for *a(X)* (step 8). In step 9, we check for a fix-point, but *b(X)* is not a leader call because it has a dependency (follower node 3) to an older call, *a(X)*. We then backtrack to the pioneer call for *a(X)* and because we have already explored the matching clause *c2* in the follower node 3, we check for a fix-point. Since we have found new solutions during the last round, the current SCC is scheduled for reevaluation (step 10). As the full set of solutions was already found during the first round, the reevaluation of the SCC does not find any further solutions (steps 11 to 19), and thus the evaluation can be completed at step 20.



```
       :- table a/1, b/1.

c1)    a(X) :- b(X).
c2)    a(2).
c3)    b(X) :- a(X).
c4)    b(1).
```

Call	Solutions
1: a(X)	4: X=2 8: X=1 20: complete
2: b(X)	5: X=2 7: X=1 20: complete

Fig. 3. A linear tabled evaluation using batched scheduling with DRE evaluation

4. Propagation of Solutions in Reevaluation Rounds

In the previous sections, one could observe that tabling does not store duplicate solutions in the table space and, instead, repeated solutions fail. This is how tabling avoids unnecessary computations and looping for duplicate solutions. However, since repeated solutions also fail in reevaluation rounds, this means that, in fact, a solution is only propagated once, i.e., in the round it is first found, which might be not sufficient to ensure the completeness of the evaluation. To solve this problem, in a reevaluation round, we start by propagating (consuming) the solutions already available for the subgoal call at hand. Alternatively, we could propagate the solutions at the end, after the fix-point check procedure, but by doing that some solutions will be propagated more than once in the same round, which is worthless.

In the previous examples, for simplicity of explanation, we have omitted some steps regarding the propagation of solutions in the leader call since, for all the examples, one propagation per solution was enough to correctly compute the corresponding evaluations. To better illustrate the importance of the propagation of solutions in reevaluation rounds and, in particular, for the leader call, Fig. 4 shows a new example, using again the same program from Fig. 1, but for the query goal *a(X1), b(X2)*. For simplicity of explanation, we consider a standard linear tabled evaluation, i.e., without DRA and DRE support. In order to have a common representation of variables between the program code, the evaluation and the table space, the different calls to both *a/1* and *b/1* are presented using a generic variable *X*, instead of the *real* variables *X1* and *X2*.

In the first round of the evaluation (steps 1 to 12), the solutions found for *a(X)*, at steps 5 and 9, are propagated to the context of *a(X1)* and, in the continuation, *b(X2)* consumes (note that at this point *b(X2)* is a repeated call

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

Call	Solutions
1: a(X)	5: X=1 9: X=2 : complete
2: b(X)	4: X=1 17: X=2 : complete

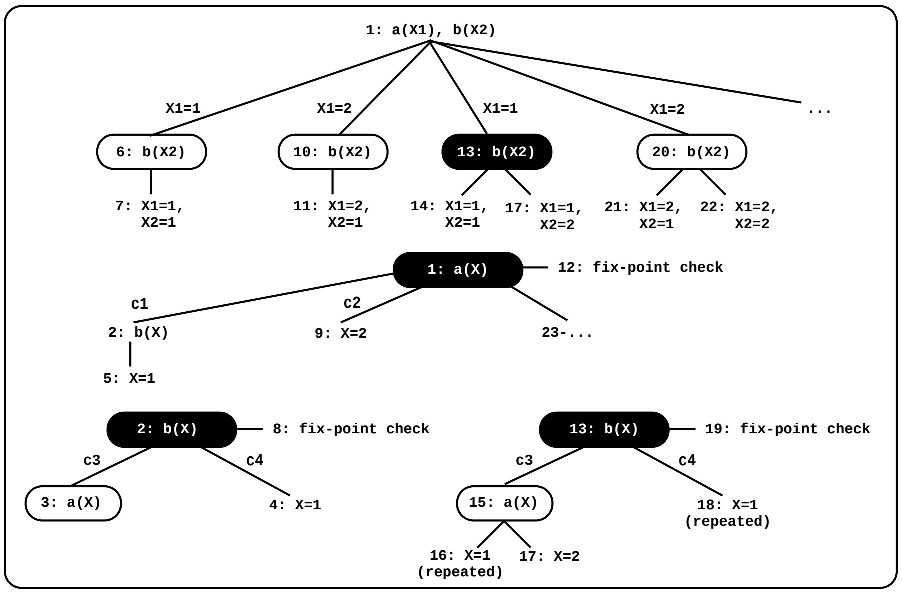


Fig. 4. Propagation of solutions in reevaluation rounds using batched scheduling

to *b(X)*) the available solution found at step 4, which originates the solutions {*X1=1, X2=1*} (step 7) and {*X1=2, X2=1*} (step 11) for the top query goal.

Next, in the second round of the evaluation, the leader call starts by propagating its first solution, calling *b(X2)* in the continuation (step 13). Since this is the first call to *b(X)* in this round, *b(X2)* also starts by propagating its current solution (step 14). Then, when reevaluating the program clauses for *b(X)* (steps 15 to 18), a new solution {*X=2*} is found (step 17). The combination of this new solution with the previous solutions for *a(X1)* originates two new solutions, {*X1=1, X2=2*} (step 17) and {*X1=2, X2=2*} (step 22), for the top query goal.

Notice that without the propagation of solutions for the leader call (steps 13 and 20), no further solutions had been found for the top query goal. In particular, the solution {*X=2*} for *b(X)* would have been found in the context of *a(X)* (similarly to the solution {*X=1*} found at step 4) but, since this originates a repeated solution for *a(X)*, the computation will fail. By failing for *a(X)*, we cannot combine the new solution for *b(X)* with the previous solutions for *a(X1)* at the top query goal. Hence, this fact, i.e., the fact that tabling fails for repeated solutions, can lead to a collateral effect where it can be blocking forward execution. To solve this problem, in a reevaluation round, we start by propagating all the available solutions.

5. Implementation Details

This section describes the implementation details regarding the extension of our framework to support batched scheduling, with particular focus on the table space data structures and on the tabling operations.

5.1. Table Space

To implement the table space, Yap uses *tries* which is considered a very efficient data structure to implement the table space [10]. Tries are trees in which common prefixes are represented only once. Tries provide complete discrimination for terms and permit look up and insertion to be done in a single pass.

In more detail, a trie is a tree structure where each different path through the *trie nodes* corresponds to a term described by the tokens labeling the traversed nodes. For example, the tokenized form of the term *p(X,1,f(Y))* is the sequence of 5 tokens *p/3*, VAR_0, 1, *f/1* and VAR_1, where each variable is represented as a distinct VAR_i constant [3]. Two terms with common prefixes will branch off from each other at the first distinguishing token. Consider, for example, a second term *p(Z,1,b)*. Since the main functor, token *p/3*, and the first two arguments, tokens VAR_0 and 1, are common to both terms, only one additional node will be required to fully represent this second term in the trie, thus allowing to save three trie nodes in this case.

As other tabling engines, Yap uses two levels of tries: one for the subgoal calls and other for the computed solutions. A tabled predicate accesses the table space through a specific *table entry* data structure. Each different subgoal call is represented as a unique path in the *subgoal trie* and each different solution is represented as a unique path in the *solution trie*. Contrary to subgoal tries, solution trie paths hold just the substitution terms for the free variables that exist in the argument terms of the corresponding subgoal call [10]. An example for a tabled predicate *p/3* is shown in Fig. 5.

Initially, the table entry for *p/3* points to an empty subgoal trie. Then, the subgoal *p(X,1,Y)* is called and three trie nodes are inserted to represent the arguments in the call: one for variable *X* (VAR_0), a second for integer 1, and a last one for variable *Y* (VAR_1). Since the predicate's functor term is already represented by its table entry, we can avoid inserting an explicit node for *p/3* in the subgoal trie. Then, the leaf node is set to point to a subgoal frame, from where the answers for the call will be stored. The example shows two answers for *p(X,1,Y)*: $\{X=VAR_0,\ Y=f(VAR_1)\}$ and $\{X=VAR_0,\ Y=b\}$. Since both answers have the same substitution term for argument *X*, they share the top node in the answer trie (VAR_0). For argument *Y*, each answer has a different substitution term and, thus, a different path is used to represent each.

When adding answers, the leaf nodes are chained in a linked list in insertion time order, so that the recovery may happen the same way. In Fig. 5, we can observe that the leaf node for the first answer (node VAR_1) points (dashed arrow) to the leaf node of the second answer (node *b*). To maintain this list, two fields in the subgoal frame data structure point, respectively, to the first and last answer

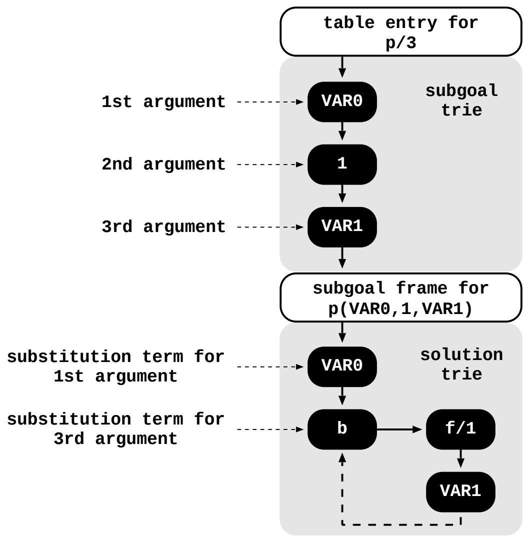


Fig. 5. Table space organization

of this list (for simplicity of illustration, these pointers are not shown in Fig. 5). When consuming answers, a consumer node only needs to keep a pointer to the leaf node of its last loaded answer, and consumes more answers just by following the chain. Answers are loaded by traversing the trie nodes bottom-up (again, for simplicity of illustration, such pointers are not shown in Fig. 5).

A key data structure in this organization is the *subgoal frame*. Subgoal frames are used to store information about each tabled subgoal call, namely: the entry point to the solution trie; the state of the subgoal (*ready*, *evaluating* or *complete*); support to detect if the subgoal is a leader call; and support to detect if new solutions were found during the last round of evaluation. The DRA and DRE strategies extend the subgoal frame data structure with the following extra information [2]: support to detect, store and load looping alternatives; two new states used to detect generator and consumer calls in reevaluating rounds (*loop_ready* and *loop_evaluating*); the pioneer call; and the backtracking clause of the former call. In more detail, the most relevant subgoal frame fields in our implementation are:

***SgFr_dfn*:** is the *depth-first number* of the call. Calls are numbered incrementally and according to the order in which they appear in the evaluation.

***SgFr_state*:** indicates the state of the subgoal. A subgoal can be in one of the following states: *ready*, *evaluating*, *loop_ready*, *loop_evaluating* or *complete*.

***SgFr_is_leader*:** indicates if the call is a leader call or not. New calls are by default leader calls.

***SgFr_prev_on_scc*:** points to the subgoal frame corresponding to the previous call in evaluation (i.e., with *SgFr_state* as *evaluating* or *loop_evaluating*) in the current SCC. It is used by the leader call to traverse the subgoal frames in order to mark them for reevaluation or as completed. A global variable *TOP_SCC* always points to the youngest subgoal frame in evaluation in the current SCC.

***SgFr_prev_on_branch*:** points to the subgoal frame corresponding to the previous call in the current branch that is in the first round (i.e., with *SgFr_state* as *evaluating*) or that is a leader call. It is used to traverse the subgoal frames in order to detect looping alternatives and to detect non-leader calls. A global variable *TOP_BRANCH* always points to the youngest subgoal frame in the current branch.

SgFr_new_solutions: indicates if new solutions were found during the execution of the current round.

***SgFr_first_solution*:** points to the leaf trie node corresponding to the first available solution.

***SgFr_last_solution*:** points to the leaf trie node corresponding to the last available solution.

***SgFr_last_consumed*:** marks the last solution consumed in a generator (pioneer or follower) call (supports the propagation of solutions, as discussed in section 4).

5.2. Tabling Operations

We next introduce the pseudo-code for the main tabling operations required to support batched scheduling with DRA and DRE evaluation.

We start with Algorithm 1 showing the pseudo-code for the new solution operation. Initially, the operation simply inserts the given solution *SOL* in the solution trie structure for the given subgoal frame *SF* (line 1) and, if the solution is new, it updates the *SgFr_new_solutions* subgoal frame field to *TRUE* (line 2) and proceeds with forward execution as usual. Otherwise, the solution is repeated and execution fails (line 4).

Algorithm 1 $new\_solution(solution\ SOL,\ subgoal\ frame\ SF)$

```
1: if $solution\_check\_insert(SOL, SF)$ = true then {new solution}
2:     $SgFr\_new\_solutions(SF) \leftarrow$ true
3: else
4:     $fail()$
```

Next, in Algorithm 2, we show the pseudo-code for the tabled call operation. Initially, the operation starts by inserting the given subgoal call *SC* in the subgoal trie structure, from where a subgoal frame *SF*, representing the given call, is obtained (line 1). New calls to tabled subgoals are inserted into the table space by allocating the necessary data structures, which includes a new subgoal frame (this is the case where the state of *SF* starts to be *ready*). In such case, the tabled call operation then stores a new generator node[3] (line 3); updates the state of *SF* to *evaluating* (line 4); defines a new SCC (lines 5-6); adds *SF* to the current branch (lines 7-8); and proceeds by executing the current alternative (line 9).

Algorithm 2 $tabled\_call(subgoal\ call\ SC)$

```
 1: $SF \leftarrow subgoal\_check\_insert(SC)$ {SF is the subgoal frame for the subgoal call SC}
 2: if $SgFr\_state(SF) = ready$ then {new call}
 3:    $store\_generator\_node()$
 4:    $SgFr\_state(SF) \leftarrow evaluating$
 5:    $SgFr\_prev\_on\_scc(SF) \leftarrow TOP\_SCC$ {new SCC}
 6:    $TOP\_SCC \leftarrow SF$
 7:    $SgFr\_prev\_on\_branch(SF) \leftarrow TOP\_BRANCH$ {add to current branch}
 8:    $TOP\_BRANCH \leftarrow SF$
 9:    goto $evaluate(current\_alternative())$
10: else if $SgFr\_state(SF) = complete$ then {already evaluated}
11:    goto $completed\_table\_optimization(SF)$
12: else if $SgFr\_state(SF) = loop\_ready$ then {first call in reevaluation round}
13:    $store\_generator\_node()$
14:    $SgFr\_state(SF) \leftarrow loop\_evaluating$
15:    $SgFr\_prev\_on\_scc(SF) \leftarrow TOP\_SCC$ {new SCC}
16:    $TOP\_SCC \leftarrow SF$
17:    $SgFr\_last\_consumed(SF) \leftarrow SgFr\_first\_solution(SF)$
18:    if $DRA\_mode(SF)$ then
19:       goto $consume\_solutions\_and\_reevaluate(SF, first\_looping\_alternative())$
20:    else
21:       goto $consume\_solutions\_and\_reevaluate(SF, first\_alternative())$
22: else if $SgFr\_state(SF) = evaluating$ or $SgFr\_state(SF) = loop\_evaluating$ then
23:    $mark\_current\_branch(SF)$
24:    if $DRE\_mode(SF)$ and $has\_unexploited\_alternatives(SF)$ then
25:       $store\_follower\_node()$
26:       if $DRA\_mode(SF)$ and $SgFr\_state(SF) = loop\_evaluating$ then
27:          goto $consume\_solutions\_and\_reevaluate(SF, next\_looping\_alternative())$
28:       else
29:          goto $consume\_solutions\_and\_reevaluate(SF, next\_alternative())$
30:    else
31:       $store\_consumer\_node()$
32:       goto $consume\_solutions(SF)$
```

On the other hand, if the subgoal call is a repeated call, then the subgoal frame *SF* is already in the table space, and three different situations may occur. First, if the call is already evaluated (this is the case where the state of *SF* is *complete*), the operation consumes the available solutions by implementing the

[3] Generator, consumer and follower nodes are implemented as regular choice points extended with some extra fields related to the table space data structures.

completed table optimization [10] which executes compiled code directly from the solution trie structure associated with the completed call (line 11).

Second, if the call is a first call in a reevaluating round (this is the case where the state of *SF* is *loop_ready*), the operation stores a new generator node (line 13); updates the state of *SF* to *loop_evaluating* (line 14); defines a new SCC (lines 15-16); and resets the *SgFr_last_consumed* field to the first solution (line 17). Then, it executes the *consume_solutions_and_reevaluate()* procedure in order to consume the available solutions before reevaluate the matching alternatives. This procedure, consumes all the available solutions for the subgoal, starting from the first solution, and, when no more solutions are to be consumed, it starts with the evaluation of the first matching alternative, which for DRA is the first looping alternative (lines 18-21).

Third, if the call is a repeated call (this is the case where the state of *SF* is *evaluating* or *loop_evaluating*), the operation first calls the *mark_current_branch()* procedure (please see Algorithm 3 next) in order to mark the current branch as a non-leader branch and, if in DRA mode, also mark the current branch as a looping branch (line 23). Next, if DRE mode is enabled and there are unexploited alternatives (i.e., there is a backtracking clause for the former call), it stores a follower node (line 25) and proceeds by consuming the available solutions before executing the next looping alternative or the next matching alternative, according to whether the DRA mode is enabled or disabled for the subgoal (lines 26-29). Otherwise, it stores a new consumer node and starts consuming the available solutions (lines 31-32).

Algorithm 3 shows the details for the *mark_current_branch()* procedure. To mark the current branch as a non-leader branch and, if in DRA mode, as a looping branch, we follow the *TOP_BRANCH* chain and for all intermediate generator calls in evaluation up to the generator call for *SF*, we mark them as non-leader calls (note that the call at hand defines a new dependency for the current SCC) and we mark the alternatives being evaluated by each call as looping alternatives.

Algorithm 3 $mark\_current\_branch(subgoal\ frame\ SF)$

```
1: $aux\_sf \leftarrow TOP\_BRANCH$
2: while $SgFr\_dfn(aux\_sf) > SgFr\_dfn(SF)$ do
3:     $SgFr\_is\_leader(aux\_sf) \leftarrow$ false
4:     if $DRA\_mode(aux\_sf)$ then
5:         $mark\_current\_alternative\_as\_looping\_alternative(aux\_sf)$
6:     $aux\_sf \leftarrow SgFr\_prev\_on\_branch(aux\_sf)$
7: if $DRA\_mode(aux\_sf)$ then
8:     $mark\_current\_alternative\_as\_looping\_alternative(aux\_sf)$
```

Finally, we discuss in more detail how completion is detected with batched scheduling. Remember that after exploring the last matching clause for a tabled call, we execute the *fix-point check* operation. Algorithm 4 shows the pseudo-code for its implementation.

Algorithm 4 $fix\_point\_check(subgoal\ frame\ SF)$

```
1: if ($SgFr\_is\_leader(SF)$) then
2:    if $SgFr\_new\_solutions(SF)$ then {start a new round}
3:       for all $SG\ such\ that\ SG\ in\ current\ SCC$ do
4:          $SgFr\_state(SG) \leftarrow loop\_ready$
5:       $SgFr\_state(SF) \leftarrow loop\_evaluating$
6:       $TOP\_SCC \leftarrow SF$
7:       $SgFr\_new\_solutions(SF) \leftarrow$ false
8:       $SgFr\_last\_consumed(SF) \leftarrow SgFr\_first\_solution(SF)$
9:       if $DRA\_mode(SF)$ then
10:         goto $consume\_solutions\_and\_reevaluate(SF, first\_looping\_alternative())$
11:      else
12:         goto $consume\_solutions\_and\_reevaluate(SF, first\_alternative())$
13:   else {reached a fix-point}
14:      for all $SG\ such\ that\ SG\ in\ current\ SCC$ do {complete all subgoals in SCC}
15:         $SgFr\_state(SG) \leftarrow complete$
16:      $TOP\_SCC \leftarrow SgFr\_prev\_on\_scc(SF)$
17:      $fail()$
18: else {not a leader call}
19:   if $SgFr\_state(SF) = evaluating$ then {first round}
20:      $TOP\_BRANCH \leftarrow SgFr\_prev\_on\_branch(SF)$
21:   if $SgFr\_new\_solutions(SF)$ then {propagate new solutions}
22:      $SgFr\_new\_solutions(current\_leader(SF)) \leftarrow$ true
23:   $SgFr\_new\_solutions(SF) \leftarrow$ false
24:   $fail()$
```

The fix-point check operation starts by verifying if the subgoal at hand is a leader call. If it is leader and has found new solutions during the last round, then the current SCC is scheduled for a reevaluation round (lines 3-12). This includes updating the state for all subgoals in the current SCC, updating the *TOP_SCC* variable to the current subgoal frame and resetting the *SgFr_new_solutions* field to *FALSE* (lines 3-7). Then, as for a first call in a reevaluating round in the *tabled call* operation, it also resets the *SgFr_last_consumed* field to the first solution (line 8) and executes the *consume_solutions_and_reevaluate()* procedure (lines 9-12).

On the other hand, if the subgoal is leader but no new solutions were found during the current round, then we have reached a fix-point. All subgoals in the current SCC are thus marked as completed, the *TOP_SCC* variable is updated to the next subgoal frame and the evaluation fails (lines 14-17).

Otherwise, the subgoal is not a leader call. Then it removes itself from the *TOP_BRANCH* chain (lines 19-20), propagates the new solutions information to the current leader of the SCC (lines 21-22), resets the *SgFr_new_solutions* field to *FALSE* (line 23) and then fails (line 24). Note that, with batched scheduling, we can safely fail since all the solutions were already propagated to the context of the calling environment. Moreover, since the *SgFr_new_solutions* flag is propagated to the leader of the SCC, the leader will mark the SCC for a reevaluation round, which means that the current subgoal will be called again, and so it will start by consuming its solutions.

As an optimization, a non-leader call *C* executing the fix-point check operation can be removed beforehand from the *TOP_BRANCH* chain (lines 19-20 in Algorithm 4) since we already know that it is a non-leader call and have marked its looping alternatives. Thus, when we execute the *mark_current_branch()* procedure in a reevaluation round for a call *C*, then *C* might have been removed from the chain in a previous fix-point check operation. This is the reason why we need to follow the subgoal frames in the *TOP_BRANCH* chain up to the first subgoal frame with a smaller *SgFr_dfn* value than *C* (while loop on Algorithm 3).

6. Experimental Results

To the best of our knowledge, Yap is now the first tabling engine that integrates and supports the combination of different linear tabling strategies using batched scheduling. We have thus the conditions to better understand the advantages and weaknesses of each strategy when used solely or combined. In what follows, we present experimental results comparing linear tabled evaluation with and without DRA and DRE support, using batched scheduling. To put our results in perspective, we have also included experiments for the B-Prolog linear tabling system [15] and for the YapTab suspension-based tabling system [12], both using batched scheduling. In fact, for B-Prolog, we used its *eager scheduling mode*, which is similar to batched scheduling. The environment for our experiments was a PC with a 2.83 GHz Intel(R) Core(TM)2 Quad CPU and 8 GBytes of memory running the Linux kernel 3.0.0-16-generic. We used B-Prolog version 7.5 and Yap version 6.0.7[4].

