Computer Science and Information Systems 16(2):657–688 https://doi.org/10.2298/CSIS181010014B

Automated Two-phase Business Model-driven Synthesis of Conceptual Database Models*

Drazen Brdjanin, Danijela Banjac, Goran Banjac, and Slavko Maric

University of Banja Luka, Faculty of Electrical Engineering
Patre 5, 78 000 Banja Luka, Republic of Srpska, Bosnia and Herzegovina
{drazen.brdjanin, danijela.banjac, goran.banjac, slavko.maric}@etf.unibl.org

Abstract. Existing approaches to business process model-driven synthesis of data models are characterized by a direct synthesis of a target model based on source models represented by concrete notations, where the synthesis is supported by monolithic (semi)automatic transformation programs. This article presents an approach to automated two-phase business process model-driven synthesis of conceptual database models. It is based on the introduction of a domain specific language (DSL) as an intermediate layer between different source notations and the target notation, which splits the synthesis into two phases: (i) automatic extraction of specific concepts from the source model and their DSL-based representation, and (ii) automated generation of the target model based on the DSL-based representation of the extracted concepts. The proposed approach enables development of modular transformation tools for automatic synthesis of the target model based on business process models represented by different concrete notations. In this article we present an online generator, which implements the proposed approach. The generator is implemented as a web-based, service-oriented tool, which enables automatic generation of the initial conceptual database model represented by the UML class diagram, based on business models represented by two concrete notations.

Keywords: BPMN, business process model, conceptual database model, domain specific language, extractor, generator, model-driven, service-oriented, UML.

1. Introduction

Data models are essential to any information system. The process of data modeling is not straightforward. It is often time-consuming and requires many iterations before the final model is obtained. Therefore, automatic data model design is very appealing and has been the subject of research for many years.

The majority of existing approaches to automated data model design are linguistics-based, since natural languages are commonly used for requirements specifications. However, their utilization is questionable for languages with complex morphology. Currently, there are several alternative approaches taking collections of forms or models (graphically specified requirements) as the basis for automated data model design, instead of textual specifications.

* This article constitutes an extended version of the conference paper entitled "*An Online Business Process Model-driven Generator of the Conceptual Database Model*" presented at the *8th International Conference on Web Intelligence, Mining and Semantics – WIMS'18*, June 25-27, 2018, Novi Sad, Serbia.

The idea of model-driven design of data models is already thirty years old [15]. However, the fully automatic *model-driven synthesis of the data model* (MDSDM) is still the subject of extensive research. In existing literature there is only a small number of papers presenting the implemented automatic model-driven generator of the target data model with corresponding evaluation results, while the great majority only present modest achievements in (semi)automated, or even manual, data model synthesis.

The existing MDSDM approaches are characterized by the direct data model synthesis based on source models represented by some concrete notation such as BPMN[1] or UML[2] activity diagram, where the synthesis is supported by monolithic transformation programs. Direct synthesis introduces dependency of the generation process from the source notation, because different source notations require different generators. Therefore, these generators depend on changes of notations, which are caused by metamodel changes and/or vendor specific implementations. The existing tools are also platform-dependent since they are mainly implemented as transformation programs deployed in some development platform (mainly Eclipse-based). Overcoming these shortcomings motivates our research with the following objectives: (1) define an approach to business process model-driven synthesis of conceptual database models, which enables the development of modular transformation tools for automatic synthesis of the target model based on source models represented by different notations with reduced dependency of the generation process from the source notations; (2) implement an online generator of conceptual database models based on the proposed approach, which facilitates database design as well as development of web-based environments for automatic business process model-driven database design.

In this article we present an approach that fulfills the aforementioned research objectives. It is based on the introduction of a *domain specific language* (DSL) as an intermediate layer between different source notations representing *business process models* (BPMs) and the target notation representing the *conceptual database model* (CDM). This simple DSL, called *Business Model Representation Language* (BMRL), is used to represent BPM concepts (such as participants, objects, tasks, etc.) having semantic potential for automatic CDM synthesis. With the introduction of BMRL, the CDM synthesis is split into two phases. In the first phase, specific concepts are to be extracted from the source BPM and represented by BMRL. Such an extraction can be easily implemented for different source notations. In this article we completely present extraction from BPMN models. In the second phase, the target CDM is to be generated based on the BMRL-based representation of the extracted BPM concepts. This generator is to implement a rather complex set of rules, which are dependent on simple and unique BMRL concepts, but independent of different source notations. This constitutes the first main contribution of the article.

The second main contribution of the article is related to the development and presentation of an online, publicly available, service-oriented CDM generator, which implements the proposed two-phase approach to BPM-driven CDM synthesis. Its usage could be twofold: (i) developers are able to implement their own applications consuming the exposed web service, (ii) database designers are able to use the implemented client application aimed at BPM-driven CDM synthesis.

---

[1] Business Process Model and Notation [50]

[2] Unified Modeling Language [51]

This article constitutes an extended and complete version of two closely related conference papers. The first paper [17], which proposed the first ideas about two-phase data model synthesis, was presented at the *7th International Conference on Model and Data Engineering – MEDI'17*. The second paper [18], which presented implementation of the first online two-phase BPM-driven CDM generator, was presented at the *8th International Conference on Web Intelligence, Mining and Semantics – WIMS'18*. The content of both conference papers is merged and expanded by: (1) a detailed presentation of the related work, (2) detailed presentation of the proposed approach with particular focus on BPMN as the starting notation, (3) recent achievements related to the implementation of the online generator applying the proposed approach, and (4) evaluation of the approach and implemented online generator.

The article is structured as follows. After the introduction, the second section presents the related work. The principles of the two-phase BPM-driven CDM synthesis are considered in the third section. The fourth section presents the semantic capacity of BPMs for automatic CDM synthesis, as a basis for DSL specification. The first phase of the CDM synthesis is presented in the fifth section, while the second phase is presented in the sixth section. The seventh section presents the implementation of the online generator. An illustrative example of the two-phase BPM-driven CDM synthesis is presented in the eighth section. The proposed approach is evaluated in the ninth section. Finally, the last section concludes the article.

2. Related Work

The survey [15] shows that existing MDSDM approaches, with respect to the primary focus of the source notation, can be classified as: *function-oriented*, *process-oriented*, *communication-oriented* and *goal-oriented*. The chronological overview of the existing MDSDM approaches, grouped by the source notation, is given in Fig. 1. Different marks are used to differentiate the source model completeness, which can be *complete* or *partial* (partial source model contains a single diagram, although a real model contains a finite set of diagrams). The figure also shows the level of automatisation for the approaches, which can be *manual* (not supported by any software tool), *semiautomatic* (supported by a tool, but designer's assistance is still required), or *automatic* (without designer's assistance). The arrows are used to emphasise the related papers presenting the improvements in the same approach.

Our approach belongs to the process-oriented approaches. As shown in Fig. 1, *process-oriented models* (POMs) constitute the largest category of models used as a source for MDSDM. Although the first data model synthesis based on a POM (A-graph) was proposed by Wrycza [64] in 1990, the boom of these approaches was influenced by the development of metamodel-based notations, particularly UML AD (Activity Diagram) and BPMN, as well as model-to-model transformation languages ATL[3] and QVT[4].

The survey [15] shows that POMs, used as a basis for data model synthesis, have been represented by seven different notations: BPMN, UML AD, Petri Net, RAD (Role Activity Diagram), GRAPES-BM/TCD, EPC (Event-driven Process Chain) and A-graph. Although there are more than 40 papers considering the POM-based MDSDM, the

[3] ATLAS Transformation Language [38]
[4] Query/View/Transformation [49]

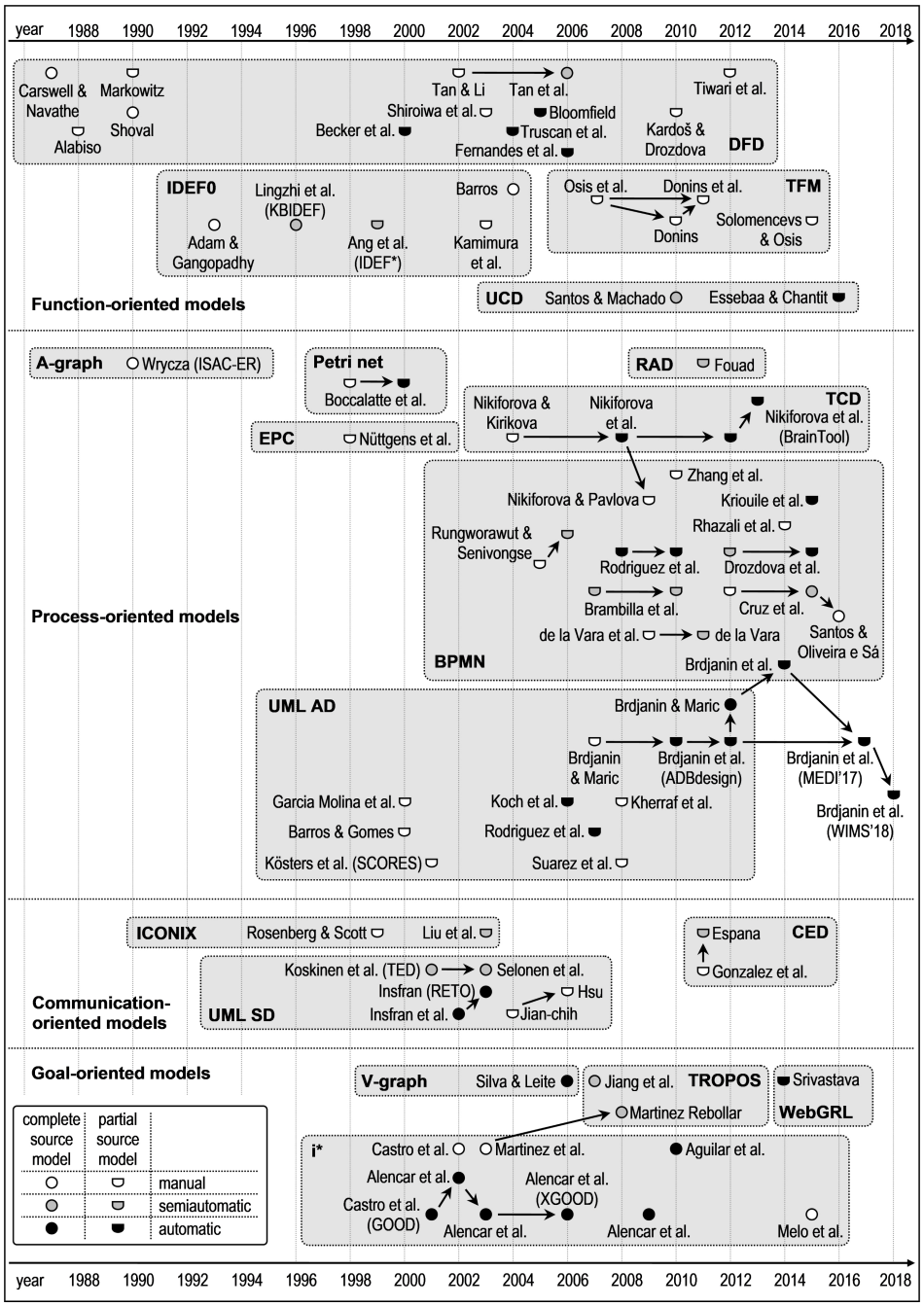


Fig. 1. Overview of existing MDSDM approaches

survey shows that the semantic capacity of POMs has still not been sufficiently identified to enable automatic synthesis of a complete data model. Furthermore, the majority of the approaches enable semiautomatic generation of the target model with modest completeness and precision. The majority of all POM-based approaches are also based on guidelines and informal rules that do not enable automatic MDSDM.

