UDC 004.43, DOI: 10.2298/CSIS0901023K

Model Driven Engineering of a Tableau Algorithm for Description Logics

Nenad Krdžavac[2], Dragan Gašević[1], and Vladan Devedžić[2]

[1]School of Computing and Information Systems, Athabasca University, Canada
[2]FON–School of Business Administration, University of Belgrade, Serbia
nenadkr@tesla.rcub.bg.ac.yu, dgasevic@acm.org, devedzic@fon.rs

Abstract. This paper presents a method for implementing tableau algorithm for description logics (DLs). The architectures of the present DL reasoners such as RACER or FaCT were developed using programming languages as Java or LISP. The implementations are not based on original definition of the abstract syntax, but they require transformation of abstract syntax into concrete syntax implementation languages use. In order to address these issues, we propose the use of model-driven engineering principles for the development of a DL reasoner where a definition of a DL abstract syntax is provided by means of metamodels. The presented approach is based on the use of a MOF-based model repository and QVT-like transformations, which transform models compliant to the DL metamodel taken from the OMG's Ontology Definition Metamodel specification into models compliant to the Tableau metamodel defined in this paper.

Keywords: Description Logics, Model Driven Architecture, Tableau Algorithm.

1. Introduction

Model-driven engineering (MDE) introduces a software development shift form the programming-centered paradigm to the model-driven paradigm [3]. Models, as the first class citizens in MDE, allow developers to build their models at different levels of abstraction. Models also reduce the costs of reusing the present software artifacts in new software solutions [21]. Having in mind the fact that models should constantly be transformed from one form (e.g., Platform Independent Model) into another one (e.g., Platform Specific Model), model transformations has been recognized as the crucial component for MDE [3].

Being developed in parallel with MDE, the concept of the Semantic Web tries to semantically interconnect Web resource using knowledge representation techniques, more specifically ontologies [25]. By using ontologies, the Semantic Web provides potentials for developing intelligent services based on logic-based reasoning. That is the reason why the present

ontology languages such as OWL have been grounded on description logics (DLs). In fact, the core of the Semantic Web is to implement DL-based reasoners that are able to reason over ontologically represented knowledge. Some publicly available DL reasoners [7], [6], [15] are mainly implemented using Java or Lisp. However, extending their functionality to support new DL types or deploying them to other platforms such as .NET can be a very tedious and time consuming task. This is due to the fact that various types of DL have different abstract syntax, which should be mapped to concrete syntax of the implementation platform or language. This, in fact, switches the focus from an actual problem to low level platform specific details. Thus, producing a reasoner takes longer and it does not possess any mechanisms that checks whether the implemented reasoner is compliant with the abstract syntax of the DL type being implemented.

The objective of this paper is to show how MDE techniques can be applied to developing a DL reasoner. In fact, we propose the method of implementing of the tableau algorithm that tries to compute the basic reasoning service of DLs (i.e., satisfiability), which is used as a basis for other reasoning services, namely subsumption, consistency, and instance checking. To implement the tableau algorithm using the MDE principles, we used the following two MOF-based metamodels:

1. The DL metamodel that is a non-normative part of the current OMG's (www.omg.org) effort for developing the Ontology Definition Metamodel (ODM) [24];
2. The Tableau metamodel that represent a tableau.

Basically, our proposed implementing method for the DL tableau algorithm means transforming a DL knowledge base (i.e., an instance of the DL metamodel in terms of the MDA) into its tableau model (i.e., an instance of the Tableau metamodel). For that transformation we use Atlas Transformation Language (ATL), a QVT-like model transformation language [8].