For benchmarking, we used three sets of programs. The **Model Checking** set includes three different specifications and transition relation graphs usually used in model checking applications: **IProto**, the transition relation graph for the i-protocol specification defined for a correct version (fix) with a huge window size (w = 2); **Leader**, the transition relation graph for the leader election specification defined for 5 processes; and **Sieve**, the transition relation graph for the sieve specification defined for 5 processes and 4 overflow prime numbers. The **Path Right** set implements the right recursive definition of the well-known $path/2$ predicate, that computes the transitive closure in a graph, using three different edge configurations. Figure 6 shows an example for each configuration. We experimented the **Pyramid** and **Cycle** configurations with depths 1000, 2000 and 3000 and the **Grid** configuration with depths 20, 30 and 40. We chose this set of experiments because the $path/2$ predicate implements a relatively easy to

[4] Source code available from http://cracs.fc.up.pt/node/5121

understand pattern of computation and its right recursive definition creates several inter-dependencies between tabled subgoals. The **Warren** set is a variation of the left recursive definition of the path problem for a linear graph (see Fig. 6), where the *path/2* clauses are duplicated to be used with the labels *a* and *b*. This problem was kindly suggested by David S. Warren as a way to stress the performance of a linear tabling system. All benchmarks find all the solutions for the problem.

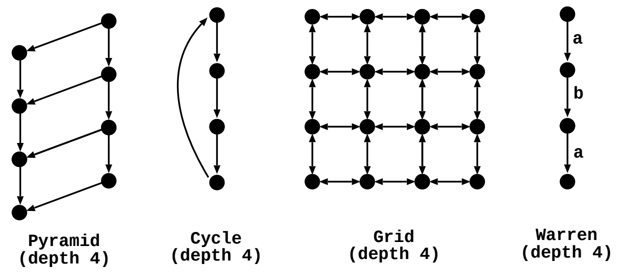


Fig. 6. Edge configurations used with the second and third set of problems

In Table 1, we show the execution time, in milliseconds, for standard linear tabling (column **Std**) and the respective execution time ratios for DRA and DRE evaluation, solely and combined (column **DRA+DRE**), B-Prolog and YapTab, using batched scheduling, for the **Model Checking**, **Path Right** and **Warren** sets of problems. Ratios higher than 1.00 mean that the respective strategy has a positive impact on the execution time, when compared with standard linear tabling. The ratio marked with $n.c.$ for B-Prolog means that we are *not considering* it in the average results (for some reason, we failed in executing this benchmark). The results are the average of five runs for each benchmark.

In addition to the results presented in Table 1, we also collected several statistics regarding important aspects of the evaluation. In Table 2, we show some of these statistics for standard linear tabling and the respective performance ratios when compared with the other models, for a subset of the benchmarks. We used the **Leader** specification for the **Model Checking** set, the configurations **Pyramid** and **Cycle** with depth 2000 and **Grid** with depth 30 for the **Path Right** set, and the configuration with depth 600 for the **Warren** set.

The statistics in Table 2 measure how the mixing with SLD (non-tabled) computations can affect the base performance of our benchmarks. For that, we extended the tabled predicates, at the beginning and at the end of each clause, with dummy SLD (non-tabled) predicates, which we named *sldi/0*, with $0 < i \leq 2n$, where n is the number of clauses defining the tabled predicate. For example, the extended definition for the *path/2* predicate is:

```
path(X,Z) :- sld1, edge(X,Y), path(Y,Z), sld2.
path(X,Z) :- sld3, edge(X,Z), sld4.
```

Table 1. Execution time, in milliseconds, for standard linear tabling and the respective execution time ratios for DRA and DRE evaluation, solely and combined, B-Prolog and YapTab, using batched scheduling (for the linear tabling models, best ratios are in bold)

Benchmark	Std	DRA	DRE	DRA+DRE	B-Prolog	YapTab
Model Checking						
IProto	2,874	**1.00**	0.50	0.93	0.36	2.39
Leader	5,355	**1.01**	0.40	0.99	0.13	2.83
Sieve	35,218	**1.00**	0.46	0.93	0.16	3.19
Average ratio		1.00	0.45	0.95	0.22	2.80
Path Right - Pyramid						
1000	983	**1.87**	0.89	1.49	1.04	1.90
2000	3,897	**1.88**	0.89	1.49	0.69	1.94
3000	9,043	**1.91**	0.89	1.53	$n.c.$	1.98
Path Right - Cycle						
1000	687	**1.27**	0.96	1.22	**1.27**	1.89
2000	2,793	**1.27**	0.97	1.22	0.91	1.82
3000	6,048	**1.29**	0.95	1.22	0.70	2.05
Path Right - Grid						
20	221	**1.33**	0.97	1.27	1.09	2.10
30	1,344	**1.32**	0.99	1.30	1.02	2.22
40	4,578	**1.31**	0.97	1.26	0.76	2.34
Average ratio		**1.50**	0.94	1.33	0.93	2.03
Warren						
400	2,673	1.02	**64.26**	**64.26**	0.34	126.09
600	9,496	0.99	**87.28**	**87.28**	0.35	162.61
800	23,163	1.00	112.88	**116.98**	0.35	216.88
Average ratio		1.00	**87.93**	**89.51**	0.35	168.53

The rows in Table 2 show the number of times each dummy SLD predicate is called for the corresponding benchmark. We can read these numbers as an estimation of the performance ratios that we will obtain if the execution time of the corresponding SLD predicate clearly overweights the execution time of the other computations. Note that the odd SLD predicates (such as **sld1** and **sld3**) correspond to re-executions of a clause and that the even SLD predicates (such as **sld2** and **sld4**) correspond to new solution operations. In our experiments, the **sld2** predicate (placed at the end of the first tabled clause) is the one that can potentially have a greater influence in the performance ratios as it clearly exceeds all the others in the number of times it is called (see Table 2).

7. Discussion

Analyzing the general picture of Table 1, the results show that DRA evaluation is able to reduce the execution time for the **Path Right** problem set (1.50 times faster, on average) but has no impact for the other two sets, when compared with standard evaluation. The results also indicate that DRE evaluation has a negative impact in the execution time for the **Model Checking** and **Path Right**

Table 2. Number of calls to the dummy SLD predicates for standard linear tabling and the respective ratios for DRA and DRE evaluation, solely and combined, B-Prolog and YapTab, using batched scheduling (for the linear tabling models, best ratios are in bold)

Benchmark	Std	DRA	DRE	DRA+DRE	B-Prolog	YapTab
Model Checking - Leader						
sld1	3	1.00	0.75	1.00	1.00	3.00
sld2	1,153,026	1.00	0.40	1.00	1.00	2.00
sld3	3	**3.00**	0.75	**3.00**	**3.00**	3.00
sld4	3	**3.00**	0.75	**3.00**	**3.00**	3.00
Path Right - Pyramid 2000						
sld1	7,999	**2.00**	1.00	**2.00**	**2.00**	2.00
sld2	37,951,017	**2.38**	0.86	1.73	**2.38**	2.38
sld3	7,999	**2.00**	1.00	**2.00**	**2.00**	2.00
sld4	23,988	**2.00**	1.00	**2.00**	**2.00**	2.00
Path Right - Cycle 2000						
sld1	6,002	1.00	1.00	1.00	1.00	3.00
sld2	18,003,000	**1.29**	1.00	**1.29**	**1.29**	2.25
sld3	6,002	**3.00**	1.00	**3.00**	**3.00**	3.00
sld4	10,000	**2.50**	1.00	**2.50**	**2.50**	2.50
Path Right - Grid 30						
sld1	2,702	1.00	1.00	1.00	0.18	3.00
sld2	13,851,534	1.29	1.00	**1.30**	0.30	2.21
sld3	2,702	**3.00**	1.00	1.02	**3.00**	3.00
sld4	17,400	**2.50**	1.00	1.27	**2.50**	2.50
Warren - 600						
sld1/sld3	302	1.00	**100.67**	**100.67**	1.00	302.00
sld2/sld4	18,044,650	1.00	66.98	**100.42**	1.00	201.17
sld5/sld7	302	**302.00**	100.67	**302.00**	**302.00**	302.00
sld6/sld8	90,600	**302.00**	100.67	**302.00**	**302.00**	302.00

sets but, on the other hand, it can significantly reduce the execution time for the **Warren** set (more than 80 times faster, on average). We next discuss in more detail each strategy.

DRA: the results for DRA evaluation show that the strategy of avoiding the exploration of non-looping alternatives in reevaluation rounds is quite effective in general and does not add extra overheads when not used. The results also show that, for the **Path Right** set, DRA is more effective for programs without loops, like the **Pyramid** configurations, than for programs with larger SCCs, like the **Cycle** and **Grid** configurations. On Table 2, we can observe that the number of dummy SLD computations is, in fact, effectively reduced with DRA evaluation.

DRE: for the **Model Checking** set, DRE evaluation is around two times slower than standard evaluation and, for the **Path Right** set, DRE has no significant impact for all the configurations. Table 2 confirms that, the strategy of allocating follower nodes, adds an extra complexity to the evaluation for the **Model Checking** set (the number of dummy SLD calls is higher) and that it has no impact for the **Path Right** set (the number of dummy SLD calls is identical to standard evaluation). For the **Warren** set, DRE evaluation produces the most interesting results. Note that, this is the set of benchmarks where suspension-based tabling (the YapTab system) is far more faster than standard linear tabling (168.53 times faster, on average) and the difference increases as the depth of the problem also increases. However, DRE evaluation is able to reduce this huge difference to a minimum. On average, DRE evaluation is 87.93 times faster than standard evaluation and the scalability, as the depth of the problem increases, is similar to the one observed for YapTab. Table 2 confirms this behavior for DRE and YapTab evaluations (the number of dummy SLD calls is clearly lower than standard evaluation).

Regarding the combination of both strategies (DRA+DRE), our experiments show that, in general, the best of both worlds is always present in the combination. The results in Table 1 show that, by combining both strategies, DRA is able to avoid DRE behavior for the **Model Checking** and **Path Right** sets. Still, the results for DRA+DRE are slightly worst than DRA used solely. For the **Warren** set, the results show that, by combining both strategies, it is possible to reduce even further the execution time when compared with DRE used solely. In particular, one can observe that, for depths 400 and 600, the execution times are the same but, for depth 800, DRA+DRE evaluation outperforms DRE used solely.

The statistics on Table 2 confirm that, in general, the best of both worlds is always present in the combination. The exceptions are the **sld2** predicate, for the **Pyramid 2000** configuration, and the **sld3** and **sld4** predicates, for the **Grid 30** configuration. On the other hand, for the **Warren 600** configuration, the **sld1**/**sld3** predicates are executed the same number of times as for DRE used solely, the **sld5** to **sld8** predicates are executed the same number of times as for DRA used solely, and the **sld2** and **sld4** predicates are executed less times than both strategies used solely, which is explained by the fact that the fix-point is achieved in less rounds (statistics not shown here).

Regarding the comparison with the B-Prolog linear tabling system, the results in Table 2 suggest that B-Prolog implements a DRA-based evaluation strategy since the statistics for B-Prolog and DRA evaluation are all the same, except for the **sld1** and **sld2** predicates in the **Grid 30** configuration. However, the execution times in Table 1 show that our DRA implementation is always faster than B-Prolog in these experiments and that, for almost all configurations, the ratio difference shows a generic tendency to increase as the depth of the problem also increases.

For all experiments, the results obtained for the YapTab suspension-based system clearly outperform the standard linear tabled evaluation but, for our DRA+DRE implementation, they are globally comparable. On average, YapTab is around 2 times faster than DRA+DRE evaluation, including the **Warren** problem set, where YapTab shows a huge difference for standard linear tabling. The results also indicate that our implementation scales as well as YapTab when we increase the depth of the problem being tested.

8. Conclusions

We have presented a new linear tabling framework that integrates and supports batched scheduling with DRA and DRE evaluation, solely or combined. We discussed how these strategies can optimize different aspects of a tabled evaluation and we presented the relevant implementation details for their integration on top of the Yap system.

Our experimental results were very interesting and very promising. In particular, the combination of DRA with DRE showed the potential of our framework to effectively reduce the execution time of the standard linear tabled evaluation. When compared with YapTab's suspension-based mechanism, the commonly referred weakness of linear tabling of doing a huge number of redundant computations for computing fix-points was not such a problem in our experiments. We thus argue that an efficient implementation of linear tabling can be a good and first alternative to incorporate tabling into a Prolog system without such support.

Further work will include adding new strategies/optimizations to our framework, and exploring the impact of applying our strategies to more complex problems, seeking real-world experimental results, allowing us to improve and consolidate our current implementation.

Acknowledgments. This work is partially funded by the ERDF (European Regional Development Fund) through the COMPETE Programme and by FCT (Portuguese Foundation for Science and Technology) within projects LEAP (FCOMP-01-0124-FEDER-015008) and PEst (FCOMP-01-0124-FEDER-022701). Miguel Areias is funded by the FCT grant SFRH/BD/69673/2010.

References

1. Areias, M., Rocha, R.: An Efficient Implementation of Linear Tabling Based on Dynamic Reordering of Alternatives. In: International Symposium on Practical Aspects of Declarative Languages. pp. 279–293. No. 5937 in LNCS, Springer-Verlag (2010)
2. Areias, M., Rocha, R.: On Combining Linear-Based Strategies for Tabled Evaluation of Logic Programs. Journal of Theory and Practice of Logic Programming, International Conference on Logic Programming, Special Issue 11(4–5), 681–696 (2011)
3. Bachmair, L., Chen, T., Ramakrishnan, I.V.: Associative Commutative Discrimination Nets. In: International Joint Conference on Theory and Practice of Software Development. pp. 61–74. No. 668 in LNCS, Springer-Verlag (1993)
4. Chen, W., Warren, D.S.: Tabled Evaluation with Delaying for General Logic Programs. Journal of the ACM 43(1), 20–74 (1996)
5. Chico, P., Carro, M., Hermenegildo, M.V., Silva, C., Rocha, R.: An Improved Continuation Call-Based Implementation of Tabling. In: International Symposium on Practical Aspects of Declarative Languages. pp. 197–213. No. 4902 in LNCS, Springer-Verlag (2008)
6. Cruz, F., Rocha, R.: Retroactive Subsumption-Based Tabled Evaluation of Logic Programs. In: European Conference on Logics in Artificial Intelligence. pp. 130–142. No. 6341 in LNAI, Springer-Verlag (2010)

7. Freire, J., Swift, T., Warren, D.S.: Beyond Depth-First: Improving Tabled Logic Programs through Alternative Scheduling Strategies. In: International Symposium on Programming Language Implementation and Logic Programming. pp. 243–258. No. 1140 in LNCS, Springer-Verlag (1996)
8. Guo, H.F., Gupta, G.: A Simple Scheme for Implementing Tabled Logic Programming Systems Based on Dynamic Reordering of Alternatives. In: International Conference on Logic Programming. pp. 181–196. No. 2237 in LNCS, Springer-Verlag (2001)
9. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag (1987)
10. Ramakrishnan, I.V., Rao, P., Sagonas, K., Swift, T., Warren, D.S.: Efficient Access Mechanisms for Tabled Logic Programs. Journal of Logic Programming 38(1), 31–54 (1999)
11. Sagonas, K., Swift, T.: An Abstract Machine for Tabled Execution of Fixed-Order Stratified Logic Programs. ACM Transactions on Programming Languages and Systems 20(3), 586–634 (1998)
12. Santos Costa, V., Rocha, R., Damas, L.: The YAP Prolog System. Journal of Theory and Practice of Logic Programming 12(1 & 2), 5–34 (2012)
13. Somogyi, Z., Sagonas, K.: Tabling in Mercury: Design and Implementation. In: International Symposium on Practical Aspects of Declarative Languages. pp. 150–167. No. 3819 in LNCS, Springer-Verlag (2006)
14. Swift, T., Warren, D.S.: XSB: Extending Prolog with Tabled Logic Programming. Theory and Practice of Logic Programming 12(1 & 2), 157–187 (2012)
15. Zhou, N.F.: The Language Features and Architecture of B-Prolog. Journal of Theory and Practice of Logic Programming 12(1 & 2), 189–218 (2012)
16. Zhou, N.F., Shen, Y.D., Yuan, L.Y., You, J.H.: Implementation of a Linear Tabling Mechanism. In: Practical Aspects of Declarative Languages. pp. 109–123. No. 1753 in LNCS, Springer-Verlag (2000)

Miguel Areias received his B.Sc. and M.Sc. degrees in Computer Science from Faculty of Science of the University of Porto, in 2008 and 2010, respectively. He is currently pursuing the Ph.D. degree at the University of Porto. He is a Researcher in the Center for Research in Advanced Computing Systems (CRACS), where he has been since 2008 under the supervision of Prof. Dr. Ricardo Rocha. His research interests lie on Parallelism, Concurrency, Multithreading and Tabling mechanisms applied to Logic Programs.

Ricardo Rocha is an Assistant Professor at the Department of Computer Science, Faculty of Sciences, University of Porto, Portugal and a researcher at the CRACS & INESC-Porto LA research unit. He received his PhD degree in Computer Science from the University of Porto in 2001 and his main research topics are the Design and Implementation of Logic Programming Systems, Tabling in Logic Programming and Parallel and Distributed Computing. Another areas of interest include Inductive Logic Programming, Probabilistic Logic Programming and Deductive Databases. He is also one of the main developers of Yap Prolog system, and in particular of the execution models that support tabling and parallel evaluation. He has published more than 50 refereed papers in journals and international conferences, has supervised 11 MSc students and has leading role in two national projects: project STAMPA, funded with 150,000 Euros, and project LEAP, funded with 115,000 Euros. Currently, he also serves the ALP Newsletter as area co-editor for the topic on Implementation.

Received: November 29, 2012; Accepted: August 12, 2013.

DOI: 10.2298/CSIS130224072R

Multi-Agent Systems' Asset for Smart Grid Applications

Gregor Rohbogner[1], Ulf J.J. Hahnel[1], Pascal Benoit[1], and Simon Fey[1,2]

[1] Fraunhofer Institute for Solar Energy Systems ISE, Division Electrical Energy Systems, Heidenhofstraße 2
79110 Freiburg, Germany
{Gregor.Rohbogner, Ulf.Hahnel, Pascal.Benoit}@ise.fraunhofer.de
[2] Offenburg University of Applied Sciences, Badstraße 24
77652 Offenburg, Germany
Simon.Fey@hs-offenburg.de

Abstract. Multi-agent systems are a subject of continuously increasing interest in applied technical sciences. Smart grids are one evolving field of application. Numerous smart grid projects with various interpretations of multi-agent systems as new control concept arose in the last decade. Although several theoretical definitions of the term 'agent' exist, there is a lack of practical understanding that might be improved by clearly distinguishing the agent technologies from other state-of-the-art control technologies. In this paper we clarify the differences between controllers, optimizers, learning systems, and agents. Further, we review most recent smart grid projects, and contrast their interpretations with our understanding of agents and multi-agent systems. We point out that multi-agent systems applied in the smart grid can add value when they are understood as fully distributed networks of control entities embedded in dynamic grid environments; able to operate in a cooperative manner and to automatically (re-)configure themselves.

Keywords: computer science, information systems, multi-agent systems, smart grid, power systems, agent-based control systems

1. Introduction

Agent-based systems have been implemented in the field of technical engineering, having been adopted as new concepts for control systems during the last few decades [25], [28]. Derived from the computer sciences [9], [14], [35] the broad usage of agent technology in the technical domain has resulted in an ambiguous use and interpretation of the notions 'agent' and 'multi-agent systems'; this is especially apparent in current smart grid research projects. The common understanding of the term smart grid in research projects encompasses the development of new power control strategies and communication systems to face the challenges (e.g.

fluctuating power generation, higher dynamics in grid frequency and grid voltage) which arise from the expansion of renewable energies (e.g. photovoltaic systems, wind turbines, and combined heat and power systems) and new electrical loads (e.g. heat pumps and electrical vehicles) [8], [38]. Numerous smart grid projects labeled with the term 'agent' sprouted up in the last few years, exhibiting various interpretation of when and how to apply the term 'agent' [19], [24]. It seems that engineers use the term 'agent' without a common understanding of what it actually embodies.

In the present paper, we aim at providing a clear definition of multi-agent systems in the realm of smart grid distributed control applications[1]. We do so by first identifying the characteristics that a control system should posses to be appropriately labeled as an agent system by emphasizing the differences between agents, optimizers, closed-loop controls, and learning systems. Second, we systematically analyze the interpretations and implementations of multi-agent systems in recent smart grid projects. We then contrast our understanding of agents and multi-agent systems with the existing multi-agent based smart grid projects and discuss which systems can be really understood as such. Finally, we describe the extent to which agent technologies may be of further value in improving smart grid applications, and give directions for future research and practice in the realm of agent-based systems.

2. Smart Grid – Definition and Applications

In accordance with the goals defined by the European Union, central Europe is striving for an energy supply powered by renewable energies [7]. With a rising share of distributed fluctuating renewable energy resources, it will become more and more challenging to ensure a secure and reliable energy supply in the future. This is due to:

Firstly, weather dependent fluctuation of the power supply of renewable energies (e.g. wind turbines or photovoltaic systems) makes the indispensable balancing of power generation and consumption more challenging.

Secondly, the generated renewable electricity is fed mainly into distribution and low-voltage grids. In the past, energy was only consumed in these grids while it was produced by large fossil power plants connected to the transmission and sub-transmission grids (see Fig. 1). Thus, the recent primary grid infrastructure (e.g. local transformers, circuit breakers, lines) is designed and parameterized for these unidirectional flows. In combination with the rising electricity generation in distribution grids, problems (e.g. over-voltage and over-currents) are to be assumed.

[1] [24] identifies four different power engineering domains where recent multi-agent systems are applied. These are: Protection, Modeling & Simulation, Distributed Control, and Monitoring & Diagnostics

Thirdly, the rising amount of distributed energy generation affects the grid frequency today already. Since the frequency is a grid-wide value, it is also necessary to ensure that the controllers of distributed energy resources are parameterized properly to ensure a stable grid. European standardization committees are currently addressing the 50.2 Hz problem, which occurs with a common switching threshold of 50.2 Hz in current inverters. This parameterization involves the danger of a major disturbance in the main grid when suddenly several gigawatt of generation disconnect in case of an underload (frequency above 50.2 Hz) and cause thereby an overload [17].

Fourthly, new applications and business models such as consumption of self-produced energy, electric vehicles, or the bundling of distributed generation to virtual power plants, will change the current static time-invariant behavior of grids to a more dynamic one.