The BPMN is the most commonly used source notation for POM-based MDSDM. Among almost 20 papers, there are three QVT-based proposals [53, 54, 42], but with modest achievements in the automatic generation of analysis level class diagrams. There is an XSLT-based proposal [29] for automatic generation, as well as several proposals [55, 11, 12, 28, 30] for semiautomatic generation of class diagrams. A MDSDM based on BPMN is also considered in [65, 48, 30], but without implementation. The formal rules for automatic CDM synthesis based on BPMN are presented in [21, 56], and partially in [26, 27]. Other papers consider only the guidelines that do not enable automatic synthesis. A set of interrelated BPMs is considered in [27, 56], but the approaches are not implemented.

Among more than ten papers using UML AD as the source notation, only [23] presents an automatic CDM generator (named ADBdesign) based on the complete source model. Several papers [39, 40, 52, 16, 13, 22, 14] present the automatic, mainly ATL- and QVT-based, data model generation based on the incomplete source model, but with modest completeness and precision, while the others present only manual data model derivation.

There are also several related papers proposing the usage of TCD notation as a starting point for MDSDM, initially through an intermediate model, while [46] presents the BrainTool generator, which generates the data model directly from the TCD. However, like the majority, they do not consider the complete source model. Among the other POM-based approaches, there are only two papers [10, 34] presenting software tools for the (semi)automatic data model generation based on the partial source model.

The survey [15] shows that *function-oriented models* (FOMs), used as a basis for data model synthesis, have been represented by four different notations: DFD (Data Flow Diagram), SADT/IDEF0, TFM (Topological Functioning Model), and UML UCD (Use Case Diagram). Although the first ideas about the FOM-based MDSDM appeared in the second half of the 1980s, the survey shows that the semantic capacity of FOMs has not been sufficiently identified to enable automatic synthesis of the complete target data model. The large majority of the approaches are based on guidelines and informal rules and take an incomplete source model as the basis for data model synthesis. The automatic data model generation is presented in [8, 62, 33, 9, 32], while the semiautomatic generation is presented in [61, 43, 6, 57].

The survey [15] shows that *goal-oriented models* (GOMs), used as a starting point for data model synthesis, have been represented by the i* notation and some i*-originated notations like TROPOS, V-graph, and WebGRL. The automatic data model synthesis (to some extent) is presented in [25, 4, 3, 5, 59, 37, 45, 2, 1, 60]. The GOM-based approaches use complete source models.

The smallest number of models used for MDSDM are *communication-oriented models* (COMs). They have been represented by three different notations: UML SD (Sequence Diagram), CED (Communicative Event Diagram) and ICONIX (Robustness Diagram). UML SD is used in the majority of COM-based MDSDM approaches. The automatic data model synthesis is presented in [35, 36], while the semiautomatic synthesis is presented in [41, 58, 44, 31].

The large majority of all proposed MDSDM approaches are not evaluated at all (more than 90% according to [15]). Most of the papers reporting evaluation results mainly focus on approach usability, but not on the qualitative/quantitative assessment of the implemented tools or generated data models. The GOM-based approaches are not evaluated. Only one COM-based approach [31] is evaluated based on lab demos and a controlled experiment with master students (model completeness is $\sim$70%). Regarding the evaluation of the FOM-based approaches, only [61] presents evaluation results based on a controlled experiment (but the authors do not focus on the assessment of method effectiveness and efficiency). Regarding the evaluation of the POM-based approaches, three papers [21, 23, 47] report case-study based evaluation, while the results of controlled experiments are reported in [28, 34, 7, 19]. The most complete evaluation results of an MDSDM approach, which are based on the experiment conducted with a significant number of database practitioners, are presented in [19, 20] (average model completeness and precision are over 80%).

This article presents the recent achievements of an ongoing long-term research project about automatic BPM-driven CDM synthesis. The first ideas and prototype implementation (ADBdesign) were presented in 2010 [16]. The initial implementation was based on BPMs represented by UML ADs. The initial set of rules was upgraded, amended, and formalised in [14]. The automatic synthesis based on the finite set of UML ADs, was presented in [23]. The set of formal rules [14] was amended and applied [21] to collaborative BPMs represented by BPMN, and subsequently improved after a controlled experiment conducted with undergraduate students [7]. Through the experiment with students we obtained a very high completeness and precision of automatically generated CDMs (both average measures over 85%). After that experiment, we conducted the experiment [19, 20] with database practitioners, which almost confirmed the previous results. Based on the semantic capacity of POMs, which was identified and proved in the previous research and conducted experiments, we specified the aforementioned DSL named BMRL and proposed the two-phase BPM-driven approach to CDM synthesis [17]. This approach is depicted in Fig. 1 as notation-independent and outside of any POM region. In this article, we provide a detailed presentation of the approach, and the most recent implementation of the online CDM generator [18] based on the proposed approach.

In comparison to the existing approaches, the proposed approach is characterized by two-phase synthesis of the target model, while the existing approaches are characterized by direct synthesis. The implemented tool, in comparison to the existing tools, is the first online, web-based, publicly available tool. It is not dependent on any particular modeling platform, and enables automatic synthesis based on two different concrete source notations, in contrast to the existing tools, which are platform-dependent and enable (semi)automatic synthesis on the basis of a single starting notation. Furthermore, the implemented functionality is also available through the publicly available web service, which could be consumed from other modeling tools and platforms. Since the proposed two-phase approach is based on the previously experimentally evaluated approach to direct synthesis, this online tool is also characterized by very high effectiveness and efficiency, while the existing tools are not experimentally evaluated.

3. Two-phase BPM-driven CDM Synthesis

The MDSDM process is driven by a set of transformation rules combining two related sets of actions aimed at *extracting* characteristic concepts from the source model(s) and *generating* the corresponding concepts in the target model. In the existing MDSDM approaches, these two sets of actions are strongly coupled, meaning that the synthesis is performed by a single transformation program extracting concepts from the source model(s) and generating the target model. In the case of BPM-driven approaches, this means that a transformation program takes source BPMs represented by some concrete notation such as BPMN or UML AD, and generates the target data model represented by another concrete notation such as UML class diagram. Such *direct synthesis* has certain advantages such as directness and implementation facilitated by standardized transformation languages (e.g. ATL and QVT). However, it also requires different generators for different source notations. Therefore, these generators depend on modifications of the source and target notations (caused by metamodel changes and/or vendor specific implementations), and transformation rules as well. While the source notation-related modifications require modifications of only the corresponding generator, any modification of the transformation rules and/or target notation requires modifications of all generators, which can be considered a disadvantage of the direct synthesis.

In this article we argue that the above-described disadvantage can be reduced by decoupling the extracting and generating actions by separating them into two independent and consecutive activities, and splitting the synthesis into two phases. This can be achieved by introducing an intermediate layer between the source and target models. In the first phase, specific concepts are to be extracted from the source model(s) and represented at the intermediate layer. This can be achieved by transformation programs called *extractors*. A different extractor is required for each source notation. In the second phase, the target data model is to be generated based on the intermediate representation of the extracted concepts, which can be achieved by a single transformation program called *generator*. If we assume that the introduced intermediate layer is invariable[5], then the extractors depend only on the source notation-based modifications, while the generator depends on the modifications of the transformation rules and/or target notation. Like in the direct synthesis, the source notation-related modifications require modifications of only the corresponding extractor. However, any modification of the transformation rules and/or target notation requires modifications of only one generator. In this way the *indirect two-phase data model synthesis* can significantly reduce implementation efforts required to support diversity and/or modifications of the source notations and transformation rules for data model synthesis. Apart from the easier maintenance, the indirect synthesis could also be more effective in solving portability and interoperability issues, model checking, etc.

Alternatively to the proposed approach, some disadvantages of the direct synthesis could be reduced by applying a *hybrid approach*, which means that we may choose one (*primary*) source notation and implement the appropriate generator applying the principles of direct synthesis. The source models represented by other (*secondary*) modeling notations should be firstly transformed into the corresponding (equivalent)

[5] Strictly speaking, the intermediate layer is not immutable. Its changes occur with the additionally identified semantic capacity of BPMs for automatic CDM synthesis. However, these changes happen quite seldom compared with the related changes of source BPM notations.

models represented by the primary notation, and then the generator could be applied in order to obtain the target data model. For example, if we choose BPMN as the primary notation, then we need only the BPMN-based CDM generator, while the source models represented by some other notation (e.g. UML AD) should be firstly transformed into the corresponding BPMN models and subsequently used as the source for BPMN-based CDM synthesis. However, such a hybrid approach will also share the majority of the direct synthesis' disadvantages (*transformers* are dependent on both primary and secondary notations).

Following the idea of indirect data model synthesis, this article presents an approach to automated two-phase BPM-driven CDM synthesis. Although the intermediate layer could be differently represented, we use a DSL called *Business Model Representation Language* (BMRL) for its representation. It is a simple DSL for the representation of BPM concepts enabling the automatic CDM synthesis. Its specification is based on the results of our previous research [14, 21] indicating that BPMs are characterised by some typical *concepts* (such as *participants* and *objects*) and *facts* (such as *creation of objects* and *usage of objects*) that enable automatic CDM synthesis. Those concepts and facts are inherent to BPMs, but their representation may differ in different modelling languages. Independently of the used modelling notation, those concepts and facts have certain properties, meanings and roles (we use the *semantic capacity* term) that allow us to derive conclusions about the corresponding data model concepts (entity types, relationship types, etc.) and rules for mapping source BPMs to the target CDM.

With the introduction of BMRL, the CDM synthesis is split into two phases (Fig. 2). In the first phase, specific concepts are to be extracted from the source BPM and represented by BMRL. We illustrate the approach for two different notations (UML AD and BPMN) and provide details about the implemented extractors. In the second phase, the CDM (represented by the UML class diagram) is to be generated based on the BMRL-based representation of the extracted concepts. The generator has to implement a rather complex set of rules, which are dependent on simple and unique BMRL concepts, but independent of different source BPM notations.

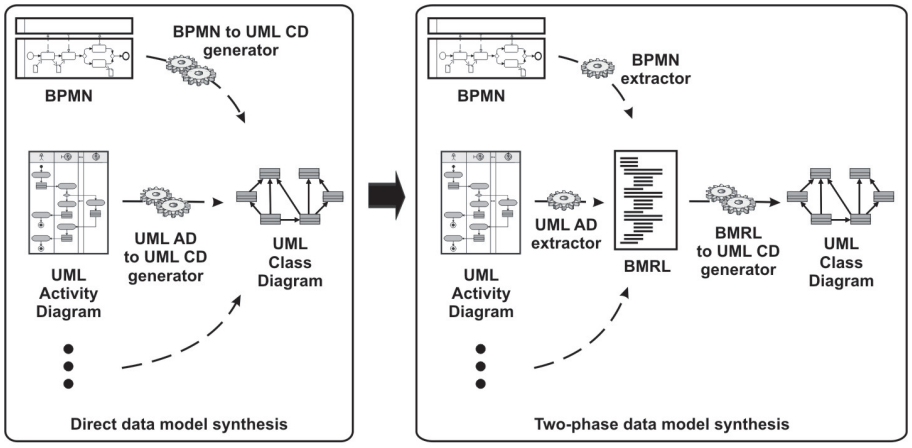


Fig. 2. Transition from direct (one-phase) to indirect (two-phase) data model synthesis

From the technical perspective (Fig. 3), the implemented extractors extract specific concepts from the source models conforming to the corresponding metamodels, and generate the corresponding BMRL representation of the extracted concepts according to the BMRL metamodel. The CDM generator generates the target model represented by UML class diagram conforming to the corresponding metamodel. As illustrated in Fig. 3, the proposed two-phase approach enables simple extensibility and support for other process-oriented notations (which are not necessarily metamodel-based) by implementing additional extractors.