The proposed implementation of the tableau algorithm is developed in the context of AIR [5]. It is a framework that uses MDA metamodelling principles for developing intelligent systems based upon different knowledge representations [5]. The central part of AIR is a model-repository [22] that various kinds of intelligent systems, as well as models of any other domain of interest. The other important part of AIR is an integrated development environment with a rich GUI for specifying the AIR workbench. The AIR workbench is built on top of the Eclipse (www.eclipse.org) plug-in architecture, today's leading extensible platform, which supports many relevant MDE efforts such as Eclipse modeling Framework (EMF) and Generative Model Transformer (GMT) (www.eclipse.org).

The next section defines one specific type of DLs, which is currently implemented by the tableau algorithm. In section 3, we describe the basic concepts of the MDA. Section 4 describes the metamodelling foundation of the tableau algorithm, by defining the DL metamodel and the Tableau metamodel. We present a method for implementing the tableau algorithms for description logics based on the ATL [8] that is an alternative to the OMG's MOF2 QVT standard [14] for model transformations. Section 6 gives a

discussion of the related work, while the final section provides concluding remarks and directions of the future research.

2. Description logics and tableau algorithm

Historically, DLs evolved from semantic networks and frame systems, mainly to satisfy the need of giving a formal semantics to these formalisms [2]. As the name DLs indicates, one of characteristics of these languages is that they are equipped with formal logic-based semantics. The basic notions in DLs are concepts (unary predicates) and roles (binary predicates) [2]. DLs are logic formalisms used as a basis for the Semantic Web ontology languages (e.g., OWL) [18] and they offer reasoning services [2], which can be applied to reasoning with ontologies. Reasoning is important to ensure the quality of ontology [18]. In this section we define syntax and semantics of ALC and define the tableau algorithm.

2.1 The ALC description logic

The smallest propositionally closed DL is the ALC DL [2]. According to [19], syntax and semantics of the language is given by the following two definitions:
Definition 1. (Syntax of ALC language).
Let N_C and N_R be disjoint and countable infinite set of concepts and role names. The set of ALC-concepts is the smallest set, such that:

1. Every concept name $A \in N_C$ is an ALC concept.
2. If C and D are ALC concepts and $R \in N_R$ then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, and $\forall R.C$ are ALC concepts.

Every ALC formula can be constructed applying 1 and 2 rule.

Definition 2. (Semantics of ALC language).
An ALC interpretation is a pair $(\Delta^I, \cdot^I)$ where Δ^I is a non-empty set called domain, and $\cdot^I$ is an interpretation function that maps every concept name A to a subset A^I of Δ^I and every role name to a binary relation R^I over Δ^I. The interpretation function is extended to complex concepts as follows:

1. $(\neg C)^I = \Delta^I \setminus C^I$,
2. $(C \sqcap D)^I = C^I \cap D^I$,
3. $(C \sqcup D)^I = C^I \cup D^I$,
4. $(\exists R.C)^I = \{d \in \Delta^I \mid (\exists e)((d,e) \in R^I \wedge e \in C^I)\}$,
5. $(\forall R.C)^I = \{d \in \Delta^I \mid (\forall e)((d, e) \in R^I \Rightarrow e \in C^I\}$.

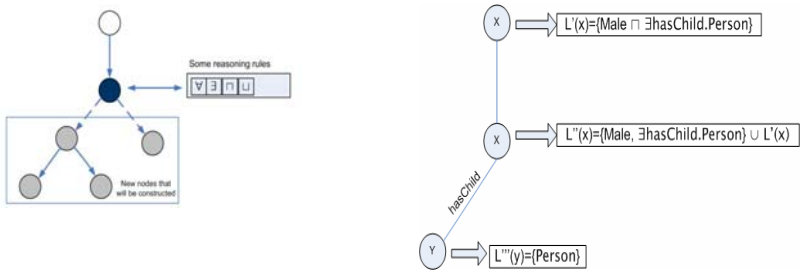
Example 1.
Suppose that the nouns Human and Male are concept names and hasChild is the role name, then the ALC concept (Human ⊓ ∃hasChild.T) represents all persons that have a child, while the concept (Human ⊓ ∀hasChild.Male) represents all persons with all children being males.□

Some extensions of the language are given in [7]. A knowledge base developed by using DLs consists of two components TBox and ABox [2]. TBox contains terms (formulas) that define concepts and describe relationships between concepts over roles (binary predicates). On the other hand, the name of assertions is ABox that represents named individuals expressed in terms of the concepts and roles [2]. A concept described in TBox is similar to a class defined in an object-oriented language or UML and roles are similar to relationships (binary) between classes, but without behavioral components. Individuals in ABox can be viewed as objects in object-oriented programming or UML, without behavioral components.