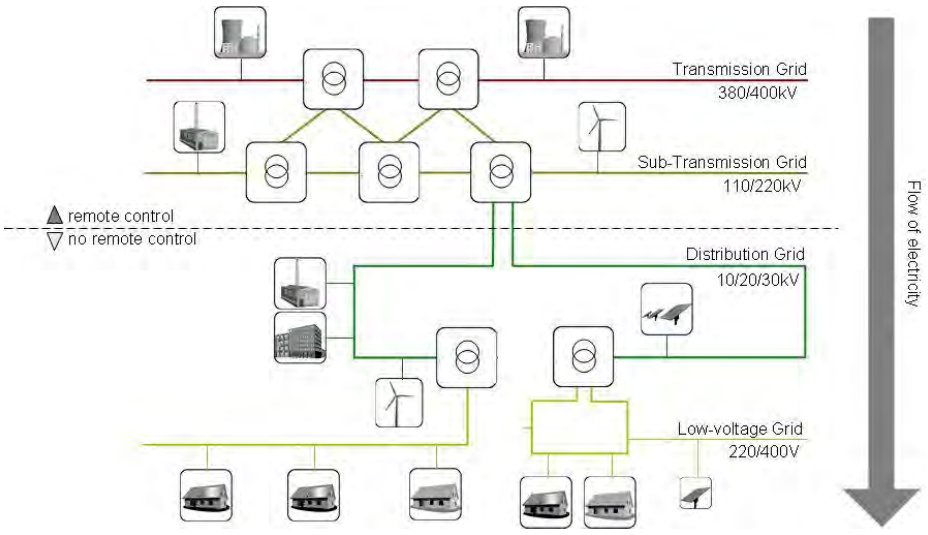


Fig. 1. Structure of today's (centralized) electricity supply system

One approach to handle these new challenges is the extension of recent grids to "Smart Grids". Various interpretations of what the term "Smart Grid" subsumes exist. For example, the European Technology Platform for Smart Grids defines a smart grid as, "an electricity network that can intelligently integrate the actions of all users connected to it - generators, consumers and those that do both – in order to efficiently deliver sustainable, economic and secure electricity supplies" [8]. Slightly variegated, and with an emphasis on the communication systems, the International Electrotechnical Commission (IEC) understands a smart grid as integration of "electrical and information technologies in between any point of generation and any point of consumption" [38]. Thus, the core idea of smart grid is the development of new control strategies and systems – using information and communication technologies – which are necessary, particularly nowadays, due to the injection of renewable and fluctuating energy (e.g. by wind turbines and photovoltaic systems). The complexity of smart grids' applications arises from their interdisciplinary character, requiring the joint work of various research disciplines: electrical engineering, control theory, information technology, jurisprudence, economics and psychology. In this paper we focus mainly on control issues while briefly describing necessary interdisciplinary background.

Numerous different smart grid control systems have been developed to find cost-efficient solutions and to approach the above-mentioned problems. One of the most popular concepts to encourage users or systems to consume power when it is produced by the fluctuating energy resources is known as Demand Side Management (DMS). Mainly based on dynamic pricing, DMS encounters users or devices to redistribute electric demand over a certain period of time [36]. Another example for a smart grid control concept is the pooling of distributed generation and loads that are collectively controlled by a central control entity. These systems are known as virtual power plants [31], [41]. Furthermore, various projects can be found which take the control of so-called 'micro-grids' into consideration. Here, micro-grids are understood as small, local distribution grids containing electric generation and loads, which can be totally separated from, or (re-)connected to, the main distribution grid [30]. Among these examples numerous other smart grid control concepts have been developed. [12] gives a broad overview about recent smart grid projects in Europe.

3. State-of-the-Art versus Agent Technology

In this section we discuss and define an agent from an engineer's perspective. Agents are first defined in an independent application way and then discussed in the context of smart grid applications in section 5. While [9], [14], and [44] discuss the difference between agent technology and various IT domains (e.g. artificial intelligence, web-services and expert systems and grid computing) we analyzed the differences between well established engineering control technologies and agent technology. This section begins with computer science's definition of agents. Based on that, we differentiate in detail between an optimizer, a closed-loop controller, learning systems, and an agent.

While numerous definitions of agents have been discussed in the past, we here post the definition of Jennings and Wooldridge [44], [45]. We consider their approach a good balance between an overly-restrictive and a too-loose definition. A survey of 'agent' definitions can be found in [11]. Jennings and Wooldridge understand an autonomous intelligent agent to be a software artifact which exhibits the following capabilities:

"**Reactivity** Intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives.

Proactiveness Intelligent agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives.

Social ability Intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives."

It seems that these attributes are to some extent also manifested in engineers' state-of-the-art control technologies. To a certain degree this assumption is valid. For the sake of clarity, we work out how an agent can be distinguished from other control technologies, and why that makes sense.

In the rest of the paper we use the term 'agent' instead of 'intelligent agent'. We do so because we believe that labeling a software artifact as 'agent' only offers added value if agents at least constitute a certain degree of intelligence, thus exhibiting the above listed attributes. Hence, the adjective 'intelligent' is dispensable.

3.1. Why Is an Agent More Than Just an Optimizer?

The classic mathematical definition describes optimization as the task of finding a $f \in F$ for which

$$c(f) \leq c(x) \text{ for all } x \in F \qquad (\mathbf{1})$$

while

$$c : F \rightarrow \mathbb{R}$$

Thereby f is the global optimum from the domain of feasible alternatives (feasible points) F and c the objective function [27]. Thus, an optimizer is a software tool which finds the optimal solution f. The optimizer's objective function is formulated once for a specific system with specific constraints. Systems can be of any kind, such as an economic system or a technical system. An example of a technical smart grid system is the cost-optimal operation of a grid-connected Combined Heat and Power Plant (CHP) [13]. Optimizers are not aware of, or in touch with, their system in terms of sensing system behavior or maintaining it like a controller does. This is of paramount importance for reaction, however. Thus, the optimizer example given above calculates the optimal set points for the CHP's operation once, and delivers these set points to a controller which is then responsible for reacting to system changes. Although we consider optimizers as non-reactive entities, it could be said that optimizers are to some extent goal-oriented and therefore proactively. Even if they try to find the best solution out of many, they are not acting proactive in terms of taking the initiative. Normally, optimizers do not automatically adapt their objective function when the system's behavior changes. Thus, the program will fail to reach its goal (i.e. finding a valid optimum). But proactive self-configuration, as we shall see in section 5, is essential for smart grid control systems.

Furthermore, optimizers are neither programmed to converse with a human, nor with other computer programs, hence they exhibit no social behavior.

3.2. Why Is an Agent More Than Just a Digital Closed-Loop Control?

Regarding the core definition of closed-loop controllers, it seems they are fairly similar to an agent. Indeed, closed-loop controllers exhibit a reactive behavior through their feedback loop. Figure 2. depicts the basic structure of a closed-loop control. The control variable y is measured and compared with the defined set point value w. Error e is fed to the controller, which then calculates actuating variable u as reaction to the control path's changes. Although conventional closed-loop controllers react to small changes in their control path (environment), they are not able to handle changes beyond the assumed system behavior of the path or an unpredicted situation. This is because closed-loop controller parameters (e.g. the integral part of a PID-controller) are tailored for the specific control path. When designing conventional controllers, it is assumed that control path behavior is time-invariant, completely known, and mathematically describable. However, conventional closed-loop controller robustness fails when the control path exhibits a time-variant dynamic.

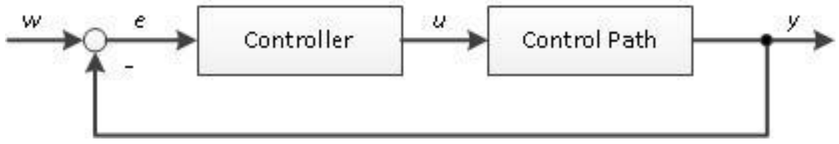


Fig. 2. Basic structure of a conventional closed-loop control

Adaptive controllers appear to approximate an agent's proactive behavior. Adaptive controllers can adjust their control parameters during run time by measuring the current actuating and control variables. Adaptive controllers do this directly or indirectly, and they decide how to adjust their parameters from these measurements [20]. Figure 3 shows the Model Reference Adaptive Control (MRAC) as an example for illustrating the main principle of adaptive controllers. In contrast to the conventional controller, the MRAC exhibits a reference model of the control path. As an indirect adaptive control system, it calculates the difference e_2 between the reference model's behavior and the control path. Based on e_2 and u, the adjustment mechanism calculates new control parameters [40].

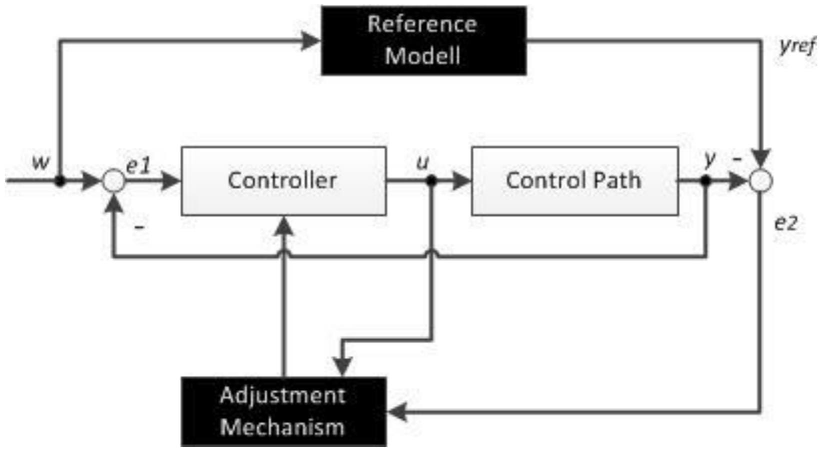


Fig. 3. Basic structure of a Model Reference Adaptive Control (MRAC)

Thus, adaptive controllers recognize environmental changes and react proactively in the sense that they adapt their initial parameters. Nevertheless, the controller is a "functional system" rather than a goal-oriented system aware of alternative ways to reach its goal. This can be illustrated with a simple logistic system:

Imagine a transport unit responsible for transporting a product from one work station to another, which has two alternative tracks available. Equipped with an adaptive control, the transport unit might be able to deliver the product in time, even when there are small obstacles on the track. But the adaptive control is not capable of choosing an alternative path through the production hall, should the track be blocked by other transport units. That is because the adaptive control is not aware of itself, what it actually performs, or the environment in which it operates. It processes the task of ensuring a stable steady state purely. In regard to the example, the controller does not know that it is operating in a production hall which features different corridors to arrive at a destination workstation. For similar reasons, the Model Predictive Control (MPC) can not be understood as proactive. MPCs predict the control path's behavior by using a reference model, but they are not capable of adapting their model to time-variant environments, nor do they have knowledge of themselves and the surrounding environment.

Thus, neither adaptive controllers nor model-predictive controllers can be deemed proactive in the manner agents are proactive. Furthermore, normally controllers are not designed to get in social contact with other controllers, technical systems, or humans. Doubtless, controllers can receive set points and communicate (actual) values of their actuating or controlled value, but they can not interact with other devices in the sense of cooperation or negotiation. This is of prime importance when several distributed controllers influence the same control path, such as in an electricity distribution grid.

3.3. Why Is an Agent More Than Solely a Learning System?

Learning systems are computer programs that use machine learning techniques. Learning as such, is described as a task which "[…] denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks […] more efficiently and more effectively the next time" [37]. Learning systems exhibit neither reactiveness nor proactiviness. This is because, in contrast to controllers or agents, they are not designed to control a system. First and foremost they are software artifacts which facilitate other computer programs or control systems adjustment in response to environmental changes, or the discovery of new patterns in measured data. To do so, they maintain an interaction with their environment, but only in the sense of; a) measuring data (passive unsupervised learning) or; b) experience with the environment while measuring the resulting effects (active unsupervised learning). Furthermore, learning systems have no real social ability. If social ability is exhibited, they are part of distributed problem solving systems which are commonly implemented as multi-agent system.

3.4. Functionalities of Agents

We pointed out that none of the analyzed technologies incorporate all attributes of an agent as described by Jennings and Wooldridge [16]: Reactiveness, proactiveness, and social ability. However, the functionalities of the above described technologies are joined together within an agent. Thus, an agent should not be conceived as synonym for the above mentioned state-of-the-art control technologies. Instead, the term 'agent' expresses a specific software artifact that joins together functionalities of optimizers, controllers, and learning systems.

In contrast to the agent definition of [35], we do not view "anything that […] [is] perceiving its environment through sensors and acting upon environment through actuators" as an agent. As said before, we base our understanding of agents on the definition of Wooldridge and Jennings. Thus, we claim that a software artifact labeled as a (holistic) agent must exhibit, in addition to optimization and controlling the following functionalities; i) some kind of reasoning; and ii) a communication system which allows a high-level agent-to-agent interaction flexible enough to achieve social speech acts as humans do. Therefore, we understand multi-agent based control systems as systems consisting of distributed agents capable of coordination, cooperation and negotiation to gain a stable common control path (environment) while reaching their individual goals. Figure 4 depicts the five functionalities that a holistic agent should exhibit and which impart the attributes of autonomous agents as defined by Wooldridge and Jennings: reactiveness, proactiveness, and social ability. Since the extent of the functionality "Learning" can vary from one application to the other and is in some application even undesirable we marked "Learning" in Figure 4 with a dashed frame as an optional functionality for agents. The five functionalities are described in detail in turn.

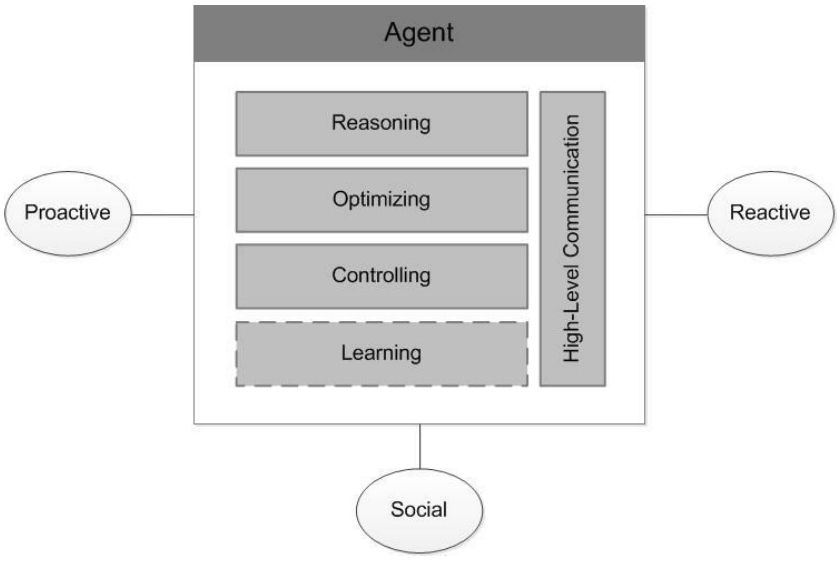


Fig. 4. Functionalities of an agent: reasoning, optimizing, controlling, and high-level communication as well as learning as optional functionality (marked with a dashed frame) imparting the mentioned attributes reactiveness, proactiveness and social ability.

Reasoning

In contrast to optimizers and controllers, agents should be able to perform tasks that were not explicitly defined and programmed at design time. Thus, agents are capable of reaching the programmer's defined goals by processing sequentially different tasks that they develop from a knowledge base of their environments. This process can be illustrated with a real-life scenario: suppose you have a meeting on the other side of city and that you usually take your bike. How would you behave if your bike was broken? You would think (reason) about other vehicles you might be able to use by checking the logical relation between bikes and other known objects, such as cars, tricycles, and walls. Understanding that walls are not vehicles is something taken for granted by humans. For computers, this is a non-trivial task. However, "practical reasoning" [2] enables agents to explore tasks or actions from a logical knowledge of their environments that suit the situation to be solved. After finding a set of possible alternatives, which can also stem from cooperation with other agents, the agents need to make the decision of which task or action they want to perform to reach their goals.

Optimization

Usually, agents make this decision by some sort of performance measurement. [44] describes the idea of associating “utilities with states of the environments”. Utility is a numeric value which shows how “good” an agent’s envisaged task/alternative is. Thus, the agent tries to find one alternative out of the set of explored alternatives that promises the highest utility. This does not imply that agents always act on basis of a single utility function to reach their goals. Sometimes they act on zero or multiple coexisting utility functions. This is illustrated in the above example: through reasoning you figured out that you have three alternative transportation options that will get you to your meeting: by car, by tricycle, or by foot. Thus, you can reach your goal either by choosing the cheapest (by foot) or the most comfortable alternative (the car). But if you have no preference it is irrelevant which vessel (task/alternative) you choose unless you reach your goal “arriving at the meeting place”. This is similar to agents. Agents can either realize the optimal task or choose a task randomly. Expressing this example in a mathematical form, the parallels between optimizers and “decision making”, as it is usually entitled by agent scientists becomes apparent (cf. section 3.1):

$$u(v_1) \leq u(v_2) \text{ for all } v \in V \tag{2}$$

while

$$u : V \rightarrow \mathbb{R}$$

and

$v_{1,2}$ = vehicle one/two, V = Pool of alternative vehicles, u = Cost of transport

Learning

Based on this example, we further want to illustrate the learning ability of agents. Agents, like humans, can recognize changes to their environmental constraints. They can identify new alternatives, such as vehicles that are faster, cheaper, or both. The ability to recognize changes in the environment is vital for reaching the agent’s goal, but also proves to be very difficult. This is because such learning requires - much like reasoning - knowledge about the logical relations of objects in the environment in which the agent is embedded. However, the learning system of an agent modifies the number of alternatives which the reasoning functionality can process and thereby it also modifies the optimizer’s pool of alternatives$V$, indirectly. As mentioned before, the agent’s ability to expand and modify its behavior via learning is not a mandatory functionality for an agent but it constitutes – when implemented – an essential distinguishing feature of an agent compared with controllers or optimizers.

Controlling
After determining which tasks should be realized, agents give commands (set points) to their technical system for which they are responsible (considering the car as technical system in our example, a command could be "start the motor and drive at a certain velocity"). Furthermore, agents need to check if the command is performed correctly and, if not, to take corrective actions. This is what closed-loop controls do, roughly. As mentioned before, closed-loops try to arrange a given set point by reacting based on the control variable's feedback.

High-level Communication
The most distinctive attribute of agents compared to other technical control systems is really an agent's social ability. Social ability does not only include the interaction with humans as done with expert systems [44], it also encompasses the freedom to interact with any valuable communication partner, be it a human, superior entity (e.g. server, market), or other agents. This freedom to automatically and flexibly establish communication at run-time makes a high-level communication interface and language necessary. Such a language is specified by the Foundation for Intelligent Physical Agents (FIPA). Within the FIPA-Specification a high-level communication language is formalized and specified. FIPA-Agents Communication Language (ACL) comprises generic messages classes, so called performatives. These performatives express the type or class of a message without specifying the content and the content language (e.g. the performative "request" is used by agents asking other agents to submit any information) [10]. Thus, agents can establish communication with any kind of content at run-time and are not limited to communication interfaces defined at design time (e.g. web-services) [14], [33]. As described in section 5, this functionality will likely bring valuable advantages and it is indispensable when agents are part of a multi-agent based control systems.

4. "Multi-Agent Systems" in Recent Smart Grid Projects

In this section, we review the most recent smart grid research projects labeled with the term 'agent' or 'multi-agent'. We do not claim to provide a complete listing of all smart grid multi-agent systems. Rather, we intend to carve out common interpretations and implementations of the agent technology by illustrating some representative examples. Therefore, we only considered the domain of distributed agent-based control systems (as defined in [24]). Furthermore, we refer to the more general term 'control entity' (CE) to describe the diverse systems (and their differing interpretations of agent systems) instead of using the term ‘agent’. Subsequently, in section 5, we discuss under which circumstances CEs and control systems can be understood as multi-agent control systems.

The majority of previous multi-agent labeled systems consist of distributed CEs, which are responsible for trading energy on local electricity markets (LEM) by sending bids. In distributed CEs scenarios, CEs calculate energy sales or energy purchase prices based on their individual cost functions. The cost functions for CEs are composed of the specific power device’s (e.g. combined heat and power plant, photovoltaic system, or simply a dwelling with different loads) cost function for which the CE is responsible. After calculating the cost-minimal operation of the power equipment, the CEs send bids to their dedicated LEMs. The LEMs subsequently match the CEs’ energy offers and demands and send the auctions outcome. CEs can either act as sellers or buyers at the local energy market [4], [5], [18], [21], [23], [34], [42].

The general structure of these systems is depicted in Figure 5. In addition to the descripted basic market-oriented structure (continuous lines) a further upper trading level is visualized in the figure (dashed lines). In this level, LEMs can trade energy at an upper electricity market (EM). However, they only do so if they were not able to locally match all supplies and demands of the CEs. Such a cascading system can be found in [18] and [43]. With few exceptions, such as [22], these market-oriented systems usually are non-predictive systems. Thus, they only calculate energy for the next time intervals that range between 500 milliseconds [42], [43] and several minutes [22], and do not calculate a cost optimal schedule (e.g. for the next 24 hours). In most all market-oriented systems labeled as MAS, control entities are only connected to their dedicated local energy market. Hence, these MAS are not programmed to search or adapt their strategies to other local or global markets. Furthermore, current market-oriented multi-agent systems are mainly programmed for one dedicated market type. For example, the above-mentioned systems can handle only bids within an auction based, active power market. However, an agent should be capable of handling all types of markets (e.g. active, reactive, and spinning reserve markets) and all offered products (e.g. spot deals or forward transactions).

Grid-oriented systems constitute the second important application field of multi-agent based distributed control systems in the smart grid. These control systems are primarily responsible for ensuring a grid operation in a normal state, which implies an operation within the standardized voltage, power, and frequency limitations of the grid. For example, [21] extends the market-oriented approach of [43] with a local frequency measurement and adjustment. [32] developed a voltage control system implementing a central CE that optimizes reactive power injection of distributed energy systems (DES). Therefore, the CEs are responsible for the DESs only receiving a set point from the central CE.

Furthermore, in previous grid-oriented projects, distributed control systems capable of performing an automatic reconfiguration of their control parameters in case of grid topology changes were developed. Grid topology[2] changes can arise either by an activation of circuit breakers and (e.g. to increment or decrement the voltage level), or from connection, or disconnection, from the main grid:

a) of either single Distributed Energy Resources (DER) at any instant or
b) of entire grid sections, due to an occurring fault like a short circuit. [30]

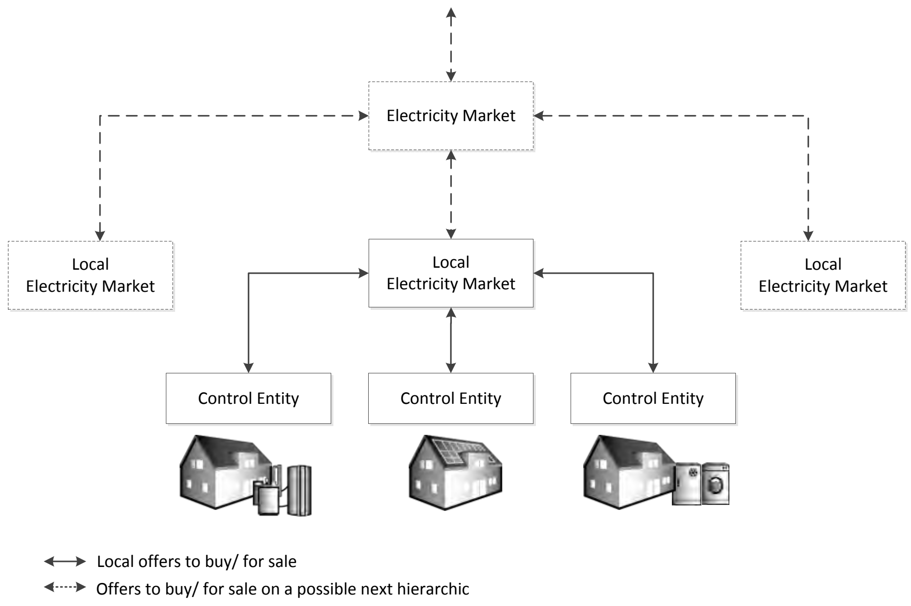


Fig. 5. Common structure of systems labeled as MAS applying an auction based active power spot market.