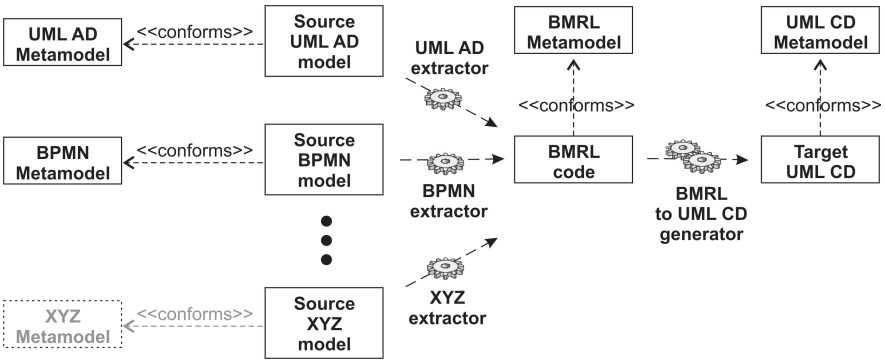


Fig. 3. Technical perspective of two-phase BPM-driven CDM synthesis

4. Semantic Capacity of BPMs for Automatic CDM Synthesis

As already stated, the previous research [14, 21, 7, 19, 20] implies that several common BPM concepts have semantic capacity for automatic CDM synthesis. This section provides a brief overview[6] and illustration (Fig. 4) of the identified semantic capacity of BPMs for automatic MDSDM. The identified semantic capacity constitutes the basis for the specification of BMRL and corresponding rules for both phases.

Typical BPM concepts that enable automatic generation of *entity types* (*classes*) in the target CDM are: *participants*, *roles*, *objects*, *message flows*, and *activations of existing objects*. Participants (may) have different roles. Participants and their roles are represented differently in BPMs (pools/lanes, partitions/subpartitions). All types of participants, and all their roles as well, are to be mapped into the corresponding classes in the target CDM (rule T_1). During the execution of a business process, participants perform tasks (actions) and exchange messages. Each different type of objects, and message flows as well, is to be mapped into the corresponding class in the target CDM (T_2). Each task/action may have a number of input and output objects that can be in different states. The objects can be *generated* in the given process, or *existing* – created in some other process. An activation represents the fact that an existing object constitutes input in a task that changes its state. Activated objects have the semantics similar to that of generated objects and need to be represented with a corresponding class (activation class) (T_3).

[6] A complete formal specification of transformation rules for direct BPM-driven CDM synthesis is given in [14, 21, 20].

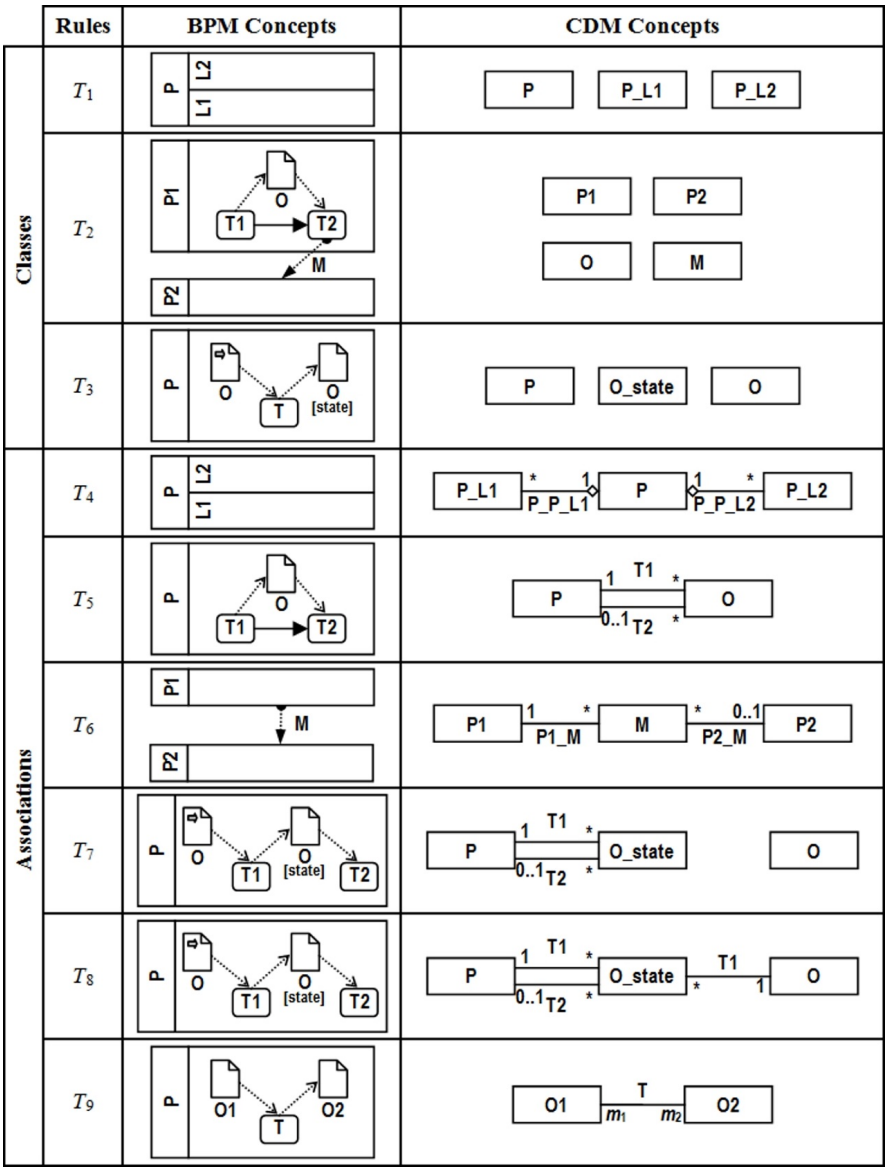


Fig. 4. Mapping of BPM concepts into CDM concepts

There are several common patterns in BPMs enabling automatic generation of *relationship types* (*associations*) in the target CDM. They enable generation of three types of associations: *participant-participant*, *participant-object*, and *object-object*. *Participant-participant* associations originate from the fact that a participant may have different roles. This implies that the class representing a pool should have associations with classes representing corresponding lanes (T_4). Process patterns having semantic potential for the generation of *participant-object* associations are: creation and subsequent usage of

generated objects (T_5), exchange of messages (T_6), and activation and subsequent usage of activated objects (T_7). Every mentioned fact is to be represented by corresponding association(s) with multiplicities 1:* or 0..1:*. There are two bases for the generation of *object-object* associations: (i) activation (T_8), which is represented with an association between the class that represents the existing object and the class that represents its activation, and (ii) tasks having input and output objects of different types (T_9), where the association end multiplicities depend on the nature of the objects (if they are either generated, non-activated existing or activated existing objects).

5. Phase I: DSL-based Representation of BPM Concepts

The first phase of the CDM synthesis includes the extraction of concepts from the source BPM and their BMRL-based representation. This section presents the BMRL specification and implementation of BPM extractors.

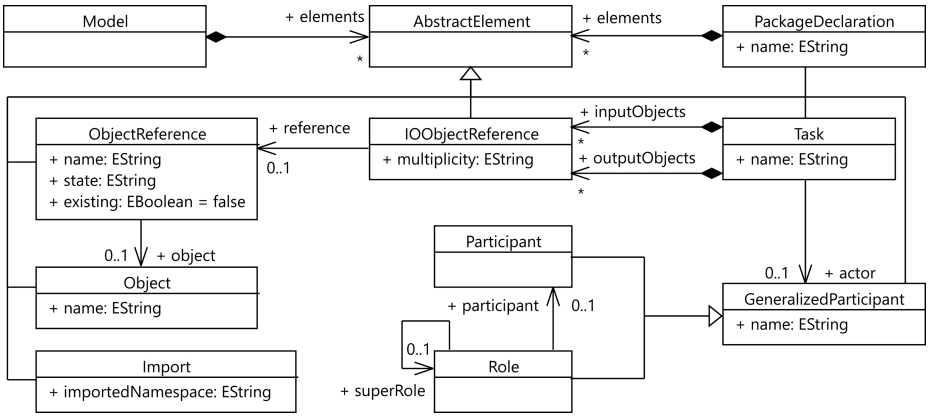
5.1. BMRL Specification

DSL is a computer programming language of limited expressiveness focusing on a particular domain. Programming languages, including DSLs, consist of three main elements: concrete syntax, abstract syntax and semantics [63]. The concrete syntax defines the notation with which users can express programs (it may be textual, graphical, tabular or combined). The abstract syntax is a data structure that can hold the semantically relevant information expressed by a program (most often represented by a tree or graph). There are two kinds of semantics. The static semantics is defined by a set of constraints and/or type system rules to which programs have to conform, while execution semantics refers to the meaning of a program once it is executed [63].

Based on the identified semantic capacity of BPMs for MDSDM, we defined a DSL named *Business Model Representation Language* (BMRL). For its specification we used Xtext[7] framework. Xtext belongs to parser-based approaches, in which a grammar specifies the sequence of tokens that forms structurally valid programs. In such systems, users interact only with concrete syntax, while the *abstract syntax tree* (AST) is constructed from the concrete syntax of a program [63]. Xtext relies on *Eclipse Modeling Framework* (EMF) [24] models for internal AST representation.

The Ecore metamodel of BMRL and its grammar are shown in Fig. 5. A BMRL program contains an arbitrary number of abstract elements: `PackageDeclaration`. `Import`, `GeneralizedParticipant`, `Task`, `Object`, `ObjectReference`, and `IOObjectReference`. Each business process participant can be represented by the `Participant` element. Participants can have roles (`Role`), and each role can have sub-roles. The `Participant` and `Role` elements have the `name` attribute. Participants (or participants with specified roles) perform tasks (`Task`). The `Task` element has the `name` attribute. Each task can have inputs and outputs, which are represented by input/output specification elements (`IOObjectReference`). Each different type of objects is represented by the `Object` element. For each type of objects, one or more references (`ObjectReference`) can be specified, since objects can be in different states (`state`). The `existing` attribute shows whether the given reference represents a reference to an

[7] http://www.eclipse.org/Xtext/




```
grammar org.unibl.etf.BMRL with org.eclipse.xtext.common.Terminals
generate bMRL "http://www.etf.unibl.org/bmrl2cd/BMRL"
Model:
    (elements+=AbstractElement)*;
PackageDeclaration:
    'package' name=QualifiedName '{' (elements+=AbstractElement)* '}';
AbstractElement:
    PackageDeclaration | Import | GeneralizedParticipant | Object |
    ObjectReference | Task;
QualifiedName:
    ID ('.' ID)*;
Import:
    'import' importedNamespace=QualifiedNameWithWildcard;
QualifiedNameWithWildcard:
    QualifiedName '.*'?;
GeneralizedParticipant:
    Participant | Role;
Participant:
    'participant' name=ID;
Role:
    'role' name=ID ('(' superRole=[Role|QualifiedName] ')' | 'of'
           participant=[Participant|QualifiedName]);
Object:
    'object' name=ID;
ObjectReference:
    'objectReference' name=ID 'references' object=[Object|QualifiedName]
                      ('[' state=ID ']')? (existing?='existing')?;
IOObjectReference:
    reference=[ObjectReference|QualifiedName]
              'multiplicity' multiplicity=Multiplicity;
Task:
    'task' name=ID '{'
    'actor' ':' actor=[GeneralizedParticipant|QualifiedName]
    ('input' '(inputObjects+=IOObjectReference)* '}')?
    ('output' '{' (outputObjects+=IOObjectReference)* '}')? '}';
Multiplicity:
    INT | '-1';
```

Fig. 5. BMRL metamodel (top) and grammar (bottom)

existing object, or to an object generated in the given BPM. The `IOObjectReference` references one of the `ObjectReference` elements and specifies its multiplicity. BMRL supports the use of packages (`PackageDeclaration`) and imports (`Import`) in order to avoid name ambiguities.