DLs terms satisfy DeMorgans's laws [7]. This means that every concept expression can be transformed into negation normal form (NNF). For example, the negation normal form of the concept ¬(Male ⊓ Female) is concept (¬Male ⊔ ¬Female). According to [7], that negation applies only to atomic concepts and not to composite concept terms. It is important in the case of reasoning with anontology.

2.2 The tableau algorithm

According to [2], basic reasoning services in DLs are: subsumption, consistency, satisfiability, and instance checking. Satisfiability of a concept expression C is a problem of checking whether there exists a model [7]. It means whether exists an interpretation I (explained in Sect. 2.1) in which $C^I \neq \emptyset$. In that context the interpretation I is a model for a concept C. Other reasoning services can be calculated with satisfiability. The tableau algorithm tries to prove satisfiability of a concept term C, by demonstrating a model in which C can be satisfied [7]. A *tableau* is a graph that represents such a model (see Fig. 1a), with nodes corresponding to individuals and edges corresponding to relationships between individuals [7]. Every DLs term can be represented by a tree (a special case of graph) structure, but a tableau has different structure then a DL formula. Fig. 1b shows the process of building a tableau for DLs. Before constructing a tableau for a concept term, we must transform such term into NNF [7].



a) b)

Fig. 1. The tree of tableau: a) general principle; b) an example of a tableau for DLs

According to [7], the tableau algorithm starts with a single individual (Fig. 1) that satisfies an arbitrary concept C. New nodes are created according to expansion rules [2]. These rules are different in different DLs. Expansion rules define rules of building new nodes. Every node is connected to a set of concept terms. Expansion rules for the ALC DL are described in [7].

3. Model-Driven Architecture

The Model Driven Architecture (MDA) is defined as a realization of MDE principles proposed by the Object Management Group (OMG) [11]. Models play a major role in the MDA. The most general definition of a model says that a model is a simplified view of reality [21], or, more formally, a model is a set of statements about a system under study [20]. In fact, one can say that a model is a clear set of formal elements that describes something being developed for a specific purpose and can be analyzed using various methods [11]. Metamodels are another key concept used in the MDA. A metamodel is a specification model for a class of systems under study, where each system under study in the class is itself a valid model expressed in a certain modeling language. A metamodel makes statements about what can be expressed (i.e., asserted) in the valid models of a certain modeling language. Basically, a metamodel defines what well-formed models are, i.e., models should conform to their metamodels.

The central part of the MDA is the four-layer modeling architecture (see Fig. 2). The topmost layer (M3) is called the metametamodel layer. The OMG has defined a standard in this layer – the MOF (Meta-Object Facility). According to [10], the MOF is the language intended for defining metamodels in the M2 layer. The rationale for having these four levels with one common metametamodel is to enable both the use and the generic managing of many models and metamodels, and to support their extensibility and integration. Examples of standard MOF-defined metamodels are the metamodel of UML and the Ontology Definition Metamodel (ODM) [13].