While the activation of circuit breakers, and the connection and disconnection of DERs, require "solely" the reconfiguration of distributed control entities, the control of separated grid sections is particularly challenging. These so called micro-grids require several non-trivial tasks like fault detection and localization, coordination of the islanded grid section's voltage, and frequency control as well as synchronization when reconnecting to the main grid [30]. [26] and [29] developed distributed control systems, responsible for reliable operation in the normal grid state as well as for automatic and secure transition from the normal grid state, to the island state

[2] Within this paper grid topology is understood as defined in electrical engineering. Thus, a change of any grid parameter (e.g. change of a load, change of impedances, and activation of switches) is regarded as topology change. It encompasses not only the restructuring of electrical grids.

and back. Switches and distributed energy systems (e.g. photovoltaic system, electricity storages, loads, etc.) equipped with interconnected CEs are capable of communicating in a many-to-many manner, and thus may coordinate the grid in every state without passing information via a dedicated central control entity.

5. Agents' Assets and Suitable Application Fields

Comparing the distinction of agents developed in this paper (Section 3) and recent smart grid projects labeled as multi-agent systems (Section 4), we claim that most projects implemented control systems exhibiting some, but not all, agent functionalities in a stricter sense. Although these multi-agent control systems have proved to be very effective, some smart grid applications do not necessitate multi-agent systems as a control system. In the following, we discuss where the usage of (the notion of) multi-agent systems and the application of holistic agents might be valuable assets in the smart grid. We structured this section according to the main smart grid domains: grid-oriented and economic-oriented approaches. While grid-oriented approaches control DERs in a way that reliable grid operation is ensured (e.g. distributed voltage control via reactive power injection), the economic-oriented approaches control DERs grid independently (e.g. virtual power plants).

5.1. Economic-oriented Control Systems

As mentioned above and posed by Jennings and Bussmann, multi-agent systems should demonstrate CEs which have the capability "to initiate (and respond to) interactions that were not foreseen at design time" [15]. Although, the investigated market-oriented systems (in section 4, paragraph 2) constitute decentralized and negotiating CEs, the systems cannot be deemed agent systems in a stricter sense (cf. section 3). This is because:

First, the negotiation of CEs in market-oriented MAS is mainly limited to only one negotiation partner, the local electricity market. Second, the investigated market-oriented MASs demonstrate CEs which are dedicated to only one market type (e.g. real time markets or active energy spot markets) and they are not capable of automatically initiating contracts (for example) on other market types (e.g. forward-markets and reactive energy (spot) markets) or to conclude a direct, bilateral, over-the-counter contract. Both seem to contradict the agent definition and somehow also the legal requirement of being able to freely choose any energy supplier[3]. In contrast, a multi-agent system, in a stricter sense, should allow agents to negotiate energy with any other agent, and thereby enter into any contracts with any

[3] Based on the Liberalization of EU's electricity supply system

partner, either of a local energy market or a gird neighbor who offers or sells energy bidirectionally over the counter.

Further, CEs in economic-oriented systems labeled as multi-agent system should have the freedom to cooperate and to dynamically form subgroups ("holons") with other CEs. Applied in the smart grid, this would enable dynamic and automatic composition (and decomposition) of energy communities. These communities would, for example, try to match electricity production and consumption within their community first, before entering negotiations with external agents or other electricity communities. In addition, it is conceivable that agents would form groups with the purpose of acting as a virtual power plant that appears as a whole, such as when trading energy at the operating reserve market [6], [39].

However, the flexibility to interact at unpredictable times, for unforeseen reasons with other unpredictable CEs makes what social ability embodies evident. **CEs, like those mentioned above** (cf. section 4), are bound to a single web-service enabling bidding at a dedicated LEM, **cannot be deemed "social" and proactive**. (cf. [14]). In combination with a single possibility to modify the bids - which makes reasoning needless - the alleged agents demonstrate more of the characteristics of communicating optimizers. However, what social ability and proactiviness applied in economic-oriented MAS should imply is illustrated by an example: Imagine a rural low-voltage grid with four nodes representing four households with different energy systems. As depicted in Figure 6., the first household is equipped with a combined heat and power plant (CHP), the second and fourth with a photovoltaic system (PV), and the third with a heat pump (HP). All energy systems are controlled by agents responsible for the technical and economical efficient operation. The agent goals, which are usually cost-minimal energy supply and profit-oriented generation, are defined at design time. Other than current control systems, agents are not bound to one control strategy in achieving these goals. Thus, the HP agent might proactively ask the CHP agent and the PV Agent – without dedicate command of a human – if they are interested in cooperating within an energy community. If the other agents agree in that cooperation, all agents need to adapt their control parameters in order to achieve the joint objective of the energy community (social ability): profit maximization while ensuring a reliable energy supply. Further, it is conceivable that the community may decide proactively to participate in the operating reserve market because they assume a higher profit. Such a machine-to-machine decision makes a social interaction and an automatic adjustment of the control parameters indispensable. Figure 6. shows examples of different possibilities through which agents can achieve their goals. While (a) depicts agents that fend for themselves, in (b) and (c) agents are cooperating for either the purpose of self-consumption or to appear as one party on energy markets.

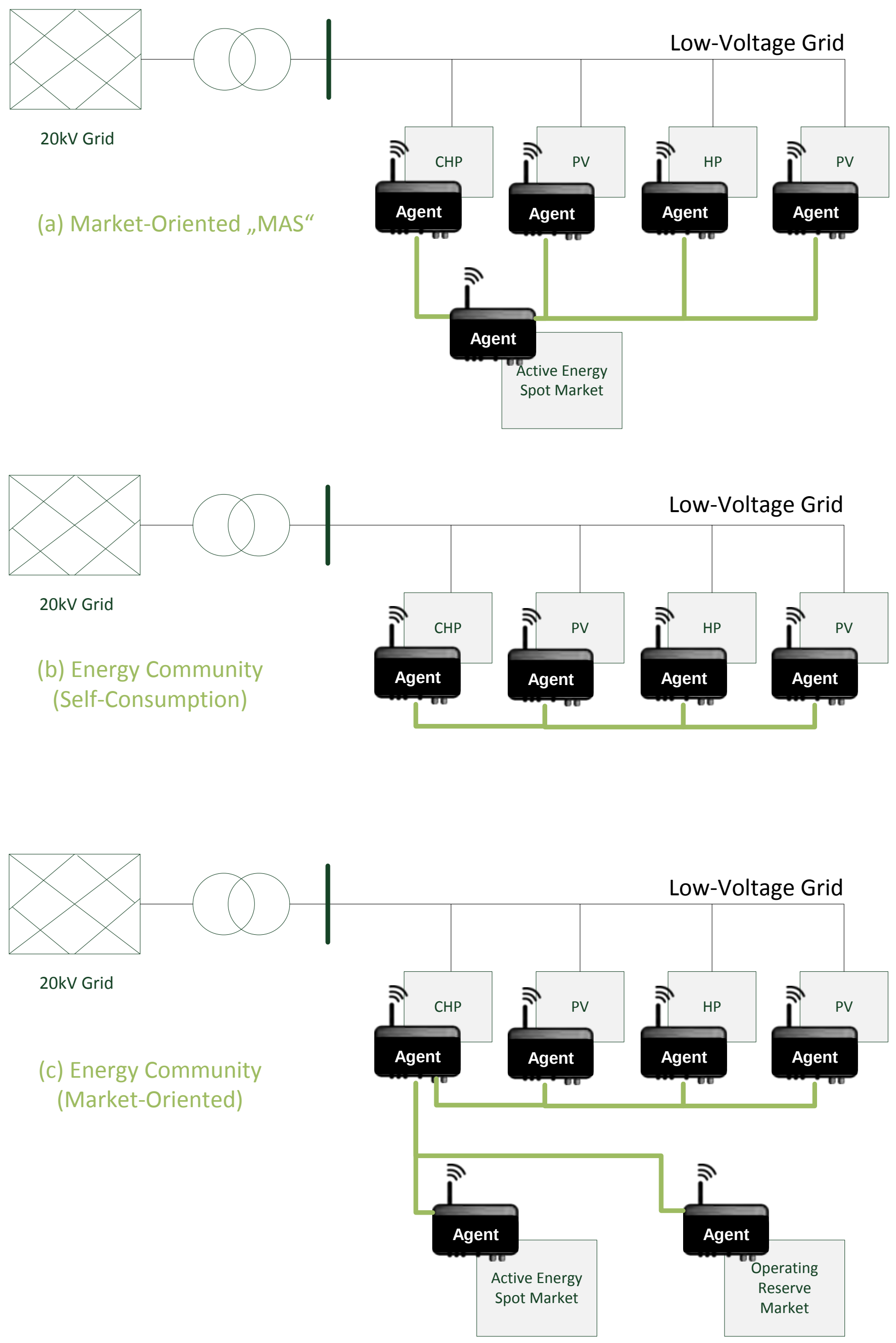

Low-Voltage Grid
20kV Grid
CHP
PV
HP
PV
Agent
(a) Market-Oriented „MAS“
Active Energy Spot Market
Low-Voltage Grid
20kV Grid
(b) Energy Community (Self-Consumption)
Low-Voltage Grid
20kV Grid
(c) Energy Community (Market-Oriented)
Active Energy Spot Market
Operating Reserve Market

Fig. 6. Examples of how agents (in a low-voltage grid) can organize themselves.

5.2. Grid-oriented Multi-Agent Systems

The development of distributed renewable systems obliges network operators to extend their (sub-)transmission control systems to the distribution and low-voltage grids. To preserve operational personal from workload increase, which even today necessitates automation [15], multi-agent systems could be a suitable solution. Industrial manufacturing and industrial process automation already demonstrate that agent-based systems bring sizable advantages when applied as a control system for uncertain, difficult to predict and varying environments. Biological wastewater treatment plants or production systems with repetitive plugging of equipment are the most noteworthy example which Metzger et al. surveyed [25]. These conditions seem – in a weakened form – also present in future electricity distribution grids. The following paragraphs discuss where agent technology in grid-oriented control is of value. Since voltage and frequency control demonstrate the most important and most discussed grid values in smart grid research, we have subdivided this section accordingly. Further, we discuss micro-grid control systems within a separate paragraph, since it is responsible for all values which are important for stable and reliable grid operation.

Voltage Control in Distribution Grids

In addition to frequency control, network operators are responsible for controlling grid voltage. In contrast to the grid frequency, the voltage depends on the grid's topology, encompassing the grid impendency, grid structure, and the active and reactive energy flows of consumers and generators connected to the grid. Voltage control in distributed and low-voltage grids was not important in the past due to the unidirectional energy flows and the associated static drop of voltage along the power line. The transformers in low-voltage grids were adjusted in the way that the voltage was high enough even at the end of the grid. Over-voltage problems are assumed when the current static transformer parameterization remains unchanged while the number of distributed generators (e.g. photovoltaic systems or combined heat and power plants) rises. In order to keep the voltage in the permitted ranges, one approach is to involve the distributed systems in grid stabilization by means of dynamic adaptation of the active (P) and reactive power (Q) depending on the grid voltage. These distributed systems need to be coordinated to avoid voltage oscillation in the grid and to ensure a maximal production of renewable active energy. For this purpose, the **controllers can be extended to agents, capable of coordinating and negotiating injection of power**. To reduce the configuration efforts of the agents and to allow automatic adaptation when the grid structures change, agents should be capable of detecting grid position. These changes in grid structure can develop because, first, short circuits or earth faults need to be recovered, or, second due to an islanding of the grid section in which the agent is located. In both cases manual reconfiguration and coordination of the distributed controllers' parameters seems inappropriate. Thus, agents which realize automatic configuration may be valuable assets in this scenario. Especially in regions with a high percentage of overhead lines (as in the United States) automatic troubleshooting of electricity grid is of paramount interest.

Grid Frequency Control

The gradual substitution of conventional power plants with distributed renewable energy systems compulsorily increase the influence of distributed systems on the frequency. To ensure a stable and reliable grid in the future frequency control need to be coordinated among those millions of distributed systems, since large frequency changes can lead to an unstable power system.

In today's central European electricity grid (ENTSO–E[4] grid), the transmission grid operators are responsible for the frequency control. The frequency control is divided in three control levels which differ in their time of activation: a) primary control (frequency-response reserve), b) secondary control (spinning reserve), and c) tertiary control (replacement reserve). In case of a frequency deviation, the control levels are activated step-wise, beginning with the primary control, via the secondary control and ending at tertiary control (if the deviation still exists). The primary control reacts within a few seconds (max. 30 seconds) to frequency deviation by automatically adapting the power of some generation units in the electricity grid according to a given static. After one minute, the secondary control replaces the primary control and tries to restore the frequency to the nominal value (in Europe 50 Hz). If it is not possible to restore the frequency with the secondary control after five minutes, tertiary control is manual (e.g. via phone) requested by transmission operators. All forms of operating reserve (frequency-response reserve, spinning reserve and replacement reserve) are traded on the operating reserve market. Currently only larger power plants are allowed to participate on the operating reserve markets.

Assuming that the current operating reserve mechanism remains unchanged, while in the future small distributed generation units are also allowed to trade replacement reserve, **agents might be of value when enabling automated participation of distributed energy systems at the operating reserve market.** As a control entity for a distributed energy system (e.g. a combined heat and power plant) the agent would autonomously negotiate at the operating reserve market in order to operate its distributed energy system cost-optimally. [21] describes such a multi-agent system which controls the frequency via a market-oriented multi-agent system. A Balancing-Agent, which is responsible for the frequency, offers active energy for consumption when the frequency is about to drop and buys energy - which is left unused - when the frequency rises. Here, the Balancing-Agent appears as a central coordinating unit to which the other agents are

[4] European Network of Transmission System Operators

dedicated. As mentioned above, it is questionable if a control entity which is slowly connected to the operating reserve market server shall be deemed an agent in strictest sense. The automatic negotiation can also be implemented using web-services. But as described by [14] web-services are not agents. Hence, labeling negotiating control entities as agents would only make sense when they demonstrate proactiviness and social behavior as described in section 3. Consistently, proactiviness and social behavior are only feasible if alternative courses of action exist. It is the control entity which is capable of deciding, for example, to either offer at the operating reserve market or to any other energy market (cf. section 5.1), which would constitute a freedom, on the basis of which a control entity may develop its proactiviness. Furthermore, social behavior only evolves if other control entities and, with that, possibilities of cooperation (e.g. as virtual power plant) exist.

Besides the question of how distributed energy resources can participate passively at the frequency control via a replacement reserve market, it still remains unclear how they can participate actively[5], without causing grid destabilization. As mentioned above, recent photovoltaic inverters were configured to disconnect from the grid if the frequency reached the limit of 50.2 Hz. If a large amount of systems would react accordingly, this would cause serious grid problems [17]. Thus, the parameterization of the distributed energy system via cooperating agents – as applied for voltage control – would be conceivable. But other than the voltage, the frequency constitutes a grid-wide value. This would imply the coordination of millions of energy systems, which seems unrealistic mainly due the associated high communication traffic. Assuming that, in the future, only some power units are responsible for the supply of frequency-response and spinning reserve power, a multi-agent system might be an appropriate solution for an automatic, autonomous, and self-parameterizing control system of the power plants control, as described in [1].

Micro-Grid Control

Micro-grids seem to be the most appropriate smart grid domain for the applications of multi-agent based control systems regarding the recent numerous micro-grid projects [19]. [3] defines micro-grids as small, local distribution systems containing generation and loads that can be separated totally from the distribution grid. Micro-grids constitute, indeed, perfect environments for the application of MAS, for the following reasons: First, other than the main grid, the micro-grid demonstrates a small separate system comprising of a limited number of control entities. That makes the coordination and the assigned communication efforts manageable. Other than in the smart grid parameterization, for example, of the frequency

[5] Actively implies that the control units measure the frequency at their dedicated grid node and react directly to frequency deviations by adapting the injected active power, or reactive power.

controllers becomes feasible. Second, micro-grids in disconnected mode are fully responsible for the stable and reliable operation of the grid. That encompasses, besides the voltage control, the frequency control and protection issues as well. The indispensable energy balance needs to be ensured, while, caused by disconnection, only a reduced number of devices that can provide reserves exist. Likewise, the micro-grid control system needs to react to regular changes in the grid topology (e.g. disconnection of distributed energy resources or loads). Hence, as described in [30], MAS would be a valuable asset, because other than in a centralized micro-grid control system, no manual adaption of the central control algorithms/models is necessary. Within a MAS automatic adaption of the distributed control parameters takes place if topology changes occur. Numerous multi-agent based micro-grid control systems can be found in literature. [19] and [30] give a broad overview about recent multi-agent based micro-grid controls.

In this section we illustrated that the terms ‘agent’ and ‘multi-agent’ systems applied in the smart grid are of value when automatic, cooperative, and coordinated reconfiguration of distributed devices (e.g. distributed energy systems or grid equipment) is required during runtime. We discussed the application of multi-agent systems for frequency control, voltage, micro-grid control (grid-oriented), and economic-oriented control systems, since they constitute the major smart grid control domains. Further applications, such as the optimal coordination of an electric vehicle fleet or substation monitoring and diagnostic systems, can be considered as further possible applicable fields for multi-agent systems. However, it is recommended to carefully consider if either multi-agent systems or current state-of-the-art control technology should be applied, as the case pops up on a case by case basis. Furthermore, the term ‘agent’ should only be used if the control systems evince the above mentioned abilities: proactivity, reactivity and social behavior, and not only as a synonym for state-of-the-art control technologies.

6. Conclusion

This article has sought to justify why and when multi-agent systems are suited for smart grid applications. First, we have posed that state-of-the-art control technologies should not be understood as agents, but, in turn, agents should be understood as software artefacts that exhibit the functionalities of optimizers, controllers, and learning systems, accompanied by the capabilities of practical reasoning and social interaction. Second, we contrasted our understanding of agents with interpretations of recent smart grid projects labelled as multi-agent control systems. These projects have already demonstrated the effectiveness of multi-agent systems, although they implement agents only in the broadest senses: due to the decentralized structure, data is locally processed where it is produced and local negotiation effects a coordination of the distributed systems. However, to explore all agent benefits and to sharpen the notion, we suggest terming control entities as agents when they encompass all of the above mentioned functionalities. The assets are: i) automatic reconfiguration in case of time-invariant environment changes (e.g. grid topology changes); ii) automatic initiation of and participation in (economic) interest groups (virtual power plants); and iii) automatic adaption to changing control strategies (e.g. in case of grid islanding). Coordination of thousands of distributed, fluctuating, electricity generators and a robust operation of our electricity grids constitutes an enormous future control challenge. This dynamic, distributed, and widespread environment seems to be perfectly suited for the application of agent-based control systems.