5.2. BPM Extractors

As previously described, the first phase includes the extraction of important concepts from the source BPM and their representation according to the implemented DSL. This extraction and generation of the corresponding BMRL code can be implemented in different ways, either by using general purpose or specialized transformation languages. We used Acceleo[8] for the implementation of extractors. So far, we have implemented two extractors – one for BPMN and another one for UML AD. In this article we provide implementation details only for the BPMN extractor[9].

The BPMN extractor performs the extraction of characteristic concepts from the source BPM represented by BPMN (the related BPMN metamodel [50] excerpt is shown in Fig. 6), and generates the corresponding BMRL representation of the extracted concepts.

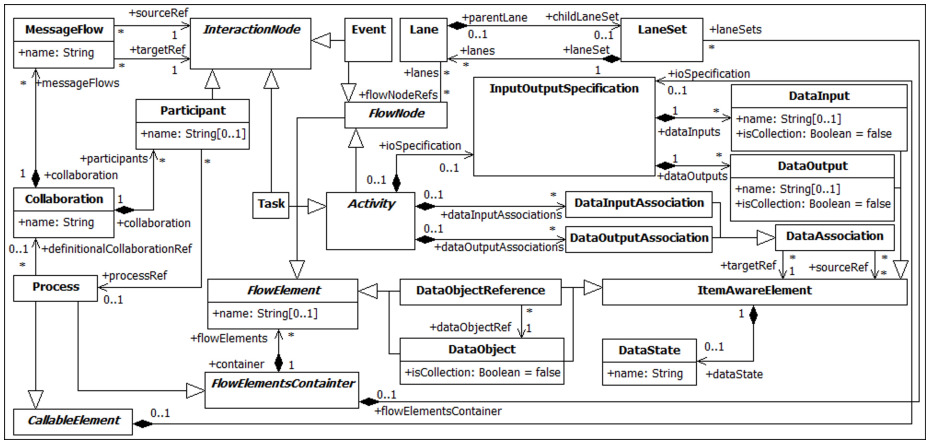


Fig. 6. BPMN metamodel [50]

The formal rules for the extraction of concepts from BPMN and their BMRL-based representation are given in Tables 1 and 2. This set of rules enables the extraction of: participants and their roles, generated and existing objects, as well as tasks having input and/or output objects.

For each `Participant` in the source model, the corresponding `Participant` element (of the same name) has to be generated in the BMRL. For each `Lane` the corresponding `Role` element has to be generated. The generated `Role` belongs to the `Participant` that corresponds to the `Participant` containing the given `Lane`.

Each `Task` may have input and/or output objects (`DataInput`, `DataOutput`, `DataObjectReference`, and `MessageFlow`), which can be in different states. For each `DataInput`, `DataOutput`, `MessageFlow`, and `DataObject` referenced by `DataObjectReference`, the corresponding `Object` and `ObjectReference` elements have to be generated. The `Object` element represents different type of objects

[8] http://www.eclipse.org/acceleo/

[9] Implementation details for the UML AD extractor are presented in [17].

and it has to be named the same as the given source element. The `ObjectReference` element represents the concrete object and it should be named by concatenating the names of the object and its state. Attributes `state` and `existing` of the `ObjectReference` depend on the state and nature (generated/existing) of the given source element. If it represents the existing object, then its name should be prefixed with `'Existing'`.

For each `Task` element in the source model, the corresponding `Task` has to be generated, whose actor attribute corresponds to the `GeneralizedParticipant` performing the given task, while the `inputObjects` and `outputObjects` attributes are represented by the `IOObjectReference` elements generated for `dataInput-Associations` and `dataOutputAssociations` elements of the task, as well as for `MessageFlows` referencing the given task. The `multiplicity` attribute has to be set based on the `isCollection` attribute of the element referenced by the (input or output) data association or message flow.

For each `MessageFlow` element referencing `Participant` or `Event` in the source model, the corresponding `Task` (named `'SendMessage'` or `'ReceiveMessage'`) has to be generated. Actor of the generated task is `GeneralizedParticipant` which corresponds to the `Participant` referenced by the message flow, or the `Participant` related with the `Event` referenced by the message flow.

Figure 7 (on the left) provides an illustration of the extraction of specific concepts from a simple BPM (BPMN) and generation of the corresponding BMRL code. This simple BPM represents an activation of the existing object `Book`, which is performed by the `Librarian` participant. The dashed arrows depict the mapping of source BPMN concepts into the target BMRL concepts.

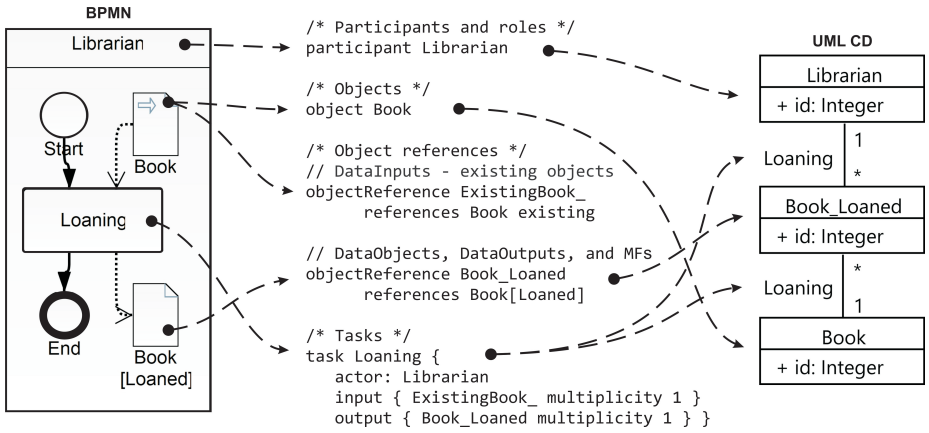


Fig. 7. From BPM (BPMN) through BMRL to CDM (UML CD)

Table 1. Rules for extraction of concepts from BPMN and their BMRL representation

Source concept in BPMN	**Target BMRL concept**
`sp:Participant`	`p:Participant` `{p.name=sp.name}`

`l:Lane`
`{l∈sp.processRef.laneSets.lanes}`
`ls:Lane`
`{ls∈l.childLaneSet.lanes}`
	`r:Role`
	`{r.name=l.name ∧ r.participant=p}`
	`rs:Role`
	`{rs.name=ls.name ∧ rs.superRole=r}`

`di:DataInput`
`{di∈sp.processRef.ioSpecification.dataInputs}`
	`o:Object`
	`{o.name=di.name}`
	`or:ObjectReference`
	`{or.object=o ∧ or.existing=true ∧`
	`or.name=concat('Existing',di.name,di.dataState.name) ∧`
	`or.state=di.dataState.name}`

`do:DataOutput`
`{do∈sp.processRef.ioSpecification.dataOutputs}`
	`o:Object`
	`{o.name=do.name}`
	`or:ObjectReference`
	`{or.object=o ∧ or.existing=false ∧`
	`or.name=concat(do.name,do.dataState.name) ∧`
	`or.state=do.dataState.name}`

`mf:MessageFlow`	`o:Object`
	$\{\,\mathbf{o.name} = \begin{cases} \mathbf{mf.name}, \mathbf{isUndefined(mf.messageRef)} \\ \mathbf{mf.messageRef.name},\ \text{otherwise} \end{cases}\,\}$
	`or:ObjectReference`
	`{or.object=o ∧ or.existing=false ∧`
	$\mathbf{or.name} = \begin{cases} \mathbf{mf.name}, \mathbf{isUndefined(mf.messageRef)} \\ \mathbf{mf.messageRef.name},\ \text{otherwise} \end{cases}\,\}$

`dor:DataObjectReference`	`o:Object`
	`{o.name=dor.dataObjectRef.name}`
	`or:ObjectReference`
	`{or.object=o ∧ or.existing=false ∧`
	`or.name=concat(dor.dataObjectRef.name,dor.dataState.name) ∧`
	`or.state=dor.dataState.name}`

Table 2. Rules for extraction of concepts from BPMN and their BMRL representation (continued)

Source concept in BPMN	Target BMRL concept
st:Task, in:BaseElement, out:BaseElement {in ∈ st.dataInputAssociations.sourceRef ∪ {mf:MessageFlow | mf.targetRef=t} ∧ out ∈ st.dataOutputAssociations.targetRef ∪ {mf:MessageFlow | mf.sourceRef=t} }	t:Task {t.name=st.name ∧ t.actor=(p ∨ r ∨ rs) ∧ iior∈t.inputObjects ∧ iior.reference=or ∧ iior.multiplicity=im ∧ oior∈t.outputObjects ∧ oior.reference=or ∧ oior.multiplicity=om, $\texttt{im} = \begin{cases} *, \texttt{C1} \\ 1, \text{ otherwise} \end{cases}$, $\texttt{om} = \begin{cases} *, \texttt{C2} \\ 1, \text{ otherwise} \end{cases}$, C1=in.isCollection ∨ ¬isUndefined(in.dataObjectRef) ∧ in.dataObjectRef.isCollection ∨ ¬isUndefined(in.messageRef) ∧ ¬isUndefined(in.messageRef.itemRef) ∧ in.messageRef.itemRef.isCollection, C2=out.isCollection ∨ ¬isUndefined(out.dataObjectRef) ∧ out.dataObjectRef.isCollection ∨ ¬isUndefined(out.messageRef) ∧ ¬isUndefined(out.messageRef.itemRef) ∧ out.messageRef.itemRef.isCollection}
mf:MessageFlow {typeof(mf.sourceRef) ∈ {Partcipant,Event}}	t:Task {t.name=concat('SendMessage',S1) ∧ t.actor=(p ∨ r ∨ rs) ∧ oior∈t.outputObjects ∧ oior.reference=or ∧ oior.multiplicity=om, $\texttt{S1} = \begin{cases} \textbf{mf.name}, \textbf{isUndefined(mf.messageRef)} \\ \textbf{mf.messageRef.name}, \text{ otherwise} \end{cases}$, $\texttt{om} = \begin{cases} *, \texttt{C3} \\ 1, \text{ otherwise} \end{cases}$, C3=¬isUndefined(mf.messageRef) ∧ ¬isUndefined(mf.messageRef.itemRef) ∧ mf.messageRef.itemRef.isCollection }
mf:MessageFlow {typeof(mf.targetRef) ∈ {Partcipant,Event}}	t:Task {t.name=concat('ReceiveMessage',S2) ∧ t.actor=(p ∨ r ∨ rs) ∧ iior∈t.inputObjects ∧ iior.reference=or ∧ iior.multiplicity=im, $\texttt{S2} = \begin{cases} \textbf{mf.name}, \textbf{isUndefined(mf.messageRef)} \\ \textbf{mf.messageRef.name}, \text{ otherwise} \end{cases}$, $\texttt{im} = \begin{cases} *, \texttt{C4} \\ 1, \text{ otherwise} \end{cases}$, C4=¬isUndefined(mf.messageRef) ∧ ¬isUndefined(mf.messageRef.itemRef) ∧ mf.messageRef.itemRef.isCollection }

6. Phase II: DSL-based CDM Synthesis

The second phase includes automatic generation of the target CDM represented by UML class diagram (related UML metamodel [51] excerpt is shown in Fig. 8) based on the BMRL representation of the source BPM.