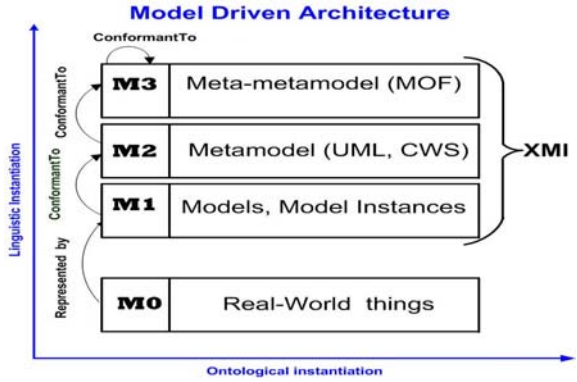


Fig. 2. Four Layer Architecture of MDA (see [3])

The next layer is the model layer (M1) – the layer where we develop real-world models. In terms of UML, this means classes, their relationships, and objects. In fact, here we refer to a modified MDA four-layer architecture where classes and their relations are defined in the M1 layer, while real-world things are in the M0 layer [1, 3]. In the original MDA proposal [16], instances of models (e.g. UML objects) were resided in the M0 layer. Defining two different types of instantiation relations, namely linguistic and ontological, Atkinson and Kühne proved why models should reside in the M1 layer [1].

There is an XML-based standard for sharing metadata that can be used for all of the MDA's layers. This standard is called XML Metadata Interchange – XMI [12] (Fig. 2). The meaning of XMI is twofold, i.e., it is a set of rules for serialization of MOF-compliant models (e.g., UML models) and a set of rules for generation of schema for each MDA layer (e.g., the UML XMI schema).

A set of reflective APIs consisting of reflective interfaces has been defined for the MOF in order to enable the management of MOF-based models, metamodels, and metametamodels in programs. Java Metadata Interfaces (JMI) [4] is a realization of the standard called JSR040 [4], and JMI defines Java programming interfaces for manipulating MOF-based models and metamodels [4]. JMI interfaces allow users to create, update, and access instances of metamodels in Java.

Model transformations have been identified as the crucial technology for achieving the MDA goals of model driven software development [3]. Responding to that need, the OMG has adopted a standard for model transformations called MOF2 Query/View/Transformation (QVT) [14]. Three vital subjects of the proposal that ensure the full realization of MDA are:

1. Queries: Take a model as input, and select specific elements from that model;
2. Views: Represent models that are derived from other models;
3. Transformations: Take a model as input and update it or create a new model.

Since the MOF2 QVT specification has lately been adopted, it is not fully supported by the present tools. However, we can instead use model

transformation engines that have been based on the model transformation languages developed by some of the major MOF2 QVT contributions such as ATL [8].

Besides the MDA's MOF-based architecture, there is also another well-known framework developed called the EMF [27]. The main difference is that the EMF is based on another metametamodel, Ecore, which is equivalent to the MDA's MOF. The EMF also supports generation of Java-based interfaces for managing Ecore defined metamodels and models. The EMF also supports XMI serialization of Ecore compliant models.

4. Metamodelling foundation

Our DL reasoner is based on the DL metamodel proposed by the OMG's working group on the ODM [13] and on the Tableau metamodel that we developed. In this section we describe both metamodels. Fig. 3 shows the place of these two metamodels in the MDA four-layer architecture.

Differences between the DL metamodel and the Tableau metamodel (Fig. 4) are:

1. A model (in terms of the MDA) of a tableau has a different structure than a model (in terms of the MDA) of a DL formula.
2. The DL metamodel describes how to build DL terms in a knowledge base. For both representations we use the XMI format.

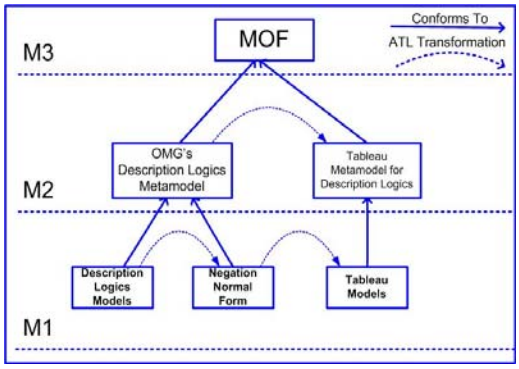


Fig. 3. Metamodelling foundation of the DL reasoner

4.1 Description Logic metamodel

The DL metamodel [13] is defined by the MOF2 language and graphically represented by using UML2's graphical notation. In an initial draft of the ODM specification [13], the DL metamodel was a central part of the ODM architecture [13]. However, the DL metamodel is not a central one, and it is