References

1. Bevrani, H. (ed.): Agent-based robust frequency regulation. In: Robust Power System Frequency Control, pp. 1–17. Power Electronics and Power Systems, Springer US (2009)
2. Bratman, M.E.: What is intention? pp. 15–32. The MIT Press, Cambridge, MA (1990)
3. Chicco, G., Mancarella, P.: Distributed multi-generation: a comprehensive view. Renewable and Sustainable Energy Reviews 13, 535–551 (2009)
4. Chung, I.Y., Yoo, C.H., Oh, S.J.: Distributed intelligent microgrid control using multi-agent systems. Engineering 5 (2013)
5. Dimeas, A., Hatziargyriou, N.: Operation of a multiagent system for microgrid control. IEEE Transactions on Power Systems 20(3), 1447–1455 (2005)
6. Dimeas, A., Hatziargyriou, N.: Agent based control of virtual power plants. In: Intelligent Systems Applications to Power Systems, 2007. ISAP 2007. International Conference on. pp. 1–6 (2007)
7. European Parliament: EU Directive 2009/28/EG. Official Journal of the European Union (2009)
8. European Technology Platform Smart Grids: Smartgrids strategic deployment document for europe's electricity networks of the future (2010)
9. Foster, I.T., Jennings, N.R., Kesselman, C.: Brain meets brawn: Why grid and agents need each other. In: Autonomous Agents & Multiagent Systems/International Conference on Autonomous Agents. pp. 8–15 (2004)
10. Foundation for Intelligent Physical Agents: FIPA ACL Message Structure Specification., http://www.fipa.org/repository/aclspecs.html, available at http://www.fipa.org/repository/aclspecs.html. Accessed on 23th February 2013
11. Franklin, S., Graesser, A.: Is it an agent, or just a program?: A taxonomy for autonomous agents. In: Müller, J.P., Wooldridge, M., Jennings, N.R. (eds.) Intelligent Agents III Agent Theories, Architectures, and Languages, Lecture Notes in Computer Science, vol. 1193, pp. 21–35. Springer Berlin Heidelberg (1997)
12. Giordano, V., Gangale, F., Fulli, G.: Smart grid projects in europe: lessons learned and current developments (2011)
13. Hollinger, R., Hamperl, S., Erge, T., et al.: Simulating the optimized participation of distributed controllable generators at the spot market for electricity considering prediction errors. In: Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES (2012)

14. Huhns, M.N.: Agents as web services. IEEE Internet Computing 6, 93–95 (2002)
15. Jennings, N., Bussmann, S.: Agent-based control systems. IEEE control systems 23(3), 61–74 (2003)
16. Jennings, N., Wooldridge, M.: Agent Technology: Foundations, Applications, and Markets. Springer-Verlag, Berlin (1998)
17. Kaestle, G., Vrana, T.K.: Das 50,2 hz-problem im kontext verbesserter netzanschlussbedingungen. In: Internationaler ETG-Kongress 2011 (ETG-FB 130) (2011)
18. Kamphuis, R., Roossien, B., Bliek, F., et al.: Architectural design and first results evaluation of the powermatching city field test. In: 4th International Conference on Integration of Renewable and Distributed Energy Resources. EPRI (2010)
19. Kulasekera, A.L., Gopura, R.A.R.C., Hemapala, K.T.M.U., et al.: A review on multi-agent systems in microgrid applications. In: Innovative Smart Grid Technologies - India (ISGT India). pp. 173–177 (2011)
20. Landau, I.D., Lozano, R., M'Saad, M., et al.: Adaptive Control Algorithms, Analysis and Applications. Springer Verlag London Limited (2011)
21. Linnenberg, T., Wior, I., Schreiber, S., et al.: A market-based multi-agent-system for decentralized power and grid control. In: Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on (2011)
22. Logenthiran, T., Srinivasa, D., Khambadkone, A.M.: Multi-agent system for energy resource scheduling of integrated microgrids in a distributed system. Electric Power Systems Research 81(1), 138–148 (January 2011)
23. Logenthiran, T., Srinivasan, D., Khambadkone, A., et al.: Multiagent system for real-time operation of a microgrid in real-time digital simulator. Smart Grid, IEEE Transactions on 3(2), 925 –933 (2012)
24. McArthur, S.D.J., Davidson, E.M., Catterson, V., et al.: Multi-Agent Systems for Power Engineering Applications – Part I: Concepts, Approaches, and Technical Challenges (2007)
25. Metzger, M., Polakow, G.: A survey on applications of agent technology in industrial process control. Industrial Informatics, IEEE Transactions on 7(4), 570 –581 (2011)
26. Nguyen, C., Flueck, A.: Agent based restoration with distributed energy storage support in smart grids. IEEE Transactions on Smart Grid 3(2), 1029 –1038 (2012)
27. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization Algorithms and Complexity. Dover Publications Inc. (2000)
28. Parunak, H.V.D., Sauter, J., Fleischer, M.: The rappid project: Symbiosis between industrial requirements and mas research. Autonomous Agents and Multi-Agent Systems 2, 11–140 (1999)
29. Pipattanasomporn, M., Feroze, H., Rahman, S.: Multi-agent systems in a distributed smart grid: Design and implementation. In: Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES. pp. 1–8 (2009)
30. Planas, E., de Muro, A.G., Andreu, J., et al.: General aspects, hierarchical controls and droop methods in microgrids: A review. Renewable and Sustainable Energy Reviews 17(0), 147 – 159 (2013)
31. Pudjianto, D., Ramsay, C., Strbac, G.: Virtual power plant and system integration of distributed energy resources. Renewable Power Generation, IET 1(1), 10–16 (2007)
32. Richardot, O.: Réglage Coordonné de Tension dans les Réseaux de Distribution à l'aide de la Production Décentralisée. Phd thesis, Grenoble, France (2006)

33. Rohbogner, G., Fey, S., Hahnel, U.J., et al.: What the term agent stands for in the smart grid. definition of agents and mulitagent systems from an engineer's perspective. In: Computer Science and Information Systems (FedCSIS), 2012 Federated Conference (2012)
34. Roossien, B.: Field-test upscaling of multi-agent coordination in the electricity grid. In: Electricity Distribution - 20th International Conference and Exhibition on. pp. 1–4 (2009)
35. Russell, S., Norvig, P.: Artificial Intelligence: A modern Approach. Pearson Education Inc. (2010)
36. Samadi, P., Mohsenian-Rad, H., Schober, R., et al.: Advanced demand side management for the future smart grid using mechanism design. Smart Grid, IEEE Transactions on 3(3), 1170–1180 (2012)
37. Simon, H.: Machine Learning: An Artificial Intelligence Approach, chap. Why should machines learn? Morgen Kaufmann (1983)
38. SMB Smart Grid Strategic Group (SG3): IEC smart grid standardization roadmap (2010)
39. Tröschel, M., Appelrath, H.J.: Towards reactive scheduling for large-scale virtual power plants. In: Braubach, L., Hoekand, W., Petta, P., et al. (eds.) Multiagent System Technologies, Lecture Notes in Computer Science, vol. 5774, pp. 141–152. Springer Berlin Heidelberg (2009)
40. Unbehauen, H.: Regelungstechnik III Identification, Adaption, Optimierung. Springer Vieweg (2011)
41. Vasirani, M., Kota, R., Cavalcante, R.L., et al.: Virtual power plants of wind power generators and electric vehicles. Tech. rep. (2012)
42. Wedde, H., Lehnhoff, S., Handschin, E., et al.: Real-time multi-agent support for decentralized management of electric power. In: Real-Time Systems, 2006. 18th Euromicro Conference on (2006)
43. Wedde, H., Lehnhoff, S., Moritz, K., et al.: Distributed learning strategies for collaborative agents in adaptive decentralized power systems. In: Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the. pp. 26 –35 (2008 2008)
44. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons Ltd (2009)
45. Wooldridge, M., Jennings, N.R.: Agent theories, architectures, and languages: A survey. Lecture Notes in Computer Science 890, 1–39 (1995)

Gregor Rohbogner studied industrial engineering at the RWTH Aachen University. He received his Diploma in July 2011. Since 2010 he is a researcher within the Institute for Solar Energy Systems ISE in Freiburg (Germany). First he focused on topics of in-house and Smart Grid communication. Since mid 2011 he works as research fellow on multi-agent systems for Smart Grid applications. Further interest includes Peer-to-Peer networks, Smart Home technologies, optimization of decentralized energy resources and electrical grid control technologies.

Ulf J.J. Hahnel studied psychology at the University of Mainz, Germany and the University of Auckland, New Zealand. Currently, he works in the department of Cognition, Emotion, and Communication at the Institute of Psychology of the Albert-Ludwigs University Freiburg and in the department of Intelligent Energy Systems at the Fraunhofer Institute for Solar Energy Systems (ISE). His research focuses on sustainable consumer behavior, human factors psychology and consumer acceptance of innovative technologies. He published his works in international journals such as Transportation Research Part A and Ergonomics.

Pascal Benoit graduated in electrical and information engineering at Karlsruhe Institute of Technology (KIT) where he specialized in renewable energies and information technologies. During his studies he stayed abroad for several student research projects at Institute of Energy Engineering of the Politécnical University of Valencia, Spain. Since 2010 he is working at Fraunhofer Institute for Solar Energy Systems (ISE) in the department of Smart Grids. There he is active in the field of smart integration of electric vehicles and communication technologies for distributed energy systems. Since 2011 he is working on a PhD thesis focusing on new approaches for control and communication systems of large-scale concentrator photovoltaic (CPV) power plants.

Simon Fey finished his Master Degree in Electrical Engineering / Information Technology at the Offenburg University of Applied Science in 2010. He wrote his diploma thesis at the Fraunhofer Institute for Solar Energy Systems (ISE) on acquisition and storage of measurement data from power plants. Simon Fey is member of the graduate school Small Scale Renewable Energy Systems (KleE) and is currently writing his PhD thesis under the supervision of Prof. Dr. Andreas H. Christ. His thesis is on IT-based communication structures and system architectures for renewable energy systems.

Received: February 24, 2013; Accepted: June 24, 2013

DOI:10.2298/CSIS130304073B

Model-based Integration of Constrained Search Spaces into Distributed Planning of Active Power Provision

Jörg Bremer[1] and Michael Sonnenschein[1]

Department of Computing Science, University of Oldenburg
26129 Oldenburg, Gemany
{Joerg.Bremer, Michael.Sonnenschein}@uni-oldenburg.de

Abstract. The current upheaval in the electricity sector demands distributed generation schemes that take into account individually configured energy units and new grid structures. At the same time, this change is heading for a paradigm shift in controlling these energy resources within the grid. Pro-active scheduling of active power within a (from a controlling perspective) loosely coupled group of distributed energy resources demands for distributed planning and optimization methods that take into account the individual feasible region in local search spaces modeled by surrogate models. We propose a method that uses support vector based black-box models for re-constructing feasible regions for automated, local solution repair during scheduling and combine it with a distributed greedy approach for finding an appropriate partition of a desired target schedule into operable schedules for each participating energy unit.

Keywords: constrained optimization, support vector machines, smart grid, decoder-based optimization.

1. Introduction

In order to allow for a transition of the current central market and network structure of today's electricity grid to a decentralized smart grid, an efficient management of numerous distributed energy resources (DER) will become more and more indispensable. Integrating a continuously rising number of renewable resources means controlling individually configured and rather small devices in order to cope with stochastic feed-in effects. At the same time, more and more electricity is generated by wind energy converters and photovoltaic panels. A forecast on this generation highly depends on only fairly foreseen weather conditions and thus puts an additional challenge on planning the electricity provision and calls for additional operating reserve.

Within an electricity grid, electricity generation and consumption have to be balanced at every moment in time for physical reasons. In the past, only the generated part used to be planned according to anticipated consumption. Only few and large power plants had to be taken into account. As in future a larger share of the generation becomes hardly controllable (e. g. wind energy conversion or photovoltaics), parts of the demand have to be integrated into the planning process.

We consider in general producers that are supposed to pool together with likewise distributed electricity consumers and prosumers (like batteries) in order to jointly gain more degrees of freedom in choosing load schedules. In this way, they become a single controllable entity with sufficient market power. In order to manage such a pool of DERs in a self-organized way, the following distributed optimization problem has to be frequently solved: A partition of a demanded (by market) aggregate schedule has to be determined in order to fairly distribute the load among all participating DERs. Optimality usually refers to local (individual cost) as well as to global (e.g. environmental impact) objectives in addition to the main goal: Resemble the wanted overall load schedule as close as possible.

In order to choose an appropriate schedule for each participating DER, an optimization algorithm must know from each DER, which schedules are actually operable and which are not. Depending on the type of DER, different constraints restrict possible operations. The information about individual local feasibility of schedules has to be spread appropriately in (distributed) optimization scenarios, in order to evaluate objectives globally in distributed search spaces. For this purpose, meta-models of constrained spaces of operable schedules have been shown indispensable for efficient communication [10]. Such models can be seen as black-box representations of the feasible region of an optimization problem related to scheduling scenarios. Such models are also used for efficiently evaluating constraints during the optimization procedure for cases where determining the feasible region has comparably high computational costs.

Real world problems like this scheduling problem often face nonlinear and/ or combined constraints. The set of constraints defines the shape of a region within the search space (the hypercube defined by operation parameter limits) that contains all feasible solutions. This region is called feasible region and might be arbitrary shaped or even be discontinuous. It is this region that defines feasibility and that has to be modeled. Such a surrogate model then allows distinguishing operable and not operable schedules when integrated into the optimization process.

At the same time, support vector machines and related approaches have been shown to have excellent performance when trained as classifiers for multiple purposes, especially real world problems. As a use case related to describing the region where some given data resides in, Tax and Duin developed the support vector domain description (SVDD) as a one-class support vector classification approach that is capable of learning the region that is defined by some given training data [43] and that has therefore been harnessed for example by [7] as a model for the feasible region in the smart grid domain.

What we will now add is a new approach for integrating constraints that are modeled by a support vector classifier into distributed optimization in a way, that allows for an efficient navigation within the feasible region. The basic idea is to construct a mapping from the whole, unconstrained domain of the problem (the hypercube) to the feasible region to be able to automatically repair an infeasible solution during optimization. In this way, the problem is transferred into an unconstrained one by mapping any arbitrary solution onto a nearby feasible one. What we will need for constructing this mapping is the set of support vectors together with the associated weights that make up the black-box model.

The rest of the paper is organized as follows: We start with a discussion of related approaches and the background of the optimization problem that is considered throughout this paper. We describe a model of individual search spaces of arbitrary energy resources based on a support vector approach and the behaviour model based method for learning it. Then, we define the mapping function that is used as solution repair mechanism within our greedy algorithm for scheduling. Finally, we conclude with results from several simulation runs and with a discussion of a possible extension to an asynchronous execution of the optimization.

2. Related work and problem background

Within the framework of today's (centralized) operation planning for power stations, different heuristics are already harnessed. Examples from the research sector are for instance shown in [28] or in [47]. The task of (short-term) scheduling of different generators is also known as unit commitment problem and assigns (in its classical interpretation) discrete-time-varying production levels to generators for a given planning horizon [36]. It is known to be an NP-hard problem [17]. Determining an exact global optimum is, in any case, not possible until ex post due to uncertainties and forecast errors. In practice the software package BoFIT is often used, harnessing a mixed integer model with operational constraints as an integral part of the implementation of the model [14]. This fact makes it hard to exchange operational constraints in case of a changed setting (e.g. a new composition of energy resources) of the whole generation system.

Coordinating a pool of distributed generators and consumers with the intent to provide certain aggregated load schedules for active power has some objective similarities to controlling a virtual power plant (VPP) [46, 27]. Within the smart grid domain the volatile character of such a group has additionally to be taken into account. On an abstract level, approaches to control groups of distributed devices can be roughly divided into centralized and distributed scheduling algorithms.

Centralized approaches have long time dominated the discussion [46], not least because a generator may achieve slightly greater benefit if optimization is done from a global, omniscient perspective [16]. Centralized methods are discussed in the context of static pools of DERs with drawbacks and restrictions regarding scalability and particularly flexibility.

Recently, distributed approaches gained more and more importance. Different works proposed hierarchical and decentralized architectures based on multi-agent systems and market based computing [21, 22]. Newer approaches try to establish self-organization between actors within the grid [20, 29, 38].

Especially for optimization approaches in smart grid scenarios, black-box models for encoding the feasible region with the set of operable schedules have been developed [10]. The encoding of a schedule's individual cost may also be easily embedded into such model [8].

This quite new support vector approach uses support vector meta-models for black-box optimization scenarios with no explicitly given constraint boundaries. In general, various classification or regression methods could be harnessed for creating such models for the boundary [23], but not all of them allow for a good integration into optimization. In general, there are three main reasons for using such a model-based approach:

1. Substituting computational costs for evaluating the constraints by a comparatively easy check on feasibility through the model.
2. Efficient communication in distributed environments due to the small footprint of the model.
3. Unification of access to the information on feasibility.

Besides, the smart grid domain serves also as an example for scenarios with (at least partly) unknown functional relationships of the constraints. The feasible region can sometimes only be derived with lacking full knowledge on hidden variables or intrinsic relations that determine the operability of an electric device and therewith the feasible region. The authors therefore have their model learned by a support vector data description approach from a set of operable (feasible) examples.

In a related approach for another use-case, [4] used a two-class SVM for learning operation point and bias of a line in a power grid for easier determining an optimal way back to stable grid conditions in case of a failure.

Surrogate modeling (also known as surface or meta modeling) is often used for replacing expensive and time consuming evaluations of simulations by a model that mimics system behavior on a local or global level [11] with a minimum of known samples from the original (simulation) model. For this reason, active sampling is often applied, i.e. the sample that makes up the model is continuously adapted and iteratively improved as analysis or optimization evolves. Several different techniques for surrogate modeling have been developed. Commonly used examples are: Datascape, Kriging, first and second order regression, response surfaces, or artificial neural networks [15, 31]. Using surrogate modeling like surface modeling in optimization, usually, all constraints are directly known in contrast to the function that is to be optimized along [32]. In our problem, we want to abstract from certain given constraint formulations and do (at optimization time) not have access to the individual simulations of the distributed resources.

For our optimization problem, we need a model that serves as a stencil for the set of (technically) operable schedules for a DER. We regard the respective simulation model as a characteristic function that is able to indicate whether an arbitrary, given schedule is operable by the DER or not. We want to build a model that is able to guide an optimization algorithm towards feasible solutions. Thus, we need a model that is able to capture the characteristic function that indicates operability of schedules. As we do not have access to the simulation model at optimization time anymore, we have to build the model completely in advance and are not able to come back to iteratively improving modeling techniques like active learning. A support vector model can do this and an important advantage of such a model for our use case is the internal representation that can easily and directly be used for further calculations.

In this paper, we will consider the following optimization problem for a given group (consumers, producers and/ or prosumers) of DERs: A schedule for a given future time horizon is requested (e.g. via an electricity market mechanism) and is supposed to be jointly operated by the group. We developed a general method for arbitrarily composed groups of different types of DER. As the method will abstract from precise modeling of each DER as well as from constraint modeling, it is suitable especially for groups that dynamically reformate frequently in a self-organizing system. For an (not necessarily known in advance) group, a partition has to be found. Thus, we do not assume a certain size or composition of different DERs. A partition of the requested target schedule has to be determined in order to fairly distribute the load among all participants.

With the term load, we denote the mean electrical active power that is produced or consumed by a DER within a certain time interval (today: usually 15 minutes). A schedule then is a vector that determines the loads for a given number of subsequent time intervals. This definition is equivalent to defining a schedule by using the respective amounts of energy produced or consumed within a time interval.

For the sake of simplicity, we will consider optimality as a close as possible adaption of the aggregated schedule (sum of individual loads) to the requested one during this paper. Optimality usually refers to additional local (individual cost) as well as to global (e.g. environmental impact) objectives. As we present a general approach that is independent of a certain optimization objective, we have chosen this simple version for demonstration purposes. When coming into operation, one would use a more sophisticated objective; and usually a many-objective approach that takes into account local (individual user-specific) optimality as well as additional global objectives like minimizing the coincidence factor [1].

When determining an optimal partition of the target schedule for load distribution, exactly one alternative schedule is taken from each DER's search space of individual operable schedules in order to assemble the desired aggregate schedule. Therefore, the optimization problem is to find any combination of schedules (one from each DER with $\mathcal{F}_i$ as the set of possible choices, i.e. the individual feasible region) that resembles the target schedules s_{target} as close as possible, i.e. minimize the Euclidean distance ($\|\cdot\|$) between aggregated and target schedule:

$$\|\sum_i x_i - s_{target}\| \to \min, \tag{1}$$

such that each schedule is taken from the respective feasible region $\mathcal{F}_i$ of operable schedules

$$x_i \in \mathcal{F}_i$$

of unit i. The term unit in this context denotes a single DER or an aggregation of commonly controlled energy resources, e.g. the set of all controllable appliances in a household that (from an outside position) can be seen as a single controllable unit. The individual schedules as well as the wanted target schedule are each given as a vector $x_i, s_{target} \in \mathbb{R}^d$ that represents with its elements the mean active power during the respective time period (usually but not necessarily a 15-minute interval). Usually the problem consist of additional objective functions and results in an many objective problem. For demonstration purposes, we will stick with the single objective problem throughout the rest of the paper.

Moreover, the choice of using the Euclidean distance as metric would have to be reconsidered according to the given problem at hand as it is known that the distance of two arbitrarily chosen points tends to approach 1 with growing dimensionality. The same problem holds true when learning a model. For kernel based approaches, [13] proposes methods to overcome this problem. For planning periods of one day with 96 intervals of 15-minutes, the choice will be sufficient for the case of the objective function. As an alternative distance, for example [18] uses the L_1-distance. Depending on special objectives that one wants to achieve (e.g. minimizing surplus production), maybe also metrics like excess supply, are appropriate. An overview on methods to assess the match between schedules can for example be found in [5].

The following section will explain our approach for solving this optimization problem with individually acting DER as part of a coalition of DERs in a distributed approach. Although the problem is defined on a group of DERs, we will first have to look at a single unit and on how to model its abilities before putting together the models from each DER into a jointly solved optimization approach for the multi DER problem.

If such a group of DERs consists of individually operated units, we first have to determine which set of alternative schedules each unit has to offer for the afterwards optimization step. A schedule for d time intervals will be geometrically interpreted as a point in $\mathbb{R}^d$. The model that we present will be applicable to arbitrary types of DER but we will restrict our explanations to the example of a co-generation plant.

3. The model-based optimization approach

3.1. The feasible region

Each DER has to serve the purpose it has been built for and this purpose may usually be achieved in different alternative ways. For example, it is the main purpose of a μCHP (combined heat and power generator) to deliver enough heat for varying heat demands in a household at every moment in time. Nevertheless, heat usage is usually decoupled from heat production by using a thermal buffer store. Thus, different production profiles may be used for generating the heat. This leads, in turn, to different respective electric load profiles that may be offered as alternatives to a scheduling controller.

Each DER offers a set of operable schedules for a given (future) time horizon. We see a schedule as a data vector $x \in \mathbb{R}^d$, with the number of periods d. For each period the i-th element of x describes the respective amount of electric energy produced or consumed in this period or respectively the mean active power output or input during this period.

The term operable in this context means that such a schedule might be operated by the DER without violating any technical constraint. Moreover, we consider additional non-technical constraints that may restrict the possible operations of a DER. Constraints can be distinguished into hard (usually technically rooted) and soft constraints (often economically or ecologically rooted or subject to personal preferences).
Examples for hard constraints are:

- Minimum and/or maximum power input/output
- Integrated amount of energy produced over the given time frame
- Restrictions on thermal buffer storage
- Achieve intended purpose

Examples for soft constraints are:

- Costs (e.g. fuel costs) for operating a certain schedule
- Environmental performance
- Personal preferences (e.g. noise pollution in the evening)

These constraints can be interpreted geometrically. Without any constraint, the whole hypercube $[0,1]^d$ (active power between 0 and 100%) would be a model for the region of feasible schedules. With each constraint, a different part (region) of the hypercube falls off the feasible region, because the respective schedules are not operable due to the constraint. Only the finally remaining region (hypercube minus superposition of all regions prohibited by constraints) is the feasible region of the DER. Only from this region, schedules might be taken during optimization.

It has been shown in [10] that the feasible region of operable schedules of a DER is not necessarily a convex polytope nor a single and connected region. For this reason, concavity and clusters have to be taken into account, too. Such considerations have led to black-box models based on machine learning approaches that may

- capture the topological traits of the feasible region as a compact description of the set of all operable schedules.
- be easily communicated as a standardized description within distributed optimization scenarios.
- ease the calculation of the feasibility of a solution during optimization.

Before we discuss a new way of integrating this model directly into optimization, we will briefly discuss the basic idea of the model approach.

3.2. Support vector black-box model for constraints

We will describe the black-box model for the set of feasible schedules for a DER as it has been developed in [10] based on a one-class support vector data description (SVDD) [44]. The goal of building such a model is to learn the feasible region of the schedules of a DER by learning the enclosing boundary around the set of operable schedules. This task is achieved by determining a mapping $\Phi : \mathcal{X} \subset \mathbb{R}^d \to \mathcal{H}, x \mapsto \Phi(x)$ such that all data from a given region $\mathcal{X}$ is mapped to a minimal hypersphere in some high- or indefinite-dimensional space $\mathcal{H}$. Originally, the use case was to use this model as a classifier that allows for distinguishing operable and not operable schedules during optimization without explicit knowledge about the constraints that restrict the operations of the DER.

The minimal sphere with radius R and center a in $\mathcal{H}$ that encloses $\{\Phi(x_i)\}_N$ can be derived from

$$\|\Phi(x_i) - a\|^2 \leq R^2 + \xi_i \quad \forall i \tag{2}$$

with $\|\cdot\|$ denoting the Euclidean norm and incorporating slack variables $\xi_i \geq 0$ that introduce soft constraints for sphere determination.