The rules for the automatic generation of the UML class diagram, based on the BMRL representation of the source BPM, are given in Tables 3 and 4. The first two columns contain source and target concepts, while the third column contains labels for the corresponding mapping rules illustrated in Fig. 4. An informal description of these rules is given in Sect. 4.

The formal rules constitute the basis for the implementation of the CDM generator. The generator can be implemented in different ways. In our case, it is implemented as an automatic Xtend-based[10] generator.

Figure 7 above (right side) provides an illustration of the automatic generation of the UML class diagram. The dashed arrows are used to illustrate mappings of BMRL concepts into the UML concepts.

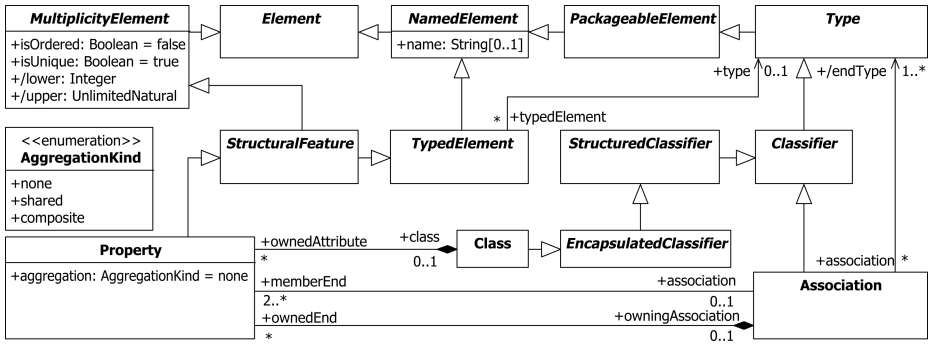


Fig. 8. UML CD metamodel [51]

Table 3. Mapping of BMRL concepts into UML class diagram concepts

Source BMRL concepts	Target UML CD concepts	Rules
`p:Participant`	`ep:Class` `{ep.name=p.name}`	T_1
`r:Role` `{r.participant=p}`	`er:Class` `{er.name=concat(r.participant.name,r.name)}`	T_1

	`rpr:Association`
	`{rpr.name=concat(r.participant.name,er.name) ∧`
	`rpr.memberEnd.source=ep ∧`
	`multiplicity(rpr.memberEnd.source)=1 ∧`
	`rpr.memberEnd.target=er ∧`
	`multiplicity(rpr.memberEnd.target)=*}`
		T_4

[10] http://www.eclipse.org/xtend/

Table 4. Mapping of BMRL concepts into UML class diagram concepts (continued)

Source BMRL concepts	Target UML CD concepts	Rules
o:Object	eo:Class {eo.name=o.name}	T_2
t:Task, in:IOObjectReference[0..*], out:IOObjectReference[0..*] {in=t.inputObjects ∧ out=t.outputObjects ∧ in.reference.existing=true ∧ in.reference.object= out.reference.object}	ea:Class {ea.name=concat(in.reference.object.name, out.reference.state)}	T_3
	rpa:Association {rpa.name=t.name) ∧ rpa.memberEnd.source=(ep ∨ er) ∧ multiplicity(rpa.memberEnd.source)=1 ∧ rpa.memberEnd.target=ea ∧ multiplicity(rpa.memberEnd.target)=∗}	T_7
	roa:Association {roa.name=t.name) ∧ roa.memberEnd.source=eo ∧ multiplicity(roa.memberEnd.source)=1 ∧ roa.memberEnd.target=ea ∧ multiplicity(roa.memberEnd.target)=∗}	T_8
t:Task, in:IOObjectReference[0..*], out:IOObjectReference[0..*] {in=t.inputObjects ∧ out=t.outputObjects ∧ ∄in ∣ in.reference.object= out.reference.object}	rgc:Association {rgc.name=t.name) ∧ rgc.memberEnd.source=(ep ∨ er) ∧ multiplicity(rgc.memberEnd.source)=1 ∧ rgc.memberEnd.target=eo ∧ multiplicity(rgc.memberEnd.target)=∗}	T_5 T_6
t:Task, in:IOObjectReference[0..*] {in=t.inputObjects ∧ in.reference.existing=false}	ru:Association {ru.name=t.name) ∧ ru.memberEnd.source=(ep ∨ er) ∧ multiplicity(ru.memberEnd.source)=0..1 ∧ ru.memberEnd.target=(eo ∨ ea) ∧ multiplicity(ru.memberEnd.target)=∗}	T_5 T_6 T_7
t:Task, in:IOObjectReference[0..*], out:IOObjectReference[0..*] {in=t.inputObjects ∧ out=t.outputObjects ∧ in.reference.object≠ out.reference.object}	roo:Association[n] {roo.name=t.name) ∧ roo.memberEnd.source=(eo ∨ ea) ∧ multiplicity(roo.memberEnd.source)=sm ∧ roo.memberEnd.target=(eo ∨ ea) ∧ multiplicity(roo.memberEnd.target)=tm} $\mathtt{n} = \begin{cases} 1, \ \mathtt{in.multiplicity} \in \{1, *\} \\ \mathtt{in.multiplicity}, \ \text{otherwise} \end{cases}$ $\mathtt{low(sm)} = \begin{cases} 0, \mathtt{in.multiplicity} = * \vee \\ \quad \exists \mathtt{r} \in \mathtt{in} \mid \mathtt{r.reference.object} = \\ \qquad \mathtt{out.reference.object} \wedge \\ \qquad \mathtt{r.reference.existing} = \mathtt{false} \\ 1, \ \text{otherwise} \end{cases}$ $\mathtt{high(sm)} = \begin{cases} *, \ \mathtt{in.multiplicity} = * \\ 1, \text{otherwise} \end{cases}$ $\mathtt{low(tm)} = 0$ $\mathtt{high(tm)} = \begin{cases} *, \ \mathtt{out.multiplicity} \neq 1 \vee \\ \quad \mathtt{in.reference.existing} = \mathtt{true} \\ 1, \text{otherwise} \end{cases}$	T_9

7. Online Two-phase BPM-driven CDM Generator

The proposed approach enables implementation of a modular tool for BPM-driven CDM synthesis, which consists of loosely coupled components aimed at automatic extraction of specific concepts from the source BPMs represented by different concrete notations, their BMRL-based representation, and automatic CDM generation.

Like all tools in the existing tool-supported MDSDM approaches, our initial set of tools, presented in [17], was also platform-dependent, since all tools were implemented as Eclipse plug-ins. In order to obtain a platform-independent and publicly available tool for the BPM-driven CDM synthesis, we performed the migration of these tools into a SOA[11] application.

7.1. Architecture of Online Generator

The online generator is implemented as an orchestration of web services. Its architecture is presented in Fig 9. We used the REST architectural style for implementation of services. In a positive scenario, the orchestrator service receives a source BPM represented by BPMN or UML AD (`input.bpmn`/`input.uml`), and returns the corresponding CDM (`cdm.uml`) to the caller.

In the first phase, the orchestrator service sends the source BPM to the corresponding extractor service, which takes the XMI representation of the source model, generates the corresponding BMRL code (`input.bmrl`) and returns it to the orchestrator service. Currently, two extractor services (shown as `BPMN extractor` and `UML AD extractor` in Fig 9) are implemented. Implementation is based on the Java archives (JAR) obtained by exporting the Acceleo-based extractors [17]. The proposed architecture enables easy extension of the online generator by additional extractors aimed at extraction of characteristic concepts from BPMs represented by other notations.

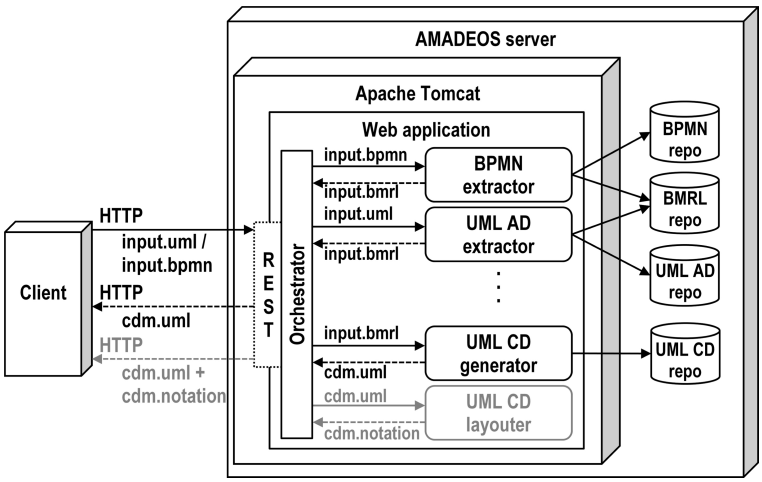


Fig. 9. Architecture of the online CDM generator

[11] Service-oriented architecture

In the second phase, the orchestrator service sends the generated BMRL representation (`input.bmrl`) to the generator service (`UML CD generator`), which takes the BMRL code, generates the XMI representation of the target CDM (`cdm.uml`) and returns it to the orchestrator service. Implementation of the generator service is based on the Java archive obtained by exporting the Xtend-based generator [17].

Each extractor stores the source model and generated BMRL code in appropriate server repositories. The generator service stores the generated CDM in the corresponding repository, as well. These repositories will be used in the future for further analysis of the approach and implemented system.

Currently, we are developing the online generator further. The next release will include the layouter service (`UML CD layouter` in Fig 9) aimed at automatic generation of the layout of the UML class diagram (`cdm.notation`), in order to enable visualization of the automatically generated CDM in the browser.

7.2. Usage of Online Generator

The usage of the implemented online generator can be twofold. In the first scenario, developers are able to implement their own applications invoking the exposed web service that orchestrates the two-phase CDM synthesis. In the second scenario, database designers are able to use the implemented client application.

For the first usage scenario, the online CDM generator[12] exposes one method for the target CDM generation, which accepts the `multipart/form-data media` type. The request should consist of two required named body parts `"source_model_type"` and `"input"`. The `"source_model_type"` body part defines the type of the input model. The permitted values are `"AD"` for UML AD and `"BPMN"` for BPMN. The `"input"` body part is the uploaded source model file. In the case of a successful generation of the target model, the service responds with status 200 (`OK`) and produces the `application/octet-stream` media type representing the generated target model. In the case of any error, the service responds with status 204 (`no content`)[13]. An example of the service client[14] is given in Fig 10.

The second usage scenario of the online generator is a client application[15] (Fig 11). Through this application database designers are able to upload the source BPM and download the XMI-representation of the automatically generated CDM, which can subsequently be imported and visualized in a certain modeling tool/platform. The visualization and editing functionalities of automatically generated models in the web browser are not currently supported by the implemented client application. The relevant work is underway.

---

[12] The implemented online generator is available at:
`http://m-lab.etf.unibl.org:8080/amadeos/services/generate/cdm`

[13] We would like to emphasize the fact that the currently supported BPM specifications are BPMN 2.0 [50] and UML 2.5 [51]. However, it is possible that the generator will return status 204 in cases of some vendor's specificities. Currently, we are developing robust extractor services in order to overcome problems related to platform and vendor serialization specificities.