now defined as a non-normative metamodel [24]. Although, the DL metamodel is non-normative, it is still a suitable reference for describing DLs using the MOF. Furthermore, from the implementation point of view it is not important, because the meaning of the metamodel has not been changed.

Fig. 4 gives an excerpt of the DL metamodel taken from the ODM specification [24]. The top-level concept of the DL metamodel metaclass hierarchy is the Term metaclass (see Fig. 4). The metaclass Term corresponds to definition of the DL syntax (see Sect. 2.1 – Def. 1). The DL metamodel [13] supports a DLs that is a logical basis for OWL [19].

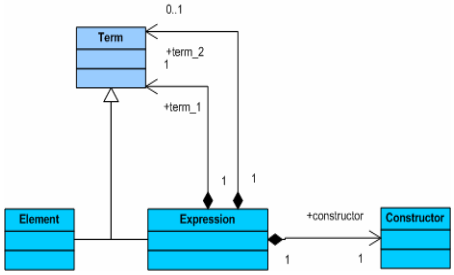


Fig. 4. A part of the OMG's DL metamodel [13]

The design pattern Composite has been used in building DLs terms (formulas). The pattern consists of the metaclasses Term, Expression and Element. The metaclasses that are used in building concepts and roles are inherited from the Element metaclass. Logical constructors are represented by metaclasses that are inherited from the Constructor metaclass. The metamodel supports metaclasses that represent TBox and Abox of a knowledge base [13]. The metaclass TBox is connected with metaclass Term via an association relation (Fig. 5).

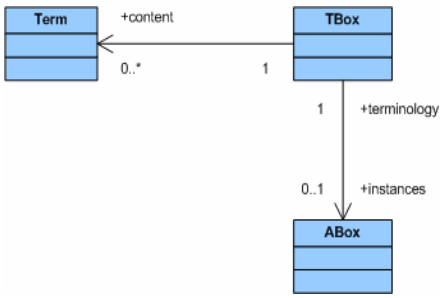


Fig. 5. The TBox and ABox metaclasses defined in the OMG's ODM specification [13]

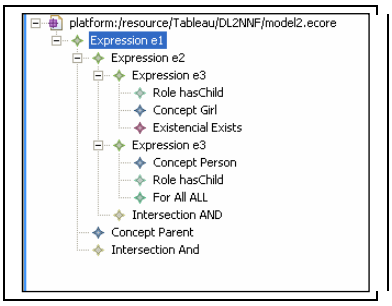


Fig. 6. A DL model represented in Ecore editor for Eclipse

According to the proposed DLs metamodel, we may build models saved in EMF [27] repository. Fig 6 illustrates an Ecore-based DL model of Father which has at least one daughter. It is intersection relation between concepts Parent and concept e3. The first concept e3 (Fig. 6) says that among all children, at least one is daughter, but the second one says that all children are persons. XML representation of the model is presented in Fig. 7.

```
<?xml version="1.0" encoding="ASCII"?>
<Expression xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="dl" Identifier="e1">
  <term_1 xsi:type="Expression" Identifier="e2">
    <term_1 xsi:type="Expression" Identifier="e3">
      <term_1 xsi:type="Role" Identifier="hasChild"/>
      <otherTerm xsi:type="Concept" Identifier="Girl"/>
      <constructor xsi:type="Existencial" Identification="Exists"/>
    </term_1>
    <otherTerm xsi:type="Expression" Identifier="e3">
      <term_1 xsi:type="Concept" Identifier="Person"/>
      <otherTerm xsi:type="Role" Identifier="hasChild"/>
      <constructor xsi:type="ForAll" Identification="ALL"/>
    </otherTerm>
    <constructor xsi:type="Intersection" Identification="AND"/>
  </term_1>
  <otherTerm xsi:type="Concept" Identifier="Parent"/>
  <constructor xsi:type="Intersection" Identification="And"/>
</Expression>
```