After introducing Lagrangian multipliers and further relaxing to the Wolfe dual form, the well known Mercer's theorem [39] may be harnessed for calculating dot products in $\mathcal{H}$ by means of a Mercer kernel in data space:

$$\Phi(x_i) \cdot \Phi(x_j) = k(x_i, x_j). \tag{3}$$

In order to gain a more smooth adaption, it is known [3] to be advantageous to use a Gaussian kernel:

$$k_{\mathcal{G}}(x_i, x_j) = e^{-\frac{1}{2\sigma^2}\|x_i - x_j\|^2} \tag{4}$$

instead of for instance polynomial kernels.

Putting it all together, the equation that has to be maximized in order to determine the desired sphere is:

$$W(\beta) = \sum_i k(x_i, x_i)\beta_i - \sum_{i,j} \beta_i \beta_j k(x_i, x_j). \tag{5}$$

Maximizing (5) is a problem of quadratic programming (QP) [42], which is known to be of cubic computational complexity $\mathcal{O}(N^3)$ with sample size N [12]. For this reason, the adoption of a technique called sequential minimal optimization [37] (SMO) is used for solving Eq. (5). SMO breaks up the large QP problem for SVM training into a series of smallest possible subproblems which can be solved analytically. In future, if real-time constraints might be involved, SVM training may be done incrementally with an online learning algorithm [25].

In this way, working on the data points one by one, it becomes possible to break the process with the so far reached result if a deadline for answering is approaching.

The result (weight vector β) represents the center a of the sphere in terms of an expansion into $\mathcal{H}$:

$$a = \sum_i \beta_i \Phi(x_i). \tag{6}$$

The distance R of the image of an arbitrary point $x \in \mathbb{R}^d$ from $a \in \mathcal{H}$ can be calculated in $\mathbb{R}^d$ by:

$$R^2(x) = \ 1 - 2\sum_i \beta_i k_{\mathcal{G}}(x_i, x) + \sum_{i,j} \beta_i \beta_j k_{\mathcal{G}}(x_i, x_j). \tag{7}$$

Finally, the radius $R_{\mathcal{S}}$ of the sphere $\mathcal{S}$ is determined by the distance to a of an arbitrary support vector as these are mapped right onto the surface. Thus the original feasible region is now modeled as

$$\{x \in \mathbb{R}^d | R(x) \leq R_{\mathcal{S}}\}. \tag{8}$$

3.3. The model of the search space

The model of the feasible region of an arbitrary DER consists of the set s of support vectors and respective weights from β as this is all that is needed for reconstructing the boundary that encloses the feasible region. From β only the non zero components for the support vectors are necessary. We denote this reduced weight vector with w. Formally, the model consists of:

– Set of support vectors (example schedules) $SV = \{x_i \in \mathcal{X} \mid \beta_i \neq 0\}$
– Associated weights: $w = (\beta_1, \dots, \beta_n) \quad \forall \beta \neq 0$
– Some additional parameters: e.g. max. power, cost factor, ...
– Decision function: $R^2(x) = \ 1-2\sum_i w_i k_{\mathcal{G}}(s_i, x) + \sum_{i,j} w_i w_j k_{\mathcal{G}}(s_i, s_j)$ for deciding on feasibility of a given schedule on for comparing two given schedules and deciding on which is nearer to feasibility.
– Feasible region : $\{x | R(x) \leq R_{\mathcal{S}}\}$

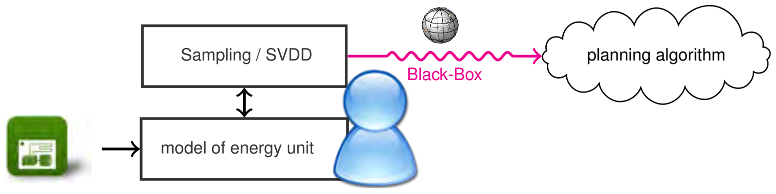


Fig. 1. Integrating the different models into an agent-based energy planning process.

The model-based planning approach that is described later, is currently integrated into a multi-agent simulation of large, future smart grid scenarios. Figure 2 shows the interplay of the involved models from an architectural point of view. As we aim at a system that enables self-organized and market-based control of distributed energy production [34], we go for an agent-based system. The whole system [33, 41, 2] will comprise several types of agents for different tasks. Among them are: market agents for different markets (real power, ancillary services, operating reserve, etc.), agents representing a DER during coalition forming and negotiation at market, or grid agents in charge of checking and assuring grid compatibility. For the rest of the paper, we will focus on the type of agent in charge of controlling a DER. In this way, the search space model represents the feasible part of the possible future behaviour of an energy unit as it is learned from a behaviour model that simulates actual device. On an agent interaction level, only the search space model is used as a surrogate for the unit's behaviour.

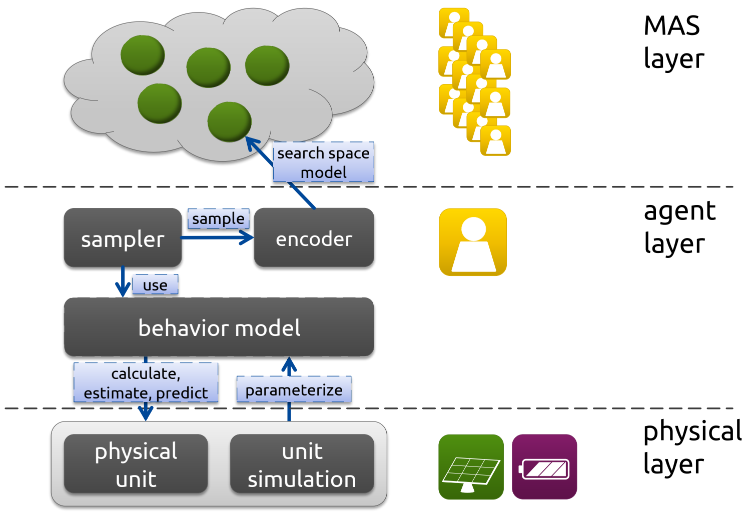


Fig. 2. Using the model inside the multi-agent system hierarchy.

Prior to determining the describing support vectors, the set of training data points $\mathcal{X}$ itself has to be determined. We do this by means of a mathematical model of a DER that can be parameterised with the current (measured) state of the device. This model must be at least able to verify (compliant with all constraints) or falsify (can not be operated) given schedules.

In many cases, it is far too complex to enumerate all possible schedules and to check them against the model. An example with 100 discrete power levels would lead to 100^{96} schedules that would have to be checked for a conventional day-ahead (horizon with 96 periods) scenario. For this reason, one would draw a comparatively small random sample from the set of all schedules.

However, feasible schedules reside in a region that is extremely small compared with $[0, p_{max}]^d$. This fact applies in particular to high dimensional schedules. If, for example, for each period one third of the alternatives is prohibited by constraints, then the ratio of feasible solutions to all solutions amounts to $(\frac{2}{3})^{96} \approx 1.25 \times 10^{-17}$ for the general case of $d = 96$ periods.

Considering load profiles for a whole day, an investigation of our simulation models has shown a proportion of valid load profiles below 10^{-23}. For this reason, it is impossible just to draw random load profiles and check their validity with the model in acceptable time. Hence, we currently draw samples as an successive drawing of period wise guessing.

1. Guess a random power level for just one period.
2. Validate this 1-dimensional schedule with the help of the DER model.
3. If it is valid: Simulate the follow-up state of the DER, re-parameterize the model and goto 1. for determination of the next period.

This approach has the advantage of leading far more likely to guesses of valid schedules. The probability $\mathcal{P}$ for guessing a valid schedule for a single period is already rather high. Allowing for multiple guessing (with number of tries n) results in the even higher probability

$$\mathcal{P}_{(n)} = \sum_{i=1}^{n} B(i|\mathcal{P}, n) = \sum_{i=1}^{n} \binom{n}{i} \mathcal{P}^i (1 - \mathcal{P})^{n-i}, \tag{9}$$

where $\mathcal{P}_{(n)}$ is the probability for at least one successful guess within n tries. Successive guessing then results in an overall probability for successfully guessing a complete schedule of d periods of

$$\mathcal{P}_{(n)}^d = \left(\sum_{i=1}^{n} B(i|\mathcal{P}, n) \right)^d . \tag{10}$$

As an example, let the probability of correctly guessing a valid power load level for a single period be 0.05. Allowing for 100 guesses for each period, then the overall probability for guessing a schedule of 96 periods correctly is still $\mathcal{P}_{(100)}^{96} = 0.5655$, which is sufficiently high in contrast to 10^{-23} (as had been estimated for some of our models).

A major drawback of this approach is that in this way we get a set of schedules that is dominated by the first periods. That means, schedules do not have equal probability for being chosen. This disadvantage is currently deferred for two reasons:

1. We are not primarily interested in statistical properties of the sample or the underlying density. Instead, we want to sound the geometric region where valid schedules reside in.

2. Simulation runs have shown that the principle structures of the scopes of action are nevertheless revealed with this method - as long as merely geometric properties are considered.

An interactive method for a sampler that uses a simulation model that implements the behaviour of an energy unit (c.f. figure 2) is shown in figure 3.

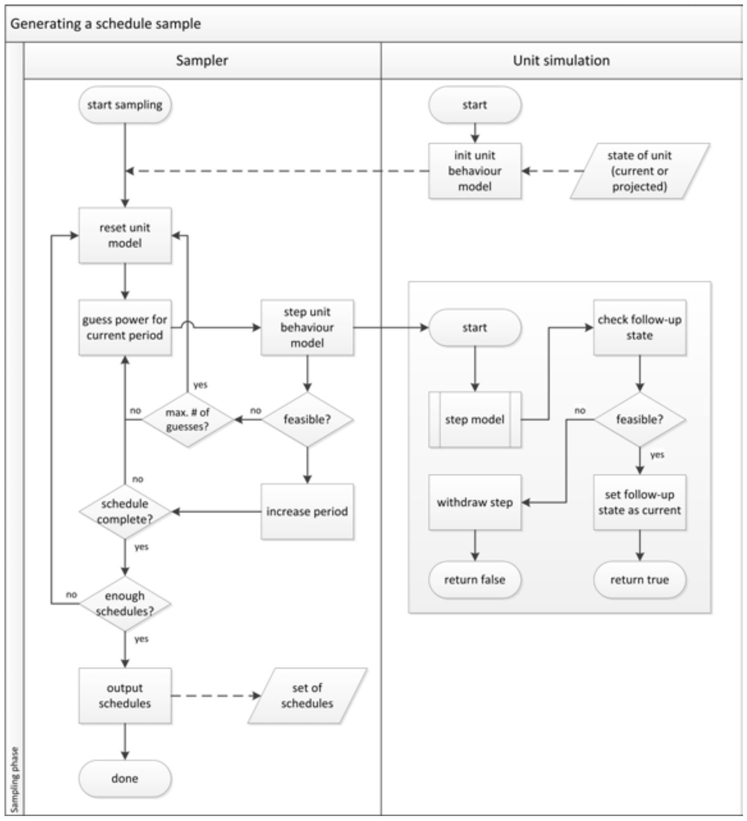


Fig. 3. Sampling process for the training sample prior to learning the search space model.

This process makes use of equation (10) and guesses for a given number of tries a short schedule (usually one period) for the near future until one is found that is actually operable. The behaviour model checks the validity and conditionally calculates the follow-up state resulting from operating the guessed schedule. This step is repeated until a schedule of sufficient length is found. In order to use an appropriate likelihood distribution for our guesses, we a priori make a kernel density estimation of the distribution of operable parts of the schedule [35].

3.4. Constructing solutions from the model

We will now show a way to use this model in a more sophisticated way than as a mere black-box model. We are interested in having a means of finding a nearby feasible schedule next to an arbitrary given schedule. For this purpose, we will harness a function that maps the d-dimensional unit hypercube (representing arbitrary schedules in a scaled scenario) onto the feasible region. In this way, any (in-)feasible schedule will be converted into a feasible one. The construction of this mapping is described in this section.

For our use case, we need a procedure that generates a nearby and feasible solution from any given (likely not feasible) schedule. Nearby in this context means that the distance in solution space between given and near feasible solution is small. This task can be achieved by constructing a mapping that maps every infeasible point from input space into or onto the feasible region. We have tested both approaches. Here, we describe the more general case of mapping into the feasible region that includes the specialized case. In general, this mapping can also be used for transforming the whole optimization problem into an unconstrained one.

Let $\mathcal{F}$ denote the feasible region within the domain of some given optimization problem bounded by an associated set of constraints. It is known, that preprocessing the data by scaling it to $[0,1]^d$ leads to better adaption [19]. Considering optimization problems in the energy sector, rescaling of the domain to $[0,1]^d$ leads to some advantages [7]. For this reason, we here consider scaled domains, too, and denote with $\mathcal{F}_{[0,1]}$ the likewise scaled region of feasible solutions. We want to construct a mapping

$$\begin{aligned} \gamma : [0,1]^d \to \mathcal{F}_{[0,1]} \subseteq [0,1]^d \\ x \mapsto \gamma(x) \end{aligned} \tag{11}$$

that is able to map the unit hypercube $[0,1]^d$ onto the d-dimensional region of feasible solutions that is bounded by a set of arbitrary (maybe nonlinear) constraints. But, instead of directly mapping to $\mathcal{F}_{[0,1]}$ we will go through the kernel space as shown in the following commutative diagram (12).

$$\begin{tikzcd} x \in [0,1]^d \arrow[r, "\hat{\Phi}_\ell"] \arrow[d, "\gamma"'] & \hat{\Psi}_x \in \mathcal{H}^{(\ell)} \arrow[d, "\Gamma_a"] \\ x^* \in \mathcal{F}_{[0,1]} \subseteq [0,1]^d & \tilde{\Psi}_x \in \mathcal{H}^{(\ell)} \arrow[l, "\Phi_\ell^1"] \end{tikzcd} \tag{12}$$

We start with an arbitrary point $x \in [0,1]^d$ from the unconstrained d-dimensional hypercube and map it to an ℓ-dimensional manifold in kernel space that is spanned by the images of the support vectors $s_1 \ldots s_\ell$. After drawing this mapped point towards the sphere in order to pull it into the image of the feasible region, we look for the pre-image of the moved image to get a point from $\mathcal{F}_{[0,1]}$. Thus, we achieve the wanted mapping as a composition of three functions:

$$\gamma = \Phi_\ell^1 \circ \Gamma_a \circ \hat{\Phi}_\ell. \tag{13}$$

We will now look at each step in more detail.

Mapping x to the support vector induced subspace $\mathcal{H}^{(\ell)}$ with an empirical kernel map Let

$$\begin{aligned} \Phi_\ell : \mathbb{R}^d &\to \mathbb{R}^\ell, \\ x &\mapsto k(.,x)|_{\{s1,\ldots,s_\ell\}} \\ &= (k(s_1,x),\ldots,k(s_\ell,x)) \end{aligned} \tag{14}$$

be the empirical kernel map w.r.t. the set of support vectors $\{s_1,\ldots,s_\ell\}$. If Φ_ℓ is modified to

$$\hat{\Phi}_\ell : x \mapsto K^{-\frac{1}{2}}(k(s_1,x),\ldots,k(s_\ell,x)) \tag{15}$$

with $K_{ij} = k(s_i,s_j)$ being the kernel Gram Matrix, then function Eq. 15 maps points x,y from input space to $\mathbb{R}^\ell$, such that $k(x,y) = \hat{\Phi}_\ell(x) \cdot \hat{\Phi}_\ell(y)$ (cf. [39]).

With $\hat{\Phi}_\ell$ we are able to map arbitrary points from $[0,1]^d$ to some ℓ-dimensional space $\mathcal{H}^{(\ell)}$ that contains a projection of the sphere. Again, points from $\mathcal{F}_{[0,1]}$ are mapped into or onto the projected sphere, outside points go outside the sphere and must be moved in $\mathcal{H}^{(\ell)}$ towards the center in the next step in order to draw them into the image of the feasible region.

Re-adjustment in kernel space In general, in kernel space $\mathcal{H}$ the image of the region is represented as a hypersphere $\mathcal{S}$ with center a and radius $R_\mathcal{S}$ (Eq. 7). Points outside this hypersphere are not images of points from $\mathcal{X}$, i.e. in our case, points from $\mathcal{F}_{[0,1]}$ are mapped (by Φ) into the sphere or onto its surface (support vectors), points from outside $\mathcal{F}_{[0,1]}$ are mapped outside the sphere. Actually, using a Gaussian kernel, Φ maps each point into an at most n-dimensional manifold (with sample size n) embedded into infinite dimensional $\mathcal{H}$. In principle, the same holds true for a lower dimensional embedding spanned by ℓ mapped support vectors and the ℓ-dimensional projection of the hypersphere therein.

We now want to pull points from outside the feasible region into that region. As we do have rather a description of the image of the region, we draw images of outside points into the image of the region, i.e. into the hypersphere; precisely into its ℓ-dimensional projection. For this purpose we use

$$\tilde{\Psi}_x = \Gamma_a(\hat{\Psi}_x) = \hat{\Psi}_x + \mu \cdot (a - \hat{\Psi}_x) \cdot \frac{R_x - R_\mathcal{S}}{R_x} \tag{16}$$

to transform the image $\hat{\Psi}_x$ produced in step 1) into $\tilde{\Psi}_x \in \hat{\Phi}_\ell(\mathcal{F}_{[0,1]})$ by drawing $\hat{\Psi}_x$ into the sphere. Alternatively, the simpler version

$$\tilde{\Psi}_x = a + \frac{(\hat{\Psi}_x - a) \cdot R_\mathcal{S}}{R_x} \tag{17}$$

may be used for drawing $\hat{\Psi}_x$ just onto the sphere but with the advantage of not having to estimate parameter $\mu \in [1, R_x]$. Parameter μ allows us to control how far a point is drawn into the sphere ($\mu = 1$ is equivalent to Eq. (17), $\mu = R_x$ draws each point onto the center). In this way, each image is re-adjusted proportional to the original distance from the sphere and drawn into the direction of the center.

Points from the interior are also moved under mapping gamma in order to compensate for additional points coming from the exterior. In this way, the whole unit hypercube is literally squeezed to the form of the feasible region without a too large increasing of the density at the boundary. Though, if the feasible region is very small compared with the hypercube, density at the boundary increases (depending on the choice of μ). On the other hand, the likelihood of an optimum being at the boundary increases likewise. So, this might be a desired effect.

After this procedure we have $\tilde{\Psi}_x$ which is the image of a point from $\mathcal{F}_{[0,1]}$ in terms of a modified weight vector $\tilde{w}^{\Gamma_a}$.

Finding an approximate pre-image As a last step, we will have to find the pre-image of $\tilde{\Psi}_x$ in order to finally get the wanted mapping. A major problem in determining the pre-image of a point from kernel space is that not every point from the span of Φ is the image of a mapped data point [39]. As we use a Gaussian kernel, actually none of our points from kernel space can be related to an exact pre-image except for trivial expansions with only one term [24]. For this reason, we will look for an approximate pre-image whose image lies closest to the given image using an iterative procedure after [30]. In our case (Gaussian kernel), we iterate x^* to find the point closest to the pre-image and define approximation Φ_ℓ^1 by equation

$$x^*_{n+1} = \frac{\sum_{i=1}^{\ell} (\tilde{w}_i^{\Gamma_a} e^{-\|s_i - x^*_n\|^2/2\sigma^2} s_i)}{\sum_{i=1}^{\ell} (\tilde{w}_i^{\Gamma_a} e^{-\|s_i - x^*_n\|^2/2\sigma^2})}. \tag{18}$$

As an initial guess for x^* we take the original point x and iterate it towards $\mathcal{F}_{[0,1]}$. As this procedure is sensitive to the choice of the starting point, it is important to have x as a fixed starting point in order to ensure determinism of the algorithm. This is an essential requirement at least for integration into evolutionary algorithms since the same schedule has to be mapped several times, e.g. during search and again after optimization when the best found solution configuration is finally converted into the solution. Empirically, x has showed up to be a useful guess.

Eventually, we have achieved our goal to map an arbitrary point from $[0, 1]^d$ into the region of feasible solutions described merely by a given set of support vectors and associated weights: x^*_n is the sought after image under mapping γ of x that lies in $\mathcal{F}_{[0,1]}$.

This model and mapping may be used in different ways during optimization. Among these use cases are:

– Repair of infeasible solutions.

– Transformation of an constrained to an unconstrained optimization problem by mapping the whole search space into the feasible region.
– Computational easy classification of an solution's feasibility.
– Compact communication of large sets of operable schedules.

Here, we will harness the capability of repairing infeasible solutions for a distributed optimization approach.

3.5. The distributed greedy algorithm

With the above sketched preliminaries, we are now able to define our optimization algorithm. In order to pay attention to the ongoing decentralization of electricity grid control, it seems way more promising to design the optimization process distributed, too. In addition, the chances for success in finding an exact solution are rather low due to problem size, what makes a heuristic most suitable.

In this sense, we propose the following greedy algorithm for approximately solving optimization problem equation (1). In our optimization scenario, we assume one type of agent: one control agent for each a single energy resource with the following responsibilities/ capabilities:

– Simulating the underlying physical device in order to determine operable example schedules.
– Calculation of the support vector based black-box model.
– Calculation of mapping γ.
– Determining the schedule for the agent's own physical device that minimizes the overall loss.
– Participation in some joint optimization process.

The procedure for optimizing the aggregated schedule is now the one depicted in Fig. 4. Within a group of agents $\mathscr{A}$, one agent is randomly chosen to start the procedure. Here, we assume an agent to be in charge of controlling a DER and to participate in the distributed procedure of determining schedules for each DER such that the aggregated schedule best fits a given objective schedule. An elected initiator initializes the solution with all values to zero. Then, solution improvement begins. The agent adds up all schedules (known from the solution object) from all other agents. This is equivalent to subtracting one's own schedule from the aggregated solution. In a next step, the difference vector Δx of this sum to the desired target schedule is determined. This difference vector Δx represents the optimal schedule for the current agent in the following sense: if the agent would be able to deliver this schedule, the target could be reached exactly. Therefore, the agent now determines the nearest schedule to Δx that is actually operable by the device. This nearby schedule can be easily calculated with the help of the mapping γ that has been described in the previous section. Function γ maps an arbitrary schedule (in our case difference schedule Δx) into the region of feasible schedules and delivers the respective operable schedule that is close to Δx, because it uses the shortest trace to the feasible region to

```
$\mathcal{A} \leftarrow$ List of all agents
**if** is initiator **then**
    $S \leftarrow zeros(n, d)$
**else**
    $S \leftarrow$aggregated schedule
    $S_{new} \leftarrow \gamma(T - (S - S_a))$
    $S \leftarrow S - S_a + S_{new}$
    **if** no stop criterion met **then**
        choose random agent $A \in \mathcal{A}$
        send message with $S$ to $A$
    **else**
        publish solution $S$
    **end if**
**end if**
```

Fig. 4. Greedy algorithm (c.f. [9]) that each agent repeatedly executes for successive solution improvement starting from a zero solution S with S denoting the aggregated overall solution and S_a denoting the individual current contribution of the agent.

move a point. Please note that the distance between sum and target schedule that we minimize by this approach is the Euclidean distance L_2.