[14] In order to facilitate development of client applications consuming the implemented online generator, as well its usage, some sample source models, Eclipse-based modeling platform and sample client code are available at GitLab: `https://gitlab.com/m-lab-research/amadeos`

[15] The client application is available at:
`http://m-lab.etf.unibl.org:8080/amadeos/generator.html`

```
FileDataBodyPart filePart =
       new FileDataBodyPart("input", new File("path_to_source_model"));
FormDataMultiPart multipart = new FormDataMultiPart();
multipart.field("source_model_type", "AD").bodyPart(filePart);
// For BPMN: multipart.field("source_model_type", "BPMN").bodyPart(filePart);

ClientConfig clientConfig = new ClientConfig().register(MultiPartFeature.class);
Client client = ClientBuilder.newClient(clientConfig);
String server = "http://m-lab.etf.unibl.org:8080/amadeos/services/";
WebTarget target = client.target(server).path("generate").path("cdm");

Response response =
    target.request().post(Entity.entity(multipart, multipart.getMediaType()));

if (response.getStatus() == 200) {
    InputStream is = response.readEntity(InputStream.class);
    File f = new File("path_to_target_model.uml");
    FileUtils.copyToFile(is, f);
    is.close();
}

filePart.cleanup();
multipart.close();
client.close();
response.close();
```

Fig. 10. An example of the service client

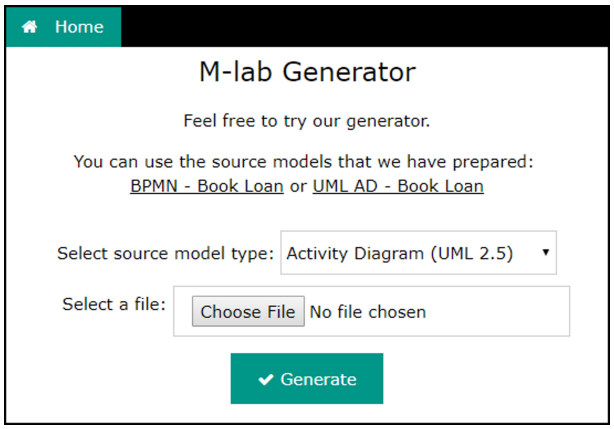
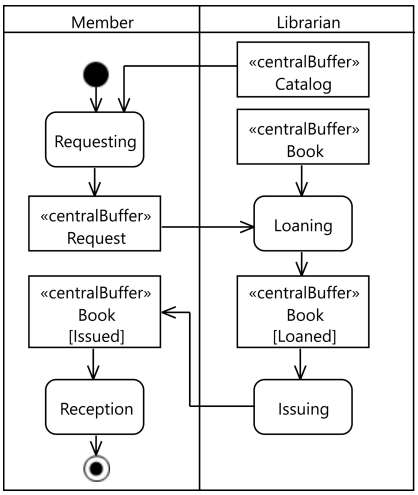


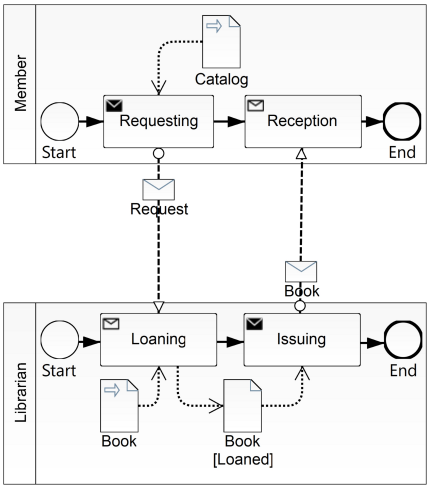
Fig. 11. Screenshot of the client application form

8. Illustrative Example of Two-phase BPM-driven CDM Synthesis

This section presents an illustrative example of the proposed approach. This example aims to illustrate the process of two-phase synthesis as such, and to prove that BPMs, regardless of the used notation, have the semantic capacity for automatic CDM synthesis. We prepared two simple BPMs represented by two concrete notations (UML AD and BPMN), which are currently supported by the online tool. Both BPMs represent the same business process (Book loaning)[16]. These two models are shown on the top of Fig. 12.

[16] A detailed description of these sample models is omitted due to their simplicity.





UML AD extractor

BPMN extractor

```
participant Librarian
participant Member

object Book
object Request
object Catalog

objectReference ExistingCatalog_
      references Catalog existing
objectReference ExistingBook_
      references Book existing
objectReference Request_
      references Request
objectReference Book_Issued
      references Book[Issued]
objectReference Book_Loaned
      references Book[Loaned]

task Loaning {
 actor: Librarian
 input { Request_ multiplicity 1
         ExistingBook_ multiplicity 1 }
 output { Book_Loaned multiplicity 1 } }
task Reception {
 actor: Member
 input { Book_Issued multiplicity 1 }
 output { } }
task Issuing {
 actor: Librarian
 input { Book_Loaned multiplicity 1 }
 output { Book_Issued multiplicity 1 } }
task Requesting {
 actor: Member
 input { ExistingCatalog_ multiplicity 1 }
 output { Request_ multiplicity 1 } }
```

```
participant Librarian
participant Member

object Book
object Request
object Catalog

objectReference ExistingBook_
      references Book existing
objectReference ExistingCatalog_
      references Catalog existing
objectReference Book_
      references Book
objectReference Request_
      references Request
objectReference Book_Loaned
      references Book[Loaned]

task Reception {
 actor: Member
 input { Book_ multiplicity 1 }
 output { } }
task Issuing {
 actor: Librarian
 input { Book_Loaned multiplicity 1 }
 output { Book_ multiplicity 1 } }
task Requesting {
 actor: Member
 input { ExistingCatalog_ multiplicity 1 }
 output { Request_ multiplicity 1 } }
task Loaning {
 actor: Librarian
 input { ExistingBook_ multiplicity 1
         Request_ multiplicity 1 }
 output { Book_Loaned multiplicity 1 } }
```

Fig. 12. BMRL representation of sample BPMs: UML AD (left) and BPMN (right)

Apart from the sample BPMs, Fig. 12 also shows their BMRL representation (bottom) automatically generated by the implemented extractors. In this way, Fig. 12 illustrates the first phase of the proposed approach.

Figure 13 depicts the result of the second phase of BPM-driven CDM synthesis. The depicted class diagram is visualized by the Papyrus[17] tool in the Eclipse IDE. It represents the automatically generated CDM based on the BMRL representation of the sample source BPM(s) shown in Fig. 12.

The fact that the same CDM represents the result of the application of the implemented tools to both sample BPMs (although represented by two different notations), proves the hypothesis that BPMs, regardless of the modeling notation, are characterized by some common concepts and facts having semantic capacity for automatic CDM synthesis.

The sample BPMs constitute the simplified models of the book loaning process. Consequently, the automatically generated CDM also constitutes a simplified version of the corresponding target CDM. Given the model simplicity, we do not provide a detailed analysis and evaluation of the automatically generated CDM, particularly its completeness. However, regardless of its simplicity, the correctness of the automatically generated CDM is very high.

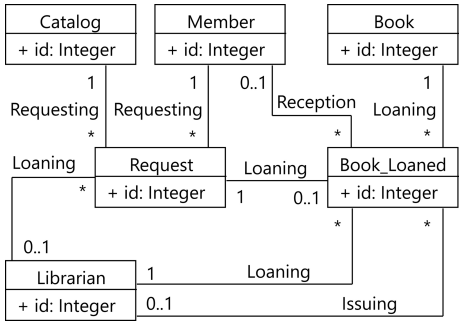


Fig. 13. UML class diagram representing the automatically generated CDM based on the source BPM(s) shown in Fig. 12

9. Verification and Validation

In this section we evaluate the proposed approach to automated two-phase BPM-driven CDM synthesis based on the experimental evaluation of the implemented online CDM generator.

9.1. Verification

Verification is the process of checking that the software meets the specification. We verified the implemented online two-phase generator against the existing direct ATL-based generator [20].

[17] https://eclipse.org/papyrus/

In order to verify implementation of the online generator, we applied it on real BPMs. Here we provide a real BPMN model (Fig. 14) of order processing, which was also used in the experiment conducted with database professionals [19, 20]. Although the presented workflow (Fig. 14) is quite intuitive, we still provide a short description. The given model represents an online purchasing and selling business process, with deferred payment option assumed. The process starts with the customer online order specification, which consists of a header and order details. After the order has been created, the salesman checks the customer's status (validity, creditworthiness, etc.) and availability of ordered items in stock. Based on the performed checks, the salesman decides whether the order is acceptable or not (in the latter case the order is canceled). If the order is acceptable, the invoice (consisting of a header and invoice details) is created and the stockman starts collecting and packing for shipment and delivery stock items for all confirmed order details. After all items have been prepared for delivery, the driver picks up the documentation and loads and delivers them to the customer. After delivery the customer confirms receipt of goods and the process finishes with setting the related document status to delivered.

Figure 15 depicts a class diagram (visualised by Papyrus) representing the automatically generated CDM based on the BMRL representation of the source BPMN model of order processing. This CDM, automatically generated by the implemented online two-phase generator, is equal to the CDM automatically generated by the direct ATL-based generator [20]. By applying the same verification procedure for other BPMs, we also obtain the complete matching of the corresponding generated CDMs. This fact, that we obtain equal CDMs by applying both two-phase generator and direct ATL-based generator, confirms that the online two-phase generator properly implements the same functionality of the automatic BPM-driven CDM synthesis.

9.2. Validation

Validation is the process of checking whether the specification captures the customer's needs. In the context of evaluation of the proposed approach and implemented online generator, validation could be twofold – from perspectives of two different classes of users: database designers and developers.

Validation from the database designers' perspective includes an assessment of the effectiveness and efficiency of the implemented online generator. Since both online CDM generator and the direct ATL-based generator [20] generate the same two CDMs based on the same BPM, the effectiveness of the two-phase BPM-driven synthesis is equal to the effectiveness of the direct BPM-driven synthesis. The efficiency is similar due to the equal complexity[18] of the approaches.

Here we refer to the main results of the experiment conducted with database practitioners [19, 20] in order to evaluate the direct BPM-driven CDM synthesis and direct ATL-based generator [20]. The evaluation was twofold. Firstly, it focused on the assessment of approach effectiveness, through the assessment of correctness (*precision*) and completeness (*recall*) of the automatically generated model. The average effectiveness (*F-measure*) was $\sim$78% for automatic generation of classes, and $\sim$85% for

[18] Both approaches have linear complexity ($O(n)$).

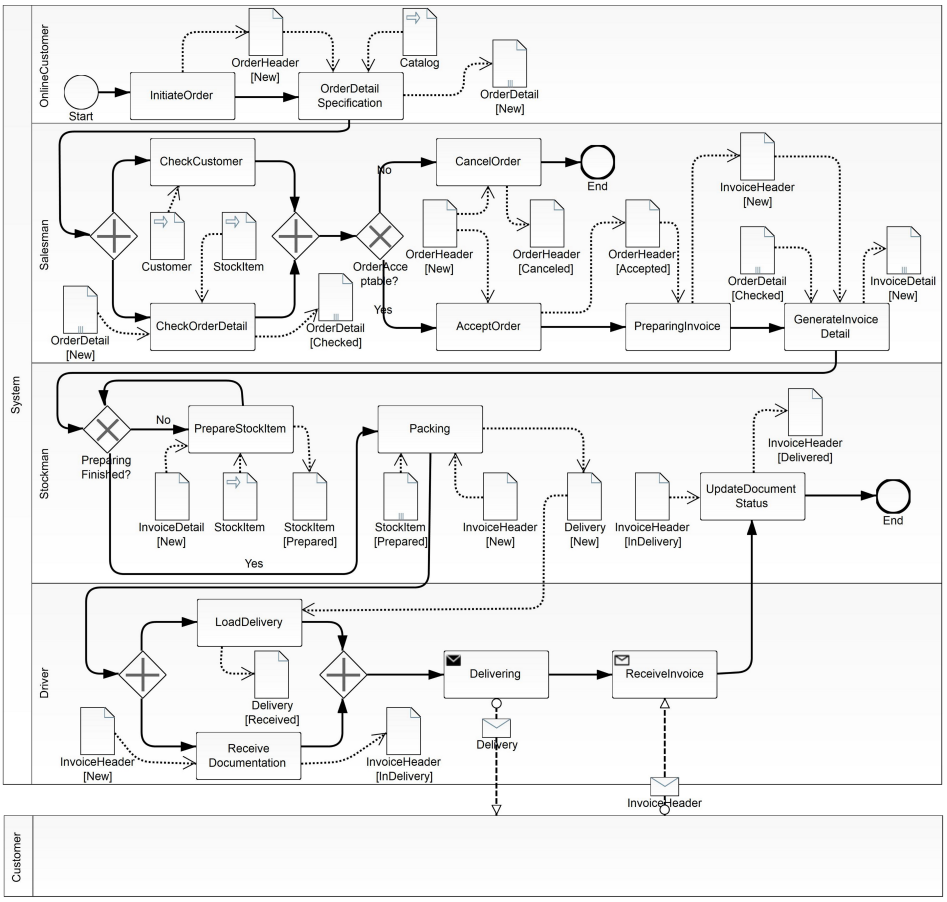


Fig. 14. BPMN model of order processing [19, 20]

associations. The average recall of the generated model was above 80%. The average precision for automatically generated classes was above 75%, while the average precision for associations was about 90%. Secondly, it focused on the assessment of usability of the automatically generated model as a starting base for manual design of the target model, as well as the assessment of efficiency of such an approach in contrast to the manual design from scratch. The experiment confirmed that the automatically generated model can also be efficiently used as a starting point for manual design of the target model, since it significantly shortens the time required for design. The calculated speed-up factors confirmed that the manual design, which takes the automatically generated model as a starting base, almost bisects the estimated efforts and actual time spent to obtain the target model in contrast to the manual design from scratch.