Fig. 7. XML representation of a DL model from Fig. 6

4.2 The Tableau metamodel

We have developed our own Tableau metamodel using MOF, while we used the UML graphical notation to represent graphically that metamodel. The tableau algorithm uses a tree (T) to represent the model being constructed [7],

hence the Composite design pattern [17] can be used in describing the core concepts of the Tableau metamodel.

The Tableau metamodel can be logically divided in two parts. The first part comprises metaclasses that represent concept terms in DLs, and correspond to metaclasses in the DL metamodel in Fig. 5. Some of them are:

1. *Formula* that represents common DL terms;
2. *AtomicFormula* that represents atomic concepts in DLs like roles and concepts;
3. *SubFormula* represents a part of the common DLs formula. It is similar to the concept of subformula in propositional logics. For example, there are three subformulas in concept Human □ ∃hasChild.Male like Human, ∃hasChild.Male, and Male.

These metaclasses have the same structure like metaclasses Term, Element and Expression in the DL metamodel [13].

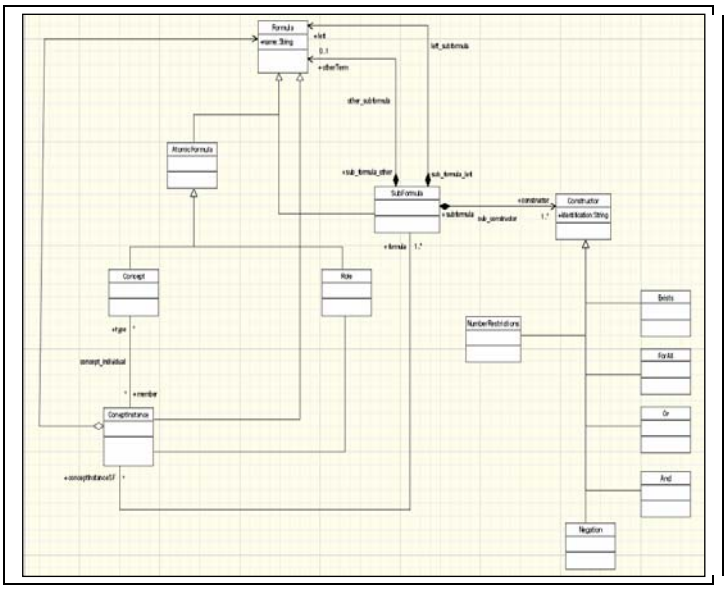


Fig. 7. A Tableau metamodel[1]

The second part of the Tableau metamodel consists of the metaclasses and their relationships that represent the model (in terms of the DL definition), which is constructed according to expansion rules described in [7]. The metaclass *ConceptInstance* represents all nodes in such a model. Every node corresponds to some set of concept terms. This is represented by the association (aggregation) between the metaclasses *Formula* and *ConceptInstance* (Fig. 7). Every role connects at least two individuals and that

1 Tableau metamodel is developed using Poseidon UML tool (www.gentleware.com)

is represented by the association between the metaclasses *Role* and *ConceptInstance*.

The Tableau metamodel (Fig. 7) supports only the ALC DL. [7] The metamodel is an extension of the metamodel published in [28]. The support is provided by the metaclasses that represent logical constructors like Or, Not, And, ForAll, Exists, NumberRestrictions.

5. The implementation of the Tableau metamodel

Fig. 8 shows the basic idea of the architecture of the reasoner. The repository contains models and both the DL and Tableau metamodels. Generated JMI interfaces and ECore classes (http://www.eclipse.org/modeling/emft/) are used to deal with the models. The reasoner functionality can be extended by using abstract classes that implement JMI interfaces.