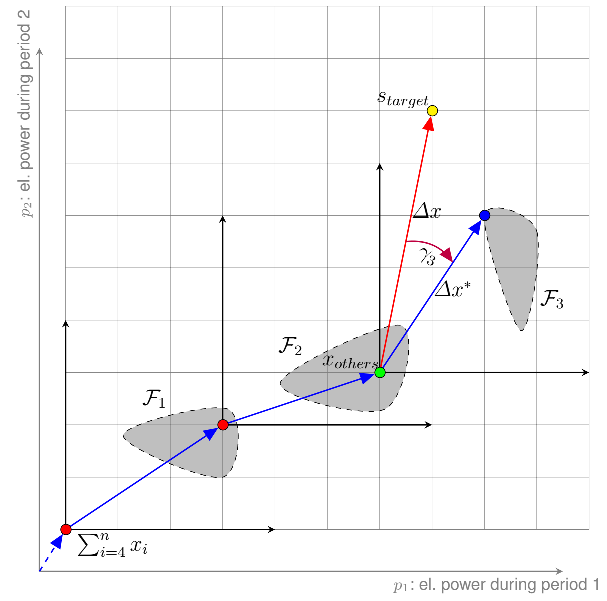
Figure 5 illustrates the approach with showing the situation at two successive iterations. The figure shows a 2-dimensional example problem. In the first step (fig. 5(a)) it is the turn of agent no. 3. Point x_{other} denotes the sum of current schedules of all other agents (from the perspective of agent 3). Vector Δx denotes the difference that is necessary to achieve the target schedule s_{traget} exactly. Because it is usually not necessarily the case that the difference Δx is feasible for the agent's energy unit, Δx is mapped to the feasible region by mapping γ resulting in the nearby schedule Δx^* that is feasible for agent 3. This schedule Δx^* is then taken as the best what agent 3 currently can do and is set as the agent's current schedule.

In the next step (fig. 5(b)) agent 2 becomes active. The sum of all other agents now is different because of the different perspective and due to new schedule of agent 3 from the previous step. Agent 2 does the same steps as previously agent 3 resulting in an updated schedule for agent 3 with a degradation ΔE of the overall error E. The figure also shows how individual schedules are moved to the respective feasible regions (grey areas) to ensure operability of the solution.

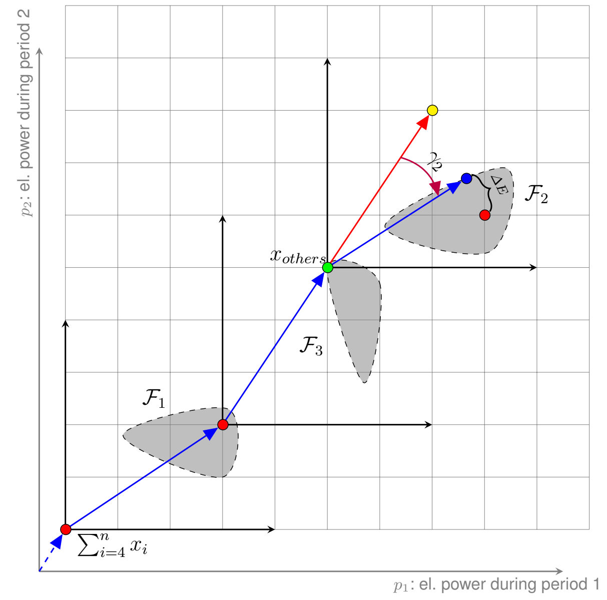
In this way, each DER chooses a schedule that is a compromise of being feasible (automatically ensured by mapping γ) and doing one's own best in bringing forth the overall solution towards the wanted adaption to the target schedule as much as possible each time when it is the respective agents turn.

As a stop criterion, we chose a maximum number of iterations at which the term iteration refers to one execution of the procedure in Fig. 4 by one agent.

Additional objectives could for example be integrated by having different cost indicators added as additional elements to the electrical schedule (in this way,



(a)



(b)

Fig. 5. Base principle of the greedy approach. Part (a) shows an optimization step with an active agent no. 3 doing an improvement by mapping the residual Δx onto his feasible region (grey area). Part (b) shows the follow-up step with an active agent no. 2.

combining schedule and evaluation criteria to a feature vector) and have the relation of schedule and evaluation criteria learned concurrently with the same approach as described here. As an example, this has been done with environmental criteria [6]. Each agent in the greedy optimization approach would then try to reach the electric active power target and a good value for each indicator at the same time, trying to reach a value of 0 for each criterion that has to be minimized and 1 else (provided that all criteria are likewise scaled to $[0,1]$) by taking these additional targets into account when calculating Δx.

By one after another, the overall solution (the aggregated schedule) is successively improved. We have chosen to activate the agents in a random order, but a round robin approach may also do if each agent knows about his successor. In this way, the algorithm is distributed and sequential as only one agent has the token to work at a time. If the objective is to adapt to a given target schedule, the only information that has to be passed around (or made globally available) is the aggregated overall solution (as sum of all local solutions) and the desired target schedule. This is sufficient as each agent may remember his own local schedule that has been determined previously. All other information can be determined by local information.

Clearly, the actual optimization is distributed but sequential. But, the most time consuming part – namely learning the model for the computation of mapping γ – can be done in advance and fully parallel, what in turn allows for faster optimization afterwards without a need for considering constraints anymore.

Finally, we will discuss the ability of the whole algorithm to be run in parallel. In this case, two realizations are possible. First, the process as described above could be further parallelized by making the update calculation asynchronously run. In this way, an agent would first choose and trigger a successive agent and then calculate his own update. A problem here lies in the responsibility for detecting sufficient convergence.

Another approach would be a realization with a central instance that holds the current solution and provides it to all agents. Then, each agent could asynchronously and without any further trigger repeatedly query for the current solution and update the own solution part. Storing the new overall solution should of course be an atomic operation. In this case, the central instance (e.g. the coalition leader) that holds the current solution would also be in charge of deciding on convergence and on bringing the process to a hold.

4. Simulation results

So far, we have tested our approach with several simulated energy resources in different groupings. Among them are: co-generation devices with thermal buffer store and a simulated residential thermal energy demand as well as simulated controllable cooling devices. We will here focus on results from CHP generation. All simulations have been done with power scaled to $[0,1]$. All simulations incorporating a μCHP also encompass the simulation of the respective household that is heated by this μCHP. This implies a simulation of the respective heat demand, heat use, different weather conditions or heat losses by thermal diffusion processes.

For our simulations, we used the model of a modulating μCHP-plant with the following specification:

- Minimum electrical power: 1.3 kW,
- Maximum electrical power: 4.7 kW,
- Minimum thermal power: 4 kW,
- Maximum thermal power: 12.5 kW,
- After shut down, a device has to stay off for at least 2 h.

A modulating CHP is a generator that may vary the level of electrical power output within a certain range resulting in different thermal power output respectively. The relationship between electrical (active) power and thermal power was modeled after Fig. 6. In order to gain more degrees of freedom for varying active power, each CHP is equipped with an 800 ℓ thermal buffer store. Thermal energy consumption is simulated by a model of a detached house with its several heat losses (heater is supposed to keep the indoor temperature on a constant level) and randomized warm water drawing for gaining more diversity among the devices.

For each simulated household, we implemented an agent capable of simulating the CHP (and surroundings and auxiliary devices) on a meso-scale level with energy flows among different model parts but with no technical details. All implementations have been done using the Java programming language. For the implementation of the multi-agent system prototype we used the MASON multi-agent simulator toolkit [26]. The support vector algorithm has been implemented using an adaption of the sequential minimization technique from [37].

All simulations have so far been done with a time resolution of 15 minutes for different forecast horizons. For each simulation, we have run 200 test series with each CHP randomly initialized with different buffer charging levels, temperatures and water drawing profiles. We tested schedules with dimension 8, 16, 32, 48, 64, and 96 periods with groups of 5, 10, 30, 100, 500, 750, and 1000 CHP. For each simulation, we have chosen some random target schedule on a

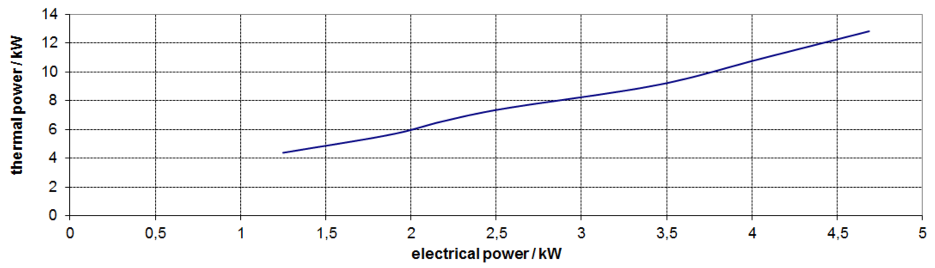


Fig. 6. Relationship between electrical and thermal power for an EcoPower CHP; modified after test bench runs from [45].

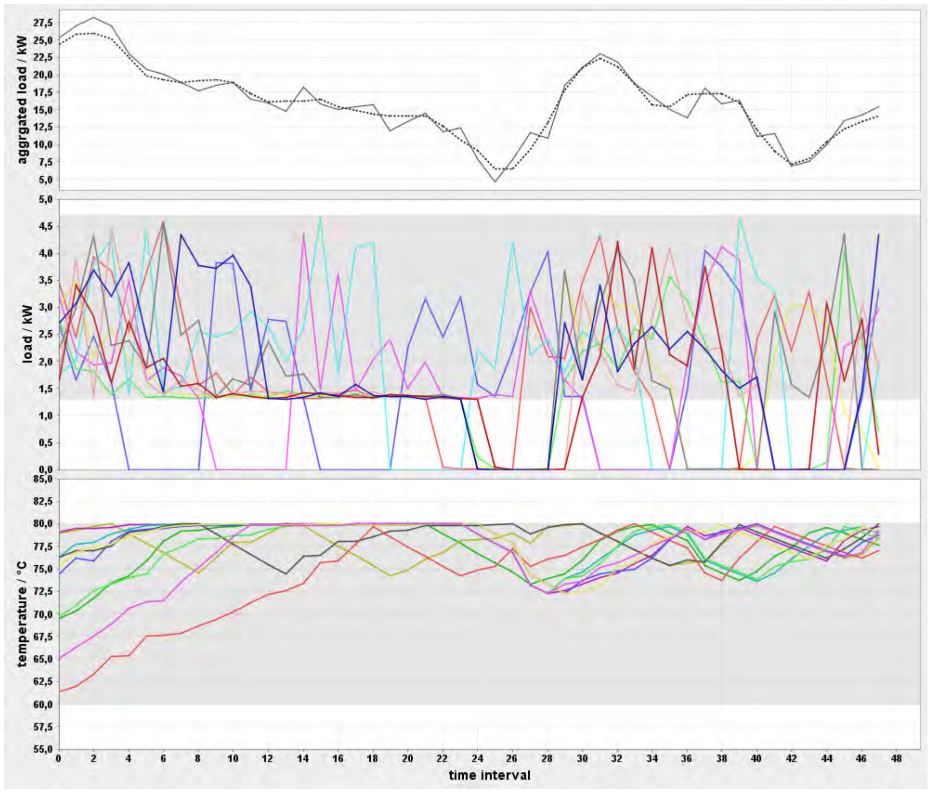


Fig. 7. Optimization result for a winter day scenario with 10 CHP (EcoPower with randomly initialized storage charging) for a time horizon of 48 15-minute intervals.

rather high level. This leads to a rather high charging of the thermal buffer stores on the long run but on the other hand forces the units to go to their limit and demonstrates that the method copes well with the buffer constraints as well.

Figure 7 shows a result (solid line in the top chart) for a group of 10 CHP that try to reach a given objective schedule (dashed line). The resulting schedules for each single CHP are depicted in the middle chart with the allowed active power band for modulation highlighted in grey. The bottom chart shows the temperatures in the thermal buffer store resulting from operating the respective electrical schedules; again with the allowed range highlighted in grey.

The desired objective schedule has been randomly chosen. These schedules have been generated in a way that they are of a reasonable magnitude order according to the capabilities of the optimized CHP, but without any guarantee that a perfect adaption might be achievable.

Figure 8 shows a similar simulation run, but for a time horizon of a whole day. Fig. 9 shows the result for optimizing a larger bunch of 30 CHP. As might have been expected, the result schedule gets closer to the target schedule if

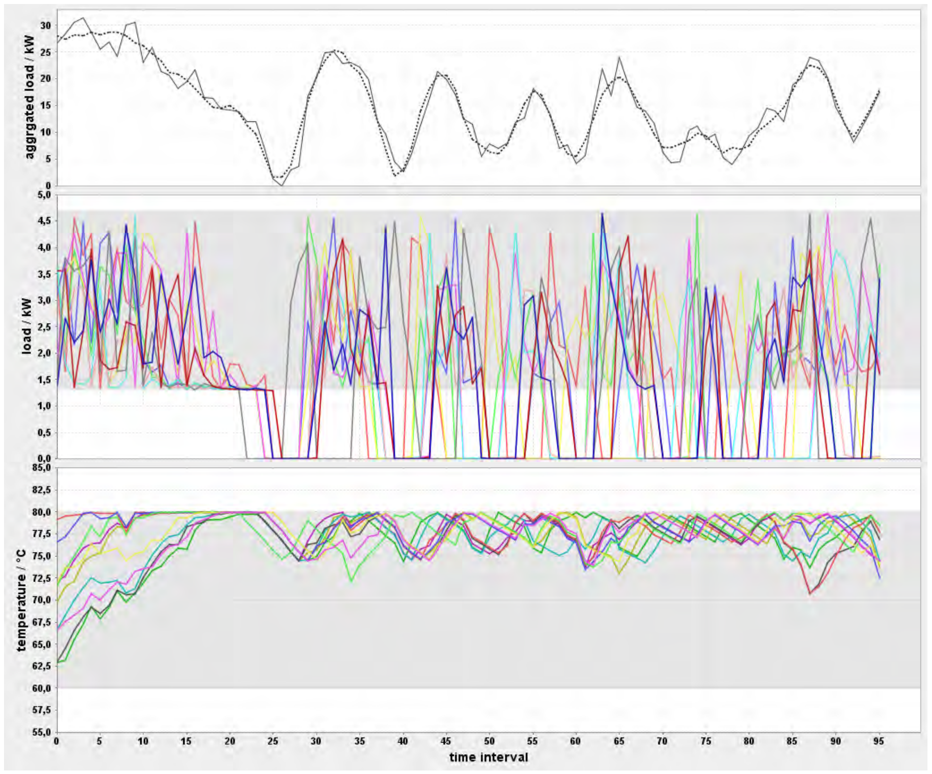


Fig. 8. Optimization result for a spring day scenario with 10 CHP (EcoPower with randomly initialized storage charging) for a time horizon of a whole day in 15-minute intervals.

more CHP are involved. This is mainly due to the availability of more degrees of freedom for the system as a whole.

As a next step, we scrutinized the speed of convergence and convergence behaviour of our algorithm. Figure 10 shows the result of some measuring series. It is noticeable that the fitness (the difference between aggregated and target solution) almost decreases strictly notwithstanding the uncoordinated, heuristic character of the approach. If the algorithm is conducted asynchronously, there are indeed more fluctuations that lead to temporarily degradations of the fitness. This can be easily overcome if the agent in charge the solution rejects solution updates that lead to degradation.

The number of necessary iterations is acceptably small, what can also be seen in Table 1, where some mean CPU time results (Java implementation on Core 2, 3 GHz) for different scenarios are listed. The simulation time t_{sim} reflects the time that is necessary for the whole simulation including the preceding calculations of the set of feasible schedules for each agent, for the calculation of all support vector models and all mapping functions γ on a single machine.

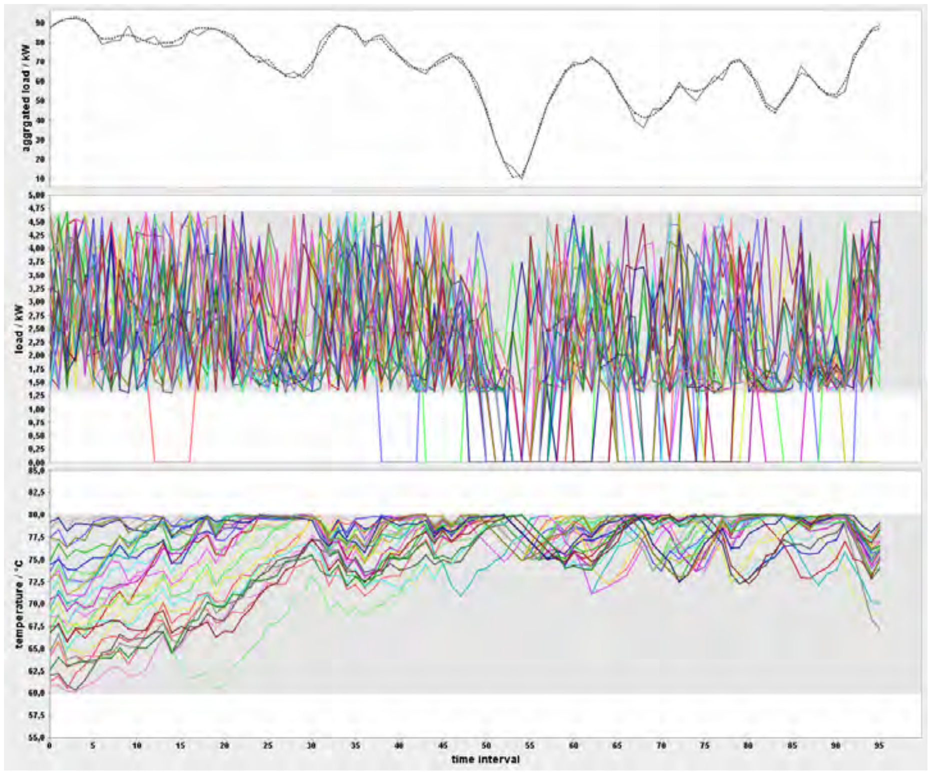


Fig. 9. Optimization result for a scenario with 30 CHP for 96 15-minute intervals. This amounts to a 2880-dimensional search space.

In a distributed productive system these calculations would be done in parallel. Considering the additional complexity that is entailed on solution evaluation, the number of support vectors that make up a model is decisive. Step 1 of calculating the mapping grows quadratically with the number of support vectors (matrix-vector multiplication). Additionally, the number of iterations necessary for finding the pre-image in step 3 has to be considered. Empirically, during our experiments, we observed for instance a mean number of 36.3 ± 26.4 for the case of 8-dimensional schedules to reach convergence with 10^{-6} accuracy.

The time necessary for the mere optimization is comparably small. In order to be able to simulate larger scenarios, we are currently thinking of distributing the simulation, too. So far, we tested a scenario with a group of 50.000 CHP. We parallelized all calculations (simulation of each CHP and optimization of the whole group) on a system with Core i3 processor and were able to complete all calculations (1 day planning horizon) within about 18 minutes. The optimization resulted in a residual error of only about 0.8 percent (compared with the worst case operation that the units could do).

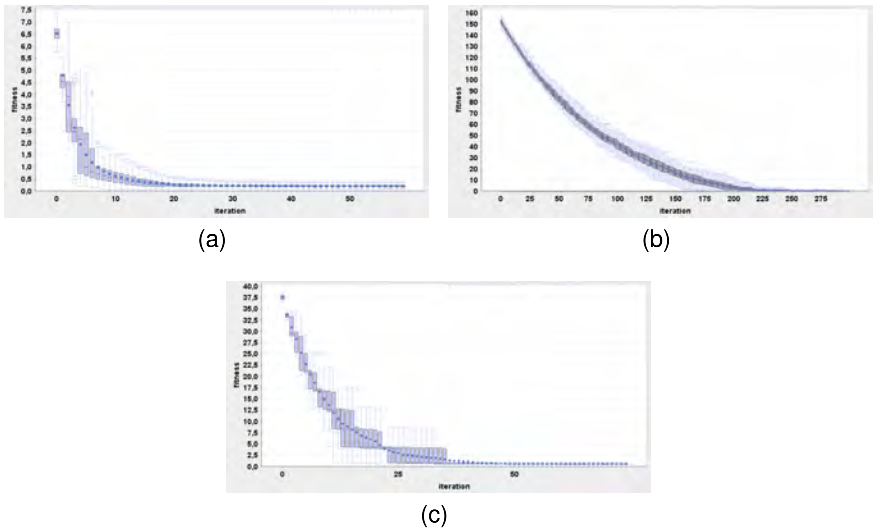


Fig. 10. Convergence for different scenarios: 10(a): 5 CHP and, 8 periods; 10(b): 100 CHP, 8 periods, 10(c): 10 CHP, 96 periods.

Table 1. CPU time for algorithm and simulation regarding different problem sizes. Quality as mean euclidean distance in kW for n_A agents, k iterations and schedules of d intervals of 15 min.

d	n_A	k	t_{sim} / s	t_{opt} / s	QUALITY
8	10	75	4.71 ± 0.23	0.006 ± 0.008	0.054 ± 0.023
8	100	750	45.2 ± 0.74	0.061 ± 0.009	0.045 ± 0.02
32	100	250	382.59 ± 27.24	0.049 ± 0.008	1.05 ± 1.09
96	10	750	251.4 ± 4.5	0.498 ± 0.127	0.049 ± 0.08

Comparing the synchronously (randomized round robin) operated and the fully parallel, asynchronously operated variant of the optimization part, figure 11 shows the convergence behaviour of both approaches by example. Depicted are the results of 500 runs each (the darker the color the more results). The iteration number on the x-axis denotes the number of solution update. Due to the inherent stochasticity of the scenario (250 CHP, 8-dimensional schedule), all errors have been scaled by the individual initial error of the randomly instantiated problem in order to make them comparable. Although the synchronous approach (bottom chart) performs slightly better in term of necessary solution updates, the asynchronous approach takes advantage in (and thus overcompensates this disadvantage by) parallel calculations of the mapping and should thus be preferred.