Some potential threats to validity of the experiment [19, 20] and derived conclusions are related to the source model quality. Someone may find that the used source BPM differs from the typical real BPMs, since it represents the result of a disciplined approach which forces modeling of resources. The approach is certainly dependent on the source

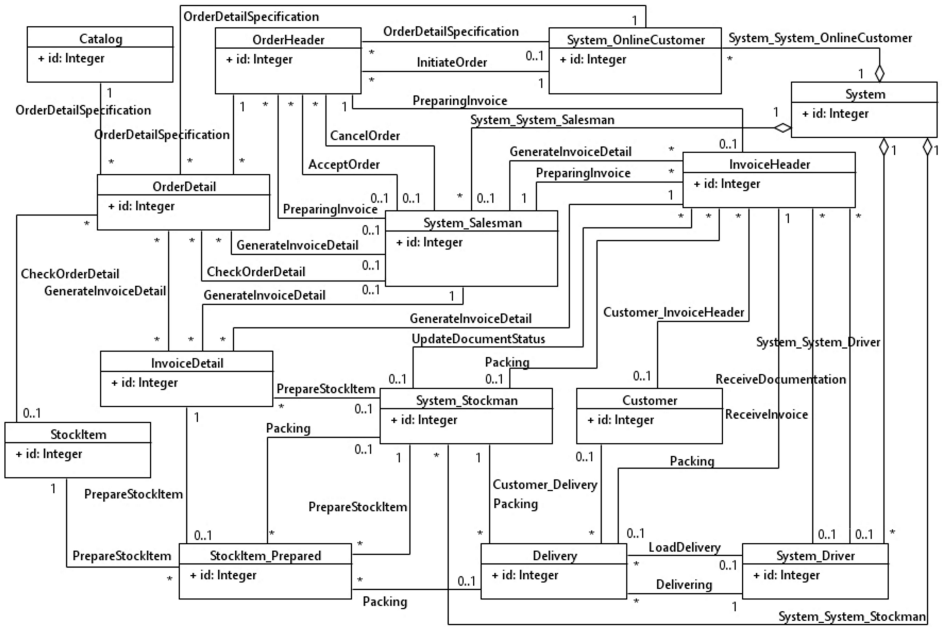


Fig. 15. UML class diagram representing the automatically generated CDM based on the BPMN model shown in Fig. 14

model quality, particularly on the representation of objects and object flows, since the completeness and correctness of the automatically generated CDM directly depend on the content of the source BPM. However, other approaches to automatic CDM synthesis are also dependent on the quality of the used source specification, regardless of whether it is textual or graphical. The experiment [19, 20] showed that a source BPM, as a result of the disciplined business modeling approach (respecting used objects and object flows), may constitute a reliable starting base for automatic CDM synthesis.

Apart from the assessment of the effectiveness and efficiency of the implemented tool, its usability could also be evaluated from the database designer's perspective. Currently we do not have any quantitative evaluation results, but we can compare the implemented system against the existing tools aimed to MDSDM. The implemented client application, which consumes the online CDM generator service, constitutes the first online publicly available tool for automatic BPM-driven CDM synthesis, which can be used for automatic synthesis of the initial CDM based on BPMs represented by two different concrete notations (BPMN and UML AD). After downloading the automatically generated CDM, a designer is able to use it in other modeling tools/platforms, without any installation and customization of additional tools and/or plugins in contrast to the existing approaches.

Validation from the developers' perspective is the process of checking whether the proposed approach and implemented online system (including all related services) satisfy the developers' needs.

In the context of the main goal of this research – development of an approach that enables implementation of an online service for automatic CDM synthesis based on BPMs represented by different concrete notations with the minimized dependency on the platform/vendor specificities, we can conclude that the implemented online service satisfies the developers' needs. Firstly, the proposed approach enables service-oriented architecture of the online system. Such a modular architecture enables separation of concerns and concurrent development of different services included in the orchestration, which further brings other related benefits. Secondly, the publicly available online CDM generator service enables other developers to simply consume it in their own applications and modeling platforms without any installation and customization of additional tools and/or plugins in contrast to the existing approaches. This could be very beneficial for researchers and other categories of developers.

10. Conclusions

In this article we presented an approach that enables automated CDM synthesis independently of different starting BPM notations. We identified BPM concepts having semantic potential for automated CDM synthesis, and we specified a simple DSL named BMRL for the representation of those characteristic concepts. With the introduction of DSL, the CDM synthesis is split into two phases. In the first phase, the specified concepts are extracted from the source BPM and represented by BMRL. In the second phase, the BMRL-based representation of the extracted BPM concepts is used for the automated generation of the target CDM. Each phase is based on a set of formal transformation rules enabling automatization of the whole process.

The proposed approach has several advantages over the existing approaches since it enables splitting of the CDM synthesis into two different phases. The first phase deals only with the extraction of the characteristic concepts from the source BPM independently of the target CDM synthesis, while the second phase only deals with the target CDM synthesis independently of the source BPM extraction. This approach reduces the CDM synthesis dependency on the source BPM notations that are caused by the metamodel changes and/or vendor specific implementations as well. If some source BPM notation is changed, then only the corresponding BPM extractor is to be changed. If some modifications of the generation rules are necessary, then only the CDM generator is to be modified, while the BPM extractors remain unchanged. Thus, the proposed approach facilitates the implementation of the required tools and simplifies the target CDM synthesis.

The proposed approach enables implementation of modular tools for BPM-driven CDM synthesis, which consist of loosely coupled components aimed at automatic extraction of specific concepts from the source models and automatic generation of the target model. Based on the proposed approach, we implemented the first online BPM-driven CDM generator as a web-based, platform- and source notation-independent tool. Currently, it enables automatic generation of the target CDM represented by the UML class diagram, based on BPMs represented by two concrete notations: BPMN and UML activity diagram. Its usage can be twofold. Firstly, database designers are able to use it through the implemented client application, which enables the source BPM upload and target CDM download – after downloading the automatically generated CDM, a

designer is able to use it in other modeling tools/platforms, without any installation and/or customization of additional tools/plugins in contrast to the existing approaches. Secondly, developers are able to consume the exposed web service from their own applications and modeling platforms, which could be very beneficial for researchers and other categories of developers.

Since the proposed approach to two-phase BPM-driven CDM synthesis is based on the identified semantic capacity of BPMs for the direct synthesis, which has previously been experimentally confirmed, the implemented online CDM generator is characterized by the same effectiveness and efficiency as the existing, experimentally evaluated, direct ATL-based generator. This means that the generator enables automatic generation of the target conceptual database model with very high completeness and precision: the average effectiveness was $\sim$80% for automatic generation of classes, and $\sim$85% for associations; the average recall of the generated model was above 80%; the average precision for automatically generated classes was above 75%, while the average precision for associations was about 90%. The experiments imply that the automatically generated model can also be efficiently used as a starting point for a manual design of the target model, since it significantly shortens the time required for design – the calculated speed-up factors confirm that the manual design, which takes the automatically generated model as a starting base, almost bisects the estimated efforts and actual time spent to obtain the target model in contrast to the manual design from scratch.

Our future work will focus on further identification of the semantic capacity of BPMs for automatic CDM synthesis, as well as improvements of the implemented tools, particularly BPM extractors in order to minimize vendor specificities. We also intend to provide visualization and editing functionalities of automatically generated models in the web browser.

References

1. Aguilar, J.A., Garrigós, I., Mazón, J.N., Trujillo, J.: An MDA approach for goal-oriented requirement analysis in web engineering. Journal of Universal Computer Science 16(17), 2475–2494 (2010)
2. Alencar, F., Marín, B., Giachetti, G., Pastor, O., Pimentel, J.H.: From i* requirements models to conceptual models of a model driven development process. In: Persson, A., Stirna, J. (eds.) POEM 2009, LNBIP, vol. 39, pp. 99–114. Springer, Berlin Heidelberg (2009)
3. Alencar, F., Pedroza, F., Castro, J., Amorim, R.: New mechanisms for the integration of organizational requirements and object oriented modeling. In: Proc. of WER 2003. pp. 109–123 (2003)
4. Alencar, F.M.R., Filho, G.A.C., Castro, J.F.: Support for structuring mechanism in the integration of organizational requirements and object oriented modeling. In: Proc. of WER 2002. pp. 147–161 (2002)
5. Alencar, F.M.R, Pedroza, F.P., Castro, J., Silva, C.T.L., Ramos, R.A.: XGOOD: A tool to automatize the mapping rules between i* framework and UML. In: Proc. of CIbSE 2006. pp. 125–138 (2006)
6. Ang, C.L., Khoo, L.P., Gay, R.K.L.: IDEF*: a comprehensive modelling methodology for the development of manufacturing enterprise systems. Int. Journal of Production Research 37(17), 3839–3858 (1999)
7. Banjac, D., Brdjanin, D., Banjac, G., Maric, S.: Evaluation of automatically generated conceptual database model based on collaborative business process model: Controlled experiment. In:

Stojanov, G., Kulakov, A. (eds.) ICT Innovations 2016, AISC, vol. 665, pp. 134–145. Springer (2016)
8. Becker, L.B., Pereira, C.E., Dias, O.P., Teixeira, I.M., Teixeira, J.P.: MOSYS: A methodology for automatic object identification from system specification. In: Proc. of ISORC 2000. pp. 198–201. IEEE Computer Society (2000)
9. Bloomfield, T.: MDA, meta-modelling and model transformation: Introducing new technology into the defence industry. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005, LNCS, vol. 3748, pp. 9–18. Springer, Berlin Heidelberg (2005)
10. Boccalatte, A., Giglio, D., Paolucci, M.: ISYDES: the project of a tool aimed at information system development. In: Proc. of AIWORC 2000. pp. 293–298. IEEE (2000)
11. Brambilla, M., Cabot, J., Comai, S.: Automatic generation of workflow-extended domain models. In: Engels, G., et al. (eds.) MoDELS 2007, LNCS, vol. 4735, pp. 375–389. Springer, Berlin Heidelberg (2007)
12. Brambilla, M., Cabot, J., Comai, S.: Extending conceptual schemas with business process information. Advances in Software Engineering, vol. 2010, Article ID 525121 (2010)
13. Brdjanin, D., Maric, S.: Towards the initial conceptual database model through the UML metamodel transformations. In: Proc. of Eurocon 2011. pp. 1–4. IEEE (2011)
14. Brdjanin, D., Maric, S.: An Approach to Automated Conceptual Database Design Based on the UML Activity Diagram. Computer Science and Information Systems 9(1), 249–283 (2012)
15. Brdjanin, D., Maric, S.: Model-driven Techniques for Data Model Synthesis. Electronics 17(2), 130–136 (2013)
16. Brdjanin, D., Maric, S., Gunjic, D.: ADBdesign: An approach to automated initial conceptual database design based on business activity diagrams. In: Catania, B., Ivanovic, M., Thalheim, B. (eds.) ADBIS 2010, LNCS, vol. 6295, pp. 117–131. Springer, Berlin Heidelberg (2010)
17. Brdjanin, D., Banjac, D., Banjac, G., Maric, S.: An approach to automated two-phase business model-driven synthesis of data models. In: Ouhammou, Y., et al. (eds.) Model and Data Engineering, LNCS, vol. 10563, pp. 57–70. Springer (2017)
18. Brdjanin, D., Banjac, D., Banjac, G., Maric, S.: An online business process model-driven generator of the conceptual database model. In: 8th International Conference on Web Intelligence, Mining and Semantics – WIMS'18. pp. 16:1–16:9. ACM (2018)
19. Brdjanin, D., Banjac, G., Banjac, D., Maric, S.: Controlled experiment in business model-driven conceptual database design. In: Reinhartz-Berger, I., et al. (eds.) Enterprise, Business-Process and Information Systems Modeling, LNBIP, vol. 287, pp. 289–304. Springer (2017)
20. Brdjanin, D., Banjac, G., Banjac, D., Maric, S.: An experiment in model-driven conceptual database design. Software & Systems Modeling pp. 1–25 (2018)
21. Brdjanin, D., Banjac, G., Maric, S.: Automated synthesis of initial conceptual database model based on collaborative business process model. In: Bogdanova, M.A., Gjorgjevikj, D. (eds.) ICT Innovations 2014: World of Data, AISC, vol. 311, pp. 145–156. Springer International Publishing, Cham (2015)
22. Brdjanin, D., Maric, S.: On automated generation of associations in conceptual database model. In: De Troyer, O., et al. (eds.) ER Workshops 2011, LNCS, vol. 6999, pp. 292–301. Springer-Verlag, Berlin Heidelberg (2011)
23. Brdjanin, D., Maric, S.: Towards the automated business model-driven conceptual database design. In: Morzy, T., Harder, T., Wrembel, R. (eds.) Advances in Databases and Information Systems, AISC, vol. 186, pp. 31–43. Springer-Verlag, Berlin Heidelberg (2012)
24. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.: Eclipse Modeling Framework. Pearson Education, Boston, USA (2003)
25. Castro, J.F., Alencar, F.M.R., Filho, G.A.C., Mylopoulos, J.: Integrating organizational requirements and object oriented modeling. In: Proc. of ISRE 2001. pp. 146–153. IEEE (2001)
26. Cruz, E.F., Machado, R.J., Santos, M.Y.: From business process modeling to data model: A systematic approach. In: Proc. of QUATIC 2012. pp. 205–210. IEEE (2012)

27. Cruz, E.F., Machado, R.J., Santos, M.Y.: Deriving a data model from a set of interrelated business process models. In: Proc. of ICEIS 2015. pp. 49–59 (2015)
28. de la Vara, J.L.: Business process-based requirements specification and object-oriented conceptual modelling of information systems. PhD Thesis, Valencia Polytechnic Uni. (2011)
29. Drozdova, M., Kardos, M., Kurillova, Z., Bucko, B.: Transformation in model driven architecture. In: Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology – ISAT 2015 – Part I. pp. 193–203. Springer, Cham (2016)
30. Drozdová, M., Mokryš, M., Kardoš, M., Kurillová, Z., Papán, J.: Change of paradigm for development of software support for elearning. In: Proc. of ICETA 2012. pp. 81–84. IEEE (2012)
31. España, S.: Methodological integration of communication analysis into a model-driven software development framework. PhD Thesis, Valencia Polytechnic Uni. (2011)
32. Essebaa, I., Chantit, S.: Toward an automatic approach to get pim level from cim level using qvt rules. In: 2016 11th International Conference on Intelligent Systems: Theories and Applications (SITA). pp. 1–6. Mohammedia (2016)
33. Fernandes, J.M., Lilius, J., Truscan, D.: Integration of DFDs into a UML-based model-driven engineering approach. Software and Systems Modeling 5(4), 403–428 (2006)
34. Fouad, A.: Embedding requirements within the model driven architecture. PhD Thesis, Bournemouth Uni. (2011)
35. Insfran, E., Pastor, O., Wieringa, R.: Requirements Engineering-Based Conceptual Modelling. Requirements Engineering 7(2), 61–72 (2002)
36. Insfran, E.: Requirements engineering approach for object-oriented conceptual modeling. PhD Thesis, Valencia Polytechnic Uni. (2003)
37. Jiang, L., Topaloglou, T., Borgida, A., Mylopoulos, J.: Goal-oriented conceptual database design. In: Proc. of RE '07. pp. 195–204. IEEE, Los Alamitos, USA (2007)
38. Jouault, F., Allilaire, F., Bezivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(1-2), 31–39 (2008)
39. Koch, N.: Transformation Techniques in the Model-Driven Development Process of UWE. In: Proc. of the Workshops at ICWE'06, Art. No. 3. ACM (2006)
40. Koch, N., Zhang, G., Escalona, M.J.: Model Transformations from Requirements to Web System Design. In: Proc. of ICWE'06. pp. 281–288. ACM (2006)
41. Koskinen, J., Peltonen, J., Selonen, P., Systa, T., Koskimies, K.: Model processing tools in UML. In: Proc. of ICSE 2001. pp. 819–820. IEEE Computer Society (2001)
42. Kriouile, A., Addamssiri, N., Gadi, T.: An MDA Method for Automatic Transformation of Models from CIM to PIM. American Journal of Software Engineering and Applications 4(1), 1–14 (2015)
43. Lingzhi, L., Ang, C.L., Gay, R.K.L.: Integration of Information Model (IDEF1) with Function Model (IDEF0) for CIM Information System Design. Expert Systems with Applications 10(3/4), 373–380 (1996)
44. Liu, D., Subramaniam, K., Far, B., Eberlein, A.: Automating Transition from Use-cases to Class Model. In: Proc. of CCECE 2003. pp. 831–834. IEEE (2003)
45. Martinez Rebollar, A.: Conceptual schemas generation from organizational models in an automatic software production process. PhD Thesis, Valencia Polytechnic Uni. (2008)
46. Nikiforova, O., Gusarovs, K., Gorbiks, O., Pavlova, N.: BrainTool: A tool for generation of the UML class diagrams. In: Proc. of ICSEA 2012. pp. 60–69. IARIA (2012)
47. Nikiforova, O., Gusarovs, K., Gorbiks, O., Pavlova, N.: Improvement of the two-hemisphere model-driven approach for generation of the uml class diagram. Applied Computer Systems 14(1), 19–30 (2013)
48. Nikiforova, O., Pavlova, N.: Application of BPMN instead of GRAPES for two-hemisphere model driven approach. In: Grundspenkis, J., et al. (eds.) ADBIS 2009 Workshops, LNCS, vol. 5968, pp. 185–192. Springer, Berlin Heidelberg (2010)

49. OMG: MOF 2.0 Query/View/Transformation Specification, v1.0. OMG (2008)
50. OMG: Business Process Model and Notation (BPMN), v2.0. OMG (2011)
51. OMG: Unified Modeling Language (OMG UML), v2.5. OMG (2015)
52. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: Analysis-level classes from secure business processes through model transformations. In: Lambrinoudakis, C., Pernul, G., Tjoa, A.M. (eds.) TrustBus 2007, LNCS, vol. 4657, pp. 104–114. Springer, Berlin Heidelberg (2007)
53. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: Towards obtaining analysis-level class and use case diagrams from business process models. In: Song, I.Y., et al. (eds.) ER Workshops 2008, LNCS, vol. 5232, pp. 103–112. Springer, Berlin Heidelberg (2008)
54. Rodriguez, A., Garcia-Rodriguez de Guzman, I., Fernandez-Medina, E., Piattini, M.: Semi-formal transformation of secure business processes into analysis class and use case models: An MDA approach. Information and Software Technology 52(9), 945–971 (2010)
55. Rungworawut, W., Senivongse, T.: Using ontology search in the design of class diagram from business process model. PWASET 12, 165–170 (2006)
56. Santos, M.Y., Oliveira e Sá, J.: A Data Warehouse Model for Business Processes Data Analytics, pp. 241–256. Springer International Publishing, Cham (2016)
57. Santos, M.Y., Machado, R.J.: On the derivation of class diagrams from use cases and logical software architectures. In: Proc. of ICSEA '10. pp. 107–113. IEEE (2010)
58. Selonen, P., Koskimies, K., Sakkinen, M.: Transformations Between UML Diagrams. Journal of Database Management 14(3), 37–55 (2003)
59. Silva, L.F., Leite, J.C.S.P.: Generating requirements views: A transformation-driven approach. Electronic Communications of the EASST 3, 1–14 (2006)
60. Srivastava, S.: Model transformation approach for a goal oriented requirements engineering based webgrl to design models. International Journal of Soft Computing and Engineering (IJSCE) 3(6), 66–75 (2014)
61. Tan, H.B.K., Yang, Y., Blan, L.: Systematic transformation of functional analysis model in object oriented design and implementation. IEEE Transaction on Software Engineering 32(2), 111–135 (2006)
62. Truscan, D., Fernandes, J.M., Lilius, J.: Tool support for DFD-UML based transformation. In: Proc. of ECBS '04. pp. 378–387. IEEE (2004)
63. Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L., Visser, E., Wachsmuth, G.: DSL Engineering – Designing, Implementing and Using Domain-Specific Languages (2013)
64. Wrycza, S.: The ISAC-driven transition between requirements analysis and ER conceptual modelling. Information Systems 15(6), 603–614 (1990)
65. Zhang, J., Feng, P., Wu, Z., Yu, D., Chen, K.: Activity based CIM modeling and transformation for business process systems. International Journal of Software Engineering and Knowledge Engineering 20(3), 289–309 (2010)

Drazen Brdjanin is an Associate Professor at the Faculty of Electrical Engineering, University of Banja Luka (Bosnia and Herzegovina), where he heads the M-lab Research Group. His research interests are mainly related to databases and model-driven software development. He was participating in several R&D projects at national and international level, and authoring a number of research papers and articles in the field of model-driven development and database design.

Danijela Banjac is a Senior Teaching Assistant and PhD student at the Faculty of Electrical Engineering, University of Banja Luka (Bosnia and Herzegovina). She is a member

of M-lab Research Group. Her research interests include model-driven software development, business process modelling, object-oriented information systems, and UML. She has published several research papers and articles.

Goran Banjac is a Senior Teaching Assistant and PhD student at the Faculty of Electrical Engineering, University of Banja Luka (Bosnia and Herzegovina). He is a member of M-lab Research Group. His research interests include model-driven software development, business process modelling, databases, and UML. He has published several research papers and articles.

Slavko Maric is a Full Professor at the Faculty of Electrical Engineering, University of Banja Luka (Bosnia and Herzegovina), where he heads the Computer Science Department. His current research interests include: information systems modeling, design and development, databases, eGovernment systems, service oriented architecture and parallel processing. He has published over 50 research papers and articles, and participated in a number of research and development projects.

Received: October 10, 2018; Accepted: June 2, 2019.