Finally, figure 12 shows a result from a larger mixed scenario with two different types (in power magnitude) of CHP. Having different DERs in a scenario,

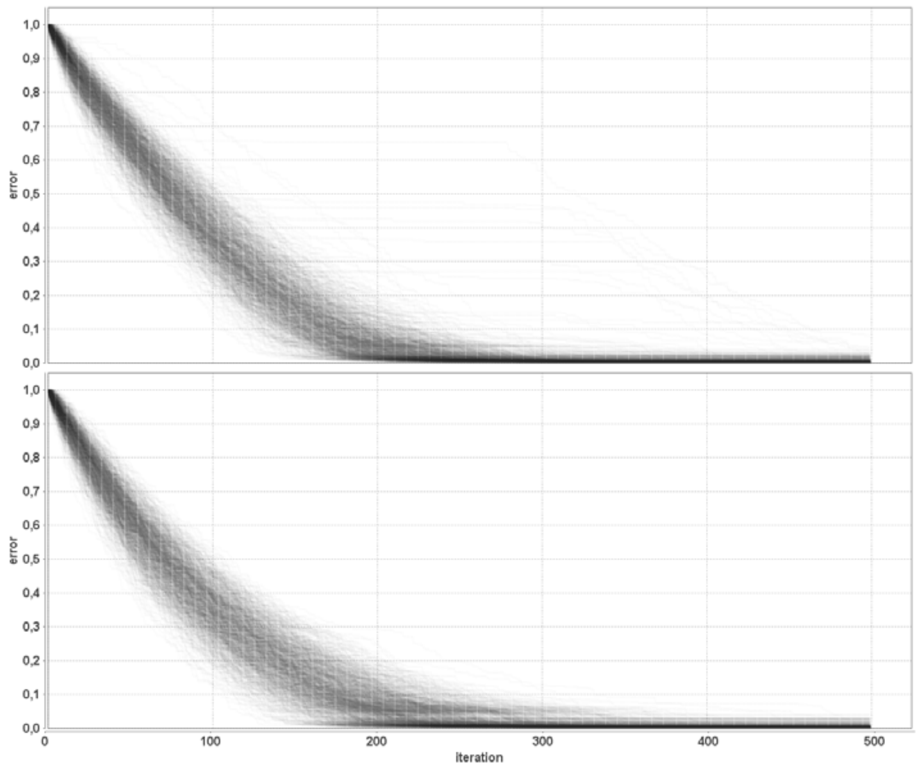


Fig. 11. Comparison of the convergence of synchronous (bottom) and asynchronous (top) operation of the greedy algorithm.

often leads to a better adaption to the target schedule as has also been seen in similar scenarios with a mixture of CHP and refrigerators e.g. in [7].

Another important issue is the question for the size of the search space model that almost exclusively depends on the number of supporting schedules. Figure 13 shows one example with the number of supporting schedules as a function of the time horizon. This example shows that events like increased heat demand in the morning (showering) or in the evening (heater) leads to an escalating increase of information and therefore to a respectively increased number of supporting schedules for appropriate encoding. So far, we have observed that the size of necessary information in fact is a trade-off between accuracy, size and calculation complexity and hence a matter of finding the optimal parameters.

A further issue is the consideration of possible errors that might occur in encoding. It may happen that the enclosing contour adapts not good enough to the point cloud [7]. This may additionally cause two (or more) subregions to be misleadingly considered as one connected region, although they are actually topologically separated by regions of invalid schedules. In both cases, the

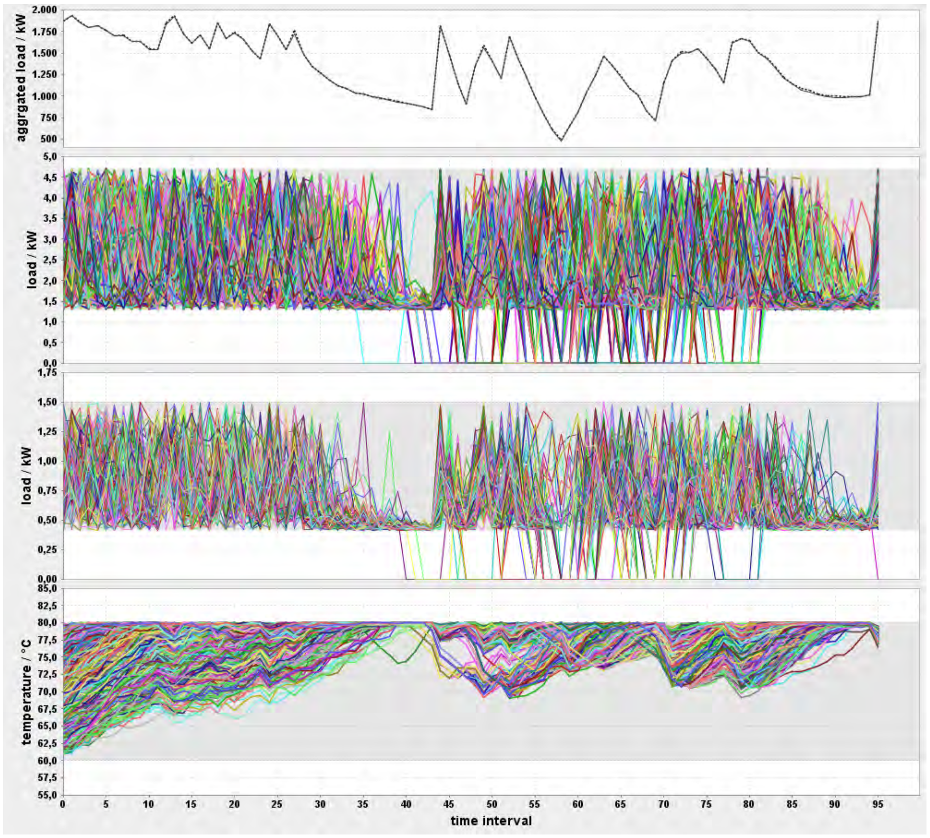


Fig. 12. Optimization result for a scenario with 750 CHP for 96 15-minute intervals. This amounts to a 72000-dimensional search space.

search space model would reconstruct a too large feasible region and consider some schedules incorrectly as operable. Whether this happens and how large these errors are, depends on the choice of parameters.

It is mainly parameter σ in (4), the width of the Gaussian kernel, that determines how smooth the resulting boundary contour adapts to the data. As the boundary adapts closer to the data with a shrinking value for σ, the enclosed region starts separating into separate regions. A smaller value for σ thus leads to a more precise description but on the other hand also to a higher number of support vectors needed for description.

This effect might partly be overcome by fitting parameters of the energy unit simulation model such that the feasible region of alternative schedules is determined too narrow in advance in order to compensate for a too wide description. On the other hand this fact might lead to a too narrow description that misses some schedules near the boundary.

Within the scenarios scrutinized here, most of the encoded schedules are at least partially based on forecasts, i.e. on the anticipated heat demand, weather conditions or similar, so that uncertainty is already inherent. If the error, that is additionally induced by our method is small compared to the already prevalent one, it can be neglected during the optimization process. Error minimization can be achieved by parameter selection.

In any case, when a certain schedule has been chosen by the planning algorithm, it has to be validated by the DER or respectively by the agent with the help of the simulation model before actually operating it. Conditionally, it has be mapped to the nearest valid schedule, where required.

Hence, a crucial point and a remaining open issue is the parameterization of the method. For the previously described simulations, we experimentally determined the best parameters. For this reason, one of our next steps will be to develop a tuning algorithm that is able to (at least semi-) automatically derive optimal parameters (sample size, kernel parameters, SMO parameters, etc.) for an optimal encoding in the sense of the best trade-off between accuracy and number of supporting schedules.

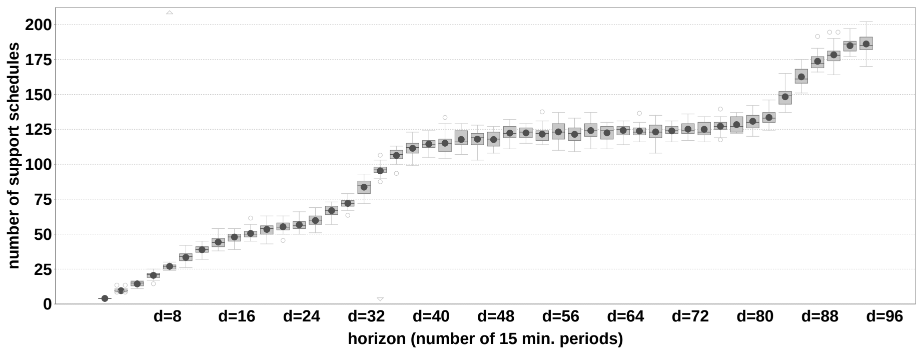


Fig. 13. Number of supporting schedules as a function of the horizon, i.e. the data dimension.

5. Conclusion and further work

We have presented a new approach for distributed optimization and control of distributed energy resources for smart grid scenarios with a large number of controllable entities. This approach is based on two new methodologies:

– A model-based strategy for handling constraints in distributed optimization scenarios that may also be used for finding nearby feasible solutions by harnessing a learned model of the feasible region.
– A well scaling greedy algorithm for harnessing that strategy during the search for an optimal partition of the requested schedule for different DERs.

We have demonstrated that the greedy heuristics scales well with the number of participating devices because the most expensive calculations may be done in parallel in advance by each controllable device.

This work is part of the ongoing smart grid research association Smart Nord (`http://smartnord.de`). This method is currently integrated into a simulation environment for large smart grid scenarios with up to 50.000 electrical units. One goal is to develop an integrated simulation environment based on the mosaik framework [40] for scrutinizing scenarios for market-based planning of active power provision by self-organized coalitions with optimization and re-scheduling capabilities [33]. At the same time ancillary services (e.g. for frequency or voltage stability) are integrated.

Clearly, building such large and diverse scenarios involves the integration of many more types of distributed energy resources. So far, we are planning to integrate among others models for photovoltaic panels (limited controllability), heat pumps, co-generation, night storage heater, white goods (fridge, dishwasher, etc.), batteries, air condition as well as non controllable resources like wind energy.

From this point of view it becomes clear that a model-based approach for the integration of all these different energy units (and the integration of future, yet unknown ones) is indispensable. So far, we paved the way from the automatic conversion of the scope of action of an energy unit or its simulation model to a standard search space model that can be easily integrated into planning and optimization algorithms. In this way, it now becomes easy to commonly integrate a diverse set of different models jointly into distributed algorithms.

Due to this abstraction by search space model and mapping, all energy units become indistinguishable for algorithms and may thus be accessed always in the same, standardized way.

Acknowledgments. The Lower Saxony research network 'Smart Nord' acknowledges the support of the Lower Saxony Ministry of Science and Culture through the Niedersächsisches Vorab grant programme (grant ZN 2764).

References

1. Bary, C.: Coincidence-factor reationships of electric-service-load characteristics. American Institute of Electrical Engineers, Transactions of the 64(9), 623–629 (1945)
2. Beer, S., Appelrath, H.J., Sonnenschein, M.: Towards a self–organization mechanism for agent associations in electricity spot markets. In: Informatik 2011 - Workshop IT für die Energiesysteme der Zukunft (10 2011)
3. Ben-Hur, A., Siegelmann, H.T., Horn, D., Vapnik, V.: Support vector clustering. Journal of Machine Learning Research 2, 125–137 (2001)
4. Blank, M., Gerwinn, S., Krause, O., Lehnhoff, S.: Support vector machines for an efficient representation of voltage band constraints. In: Innovative Smart Grid Technologies. IEEE PES (2011)

5. Born, F.: Aiding Renewable Energy Integration Through Complimentary Demand-supply Matching. University of Strathclyde (2001), http://books.google.de/books?id=w0BCHQAACAAJ
6. Bremer, J.: Ontology based description of der's learned environmental performance indicators. In: Donnellan, B., Lopes, J.P., Martins, J., Filipe, J. (eds.) Proceedings of the 1st International Conference on Smart Grids and Green IT Systems – Smart-Greens 2012. pp. 107–112. SciTePress, Porto, Portugal (04 2012)
7. Bremer, J., Rapp, B., Sonnenschein, M.: Encoding distributed search spaces for virtual power plants. In: IEEE Symposium Series in Computational Intelligence 2011 (SSCI 2011). Paris, France (4 2011)
8. Bremer, J., Rapp, B., Sonnenschein, M.: Including Environmental Performance Indicators into Kernel based Search Space Representations. Information Technologies in Environmental Engineering (ITEE 2011) (2011)
9. Bremer, J., Sonnenschein, M.: A distributed greedy algorithm for constraint-based scheduling of energy resources. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Federated Conference on Computer Science and Information Systems - Fed-CSIS 2012, Wroclaw, Poland, 9-12 September 2012, Proceedings. pp. 1285–1292 (2012)
10. Bremer, J., Rapp, B., Sonnenschein, M.: Support vector based encoding of distributed energy resources' feasible load spaces. In: IEEE PES Conference on Innovative Smart Grid Technologies Europe. Chalmers Lindholmen, Gothenburg, Sweden (2010)
11. Brusan, A.: Very short literature survey from supervised learning to surrogate modeling. CoRR abs/1203.4788 (2012)
12. Chu, C.S., Tsang, I.W., Kwok, J.T.: Scaling up support vector data description by using core-sets. In: Proceedings of 2004 IEEE International Joint Conference on Neural Networks. vol. 1, pp. 430–435 (2004)
13. Evangelista, P., Embrechts, M., Szymanski, B.: Taming the curse of dimensionality in kernels and novelty detection. In: Abraham, A., de Baets, B., Kppen, M., Nickolay, B. (eds.) Applied Soft Computing Technologies: The Challenge of Complexity, Advances in Soft Computing, vol. 34, pp. 425–438. Springer Berlin Heidelberg (2006), http://dx.doi.org/10.1007/3-540-31662-0_33
14. Franch, T., Scheidt, M., Stock, G.: Current and future challenges for production planning systems. In: Kallrath, J., Pardalos, P.M., Rebennack, S., Scheidt, M., Pardalos, P.M. (eds.) Optimization in the Energy Industry, pp. 5–17. Energy Systems, Springer Berlin Heidelberg (2009)
15. Gano, S.E., Kim, H., Brown II, D.E.: Comparison of three surrogate modeling techniques: Datascape, kriging, and second order regression. In: Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, AIAA-2006-7048. Portsmouth, Virginia (2006)
16. Gatterbauer, W.: Economic efficiency of decentralized unit commitment from a generator's perspective. In: Ilic, M. (ed.) Engineering Electricity Services of the Future. Springer (2010)
17. Guan, X., Zhai, Q., Papalexopoulos, A.: Optimization based methods for unit commitment: Lagrangian relaxation versus general mixed integer programming. vol. 2, p. 1100 Vol. 2 (2003), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1270468
18. Hinrichs, C., Lehnhoff, S., Sonnenschein, M.: A Decentralized Heuristic for Multiple-Choice Combinatorial Optimization Problems. In: Operations Research 2012 – Selected Papers of the International Conference on Operations Research (OR

2012). Springer, Hannover, Germany (2013), http://www-ui.informatik.uni-oldenburg.de/download/Publikationen/HLS12.pdf

19. Juszczak, P., Tax, D., Duin, R.P.W.: Feature scaling in support vector data description. In: Deprettere, E., Belloum, A., Heijnsdijk, J., van der Stappen, F. (eds.) Proc. ASCI 2002, 8th Annual Conf. of the Advanced School for Computing and Imaging. pp. 95–102 (2002)
20. Kamper, A., Esser, A.: Strategies for decentralised balancing power. In: A. Lewis, S. Mostaghim, M.R. (ed.) Biologically-inspired Optimisation Methods: Parallel Algorithms, Systems and Applications, pp. 261–289. No. 210 in Studies in Computational Intelligence, Springer, Berlin, Heidelberg (Juni 2009), http://dx.doi.org/10.1007/978-3-642-01262-4
21. Kamphuis, R., Warmer, C., Hommelberg, M., Kok, K.: Massive coordination of dispersed generation using powermatcher based software agents. In: 19th International Conference on Electricity Distribution (May 2007)
22. Kok, K., Derzsi, Z., Gordijn, J., Hommelberg, M., Warmer, C., Kamphuis, R., Akkermans, H.: Agent-based electricity balancing with distributed energy resources, a multiperspective case study. Hawaii International Conference on System Sciences 0, 173 (2008)
23. Kramer, O.: A review of constraint-handling techniques for evolution strategies. Appl. Comp. Intell. Soft Comput. 2010, 1–19 (January 2010)
24. Kwok, J., Tsang, I.: The pre-image problem in kernel methods. Neural Networks, IEEE Transactions on 15(6), 1517–1525 (2004)
25. Laskov, P., Gehl, C., Krüger, S., Müller, K.: Incremental support vector learning: Analysis, implementation and applications. Journal of Machine Learning Research 7, 1906–1936 (2006)
26. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: A multiagent simulation environment. Simulation 81(7), 517–527 (Jul 2005), http://dx.doi.org/10.1177/0037549705058073
27. Lukovic, S., Kaitovic, I., Mura, M., Bondi, U.: Virtual power plant as a bridge between distributed energy resources and smart grid. Hawaii International Conference on System Sciences 0, 1–8 (2010)
28. Mao, Y., Li, M.: Optimal reactive power planning based on simulated annealing particle swarm algorithm considering static voltage stability. In: Proceedings of the 2008 International Conference on Intelligent Computation Technology and Automation - Volume 01. pp. 106–110. ICICTA '08, IEEE Computer Society, Washington, DC, USA (2008), http://dx.doi.org/10.1109/ICICTA.2008.427
29. Mihailescu, R.C., Vasirani, M., Ossowski, S.: Dynamic coalition adaptation for efficient agent-based virtual power plants. In: Proceedings of the 9th German conference on Multiagent system technologies. pp. 101–112. MATES'11, Springer-Verlag, Berlin, Heidelberg (2011)
30. Mika, S., Schölkopf, B., Smola, A., Müller, K.R., Scholz, M., Rätsch, G.: Kernel PCA and de-noising in feature spaces. In: Proceedings of the 1998 conference on Advances in neural information processing systems II. pp. 536–542. MIT Press, Cambridge, MA, USA (1999)
31. Myers, R.H., Montgomery, D.C.: Response Surface Methodology: Process and Product in Optimization Using Designed Experiments. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1995)
32. Neddermeijer, H.G., van Oortmarssen, G.J., Piersma, N., Dekker, R.: A framework for response surface methodology for simulation optimization. In: Proceedings of the 32nd conference on Winter simulation. pp. 129–136. WSC '00, So-

ciety for Computer Simulation International, San Diego, CA, USA (2000), http://dl.acm.org/citation.cfm?id=510378.510401

33. Nieße, A., Lehnhoff, S., Tröschel, M., Uslar, M., Wissing, C., Appelrath, H.J., Sonnenschein, M.: Market–based self–organized provision of active power and ancillary services. IEEE (06 2012)
34. Nieße, A., Lehnhoff, S., Tröschel, M., Uslar, M., Wissing, C., Appelrath, H.J., Sonnenschein, M.: Market-based self-organized provision of active power and ancillary services: An agent-based approach for smart distribution grids. In: COMPENG. pp. 1–5. IEEE (2012)
35. Parzen, E.: On estimation of a probability density function and mode. The Annals of Mathematical Statistics 33(3), pp. 1065–1076 (1962), http://www.jstor.org/stable/2237880
36. Pereira, J., Viana, A., Lucus, B., Matos, M.: A meta-heuristic approach to the unit commitment problem under network constraints. International Journal of Energy Sector Management 2(3), 449–467 (2008)
37. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Advances in Kernel Methods. pp. 185–208. MIT press (1999)
38. Ramchurn, S.D., Vytelingum, P., Rogers, A., Jennings, N.R.: Agent-based control for decentralised demand side management in the smart grid. In: Sonenberg, L., Stone, P., Tumer, K., Yolum, P. (eds.) AAMAS. pp. 5–12. IFAAMAS (2011)
39. Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K.R., Rätsch, G., Smola, A.: Input space vs. feature space in kernel-based methods. IEEE Transactions on Neural Networks 10(5), 1000–1017 (1999)
40. Schütte, Steffen; Scherfke, S.S.M.: mosaik – smart grid simulation api. In: Donnellan, B., Lopes, J.P., Martins, J., Filipe, J. (eds.) Proceedings of the 1st International Conference on Smart Grids and Green IT Systems – SmartGreens 2012. SciTePress, Porto, Portugal (04 2012)
41. Sonnenschein, M., Appelrath, H.J., Hofmann, L., Kurrat, M., Lehnhoff, S., Mayer, C., Mertens, A., Uslar, M., Nieße, A., Tröschel, M.: Dezentrale und selbstorganisierte koordination in smart grids. In: VDE-Kongress 2012 Smart Grid Intelligente Energieversorgung der Zukunft. VDE (11 2012)
42. Tavakkoli, A., Nicolescu, M., Nicolescu, M., Bebis, G.: Incremental svdd training: Improving efficiency of background modeling in videos. In: Cristea, P. (ed.) Signal and Image Processing. Acta Press, Calgary, Canada (2008)
43. Tax, D.M.J., Duin, R.P.W.: Data domain description using support vectors. In: ESANN. pp. 251–256 (1999)
44. Tax, D.M.J., Duin, R.P.W.: Support vector data description. Mach. Learn. 54(1), 45–66 (2004)
45. Thomas, B.: Mini-Blockheizkraftwerke: Grundlagen, Gerätetechnik, Betriebsdaten. Vogel Buchverlag (2007)
46. Tröschel, M., Appelrath, H.J.: Towards reactive scheduling for large-scale virtual power plants. In: Braubach, L., van der Hoek, W., Petta, P., Pokahr, A. (eds.) MATES. Lecture Notes in Computer Science, vol. 5774, pp. 141–152. Springer (Sep 2009)
47. Xiong, W., Li, M.j., Cheng, Y.l.: An improved particle swarm optimization algorithm for unit commitment. In: Proceedings of the 2008 International Conference on Intelligent Computation Technology and Automation - Volume 01. pp. 21–25. ICICTA '08, IEEE Computer Society, Washington, DC, USA (2008), http://dx.doi.org/10.1109/ICICTA.2008.363

Jörg Bremer is research assistant at the Department for Computing Science at the Carl von Ossietzky University Oldenburg, Germany. He recieved a Diploma (Environmental Informatics) in the field of agent based simulations of household energy consumption in decentralized scenarios (2006). He has been working on several projects on energy management and computational intelligence in smart grids. In addition, he has been working as a research assistant at the Department for Business Information Systems and at the OFFIS Institute for Information Technology in Oldenburg. He also teaches at the University of Oldenburg in the field of decentralized energy systems. Currently, Mr Bremer is a PhD Student at the chair of Prof. Sonnenschein.

Michael Sonnenschein is professor of Computer Science at the Carl von Ossietzky University of Oldenburg, Germany. He studied Computer Science and Mathematics at the Aachen University of Technology (Diploma 1979); PhD in computer 1983, Habilitation in Computer Science 1991, both at Aachen University of Technology. Since 1991 he is professor for Computer Science at Oldenburg University. As a member of the executive board energy of the OFFIS Institute for Information Technology he headed several projects on energy management in smart grids. His research interests include techniques for modelling, simulation, and heuristic optimization in environmental applications, particularly in smart grids.

Received: March 4, 2013; Accepted: August 3, 2013.

CIP – Каталогизација у публикацији
Народна библиотека Србије, Београд

004

COMPUTER Science and Information Systems : the International journal / Editor-in-Chief Mirjana Ivanović. – Vol. 10, No 4 (2013) - . – Novi Sad (Trg D. Obradovića 3): ComSIS Consortium, 2013 - (Belgrade : Sigra star). –30 cm

Polugodišnje. – Tekst na engleskom jeziku

ISSN 1820-0214 = Computer Science and Information Systems
COBISS.SR-ID 112261644

Cover design: V. Štavljanin
Printed by: Sigra star, Belgrade

ComSIS Computer Science and Information Systems
Volume 10, Number 4, Special Issue, October 2013

Contents

ISSN: 1820-0214