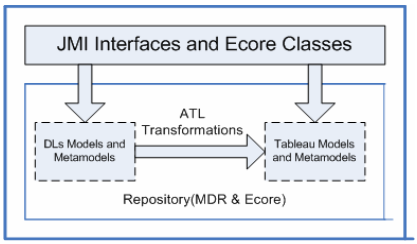


Fig. 8. The architecture of the reasoner

In the previous section we have described metamodels referred to in the first step of our implementation. In the rest of the section we describe other implementation steps.

5.1. Implementation Repository for the Tableau Metamodel

A model repository is used for storing and retrieving models that conform to both DLs and the Tableau metamodels. It is built by using the NetBeans Metadata Repository (MDR) [22], which is based on the JMI specification. To implement the repository for the Tableau metamodel, we performed the following steps:

1. Conversion of the UML metamodel of the Tableau metamodel into an equivalent MOF metamodel and conversion of the MOF-based Tableau metamodel into an Ecore metamodel [8].

Since the MOF uses UML's graphical notation, we developed the Tableau metamodel by using the tool Poseidon (http://www.gentleware.com/) for UML [23], which should be converted in the MOF. Practically, we exported the Tableau metamodel from Poseidon for UML into the UML XMI format. Using the uml2mof.jar tool [22] we converted it into the MOF XMI format.

Furthermore, since the architecture of our reasoner uses the ATL [8] for transforming models, we had to convert the MOF-based Tableau metamodel into the Ecore-based metamodel. For the sake of the conversion of the MOF-based Tableau metamodel into its Ecore-based equivalent we used Eclipse's plug-in for the EMF [27].

2. Generating JMI interfaces for the Tableau metamodel using theNetBeansMDR in order to provide Java interfaces for managing models based on the Tableau metamodel.
3. Instantiation of models based on the Tableau metamodel in theNetBeans MDR.

Performing the above steps, we faced some practical problems related to the implementation of the repository for the Tableau metamodel. Namely, during the generation of JMI interfaces, all OCL constraints (http://www.omg.org/technology/documents/formal/ocl.htm) were ignored and we had to implement them manually. For example, OCL constraint defined in DLs metamodel, like {disjoint}, is ignored during generating JMI interfaces. Manual implementation means that we should use Java, in order to satisfy all of these constraints in model (metamodel).

5.2. Implementation of the tableau algorithm using model transformations

This section indicates some reasons why model transformations is useful approach for the implementation of reasoning algorithms based on the tableau algorithm and describe our initial results in developing model transformations between the DL metamodel and the Tableau metamodel. ATL [8] is an answer to the OMG's QVT RFP (query view transformation) [14]. A plenty of useful details about the language is described in [8]. This section describes only some benefits of using such a language for implementing the reasoning algorithms. Some of these advantages are:

1. ATL can be integrated in the EMF [27].
   ATL is a declarative and hybrid language [8]. The syntax of the language can be integrated into a Java-based environment. It means that the reasoning rules for description logics can be written directly in some Java environment using the expressive power of the language. In the EMF environment, the ATL code can be run and debugged.
2. A transformation model in ATL is a set of transformation rules and Boolean operations.
   Reasoning algorithms, based on the tableau for description logics, are based on a set transformation rules [7], including Boolean operations. The ATL language supports set and Boolean operations. The syntax and semantics of ALT are described in [8].
3. ATL is compatible with JMI interfaces.
   The ATL transformation model is first read by using the ATL parser and loaded into Java meta-data repository which is based on a JMI compliant repository. The generated JMI interfaces for the Tableau metamodel can

be integrated into the ATL language and help in the implementation of the reasoning rules. The interfaces support extension of the rules for very expressive DLs. Although, mainly intended to deal with MDA models (based on MOF meta-models and accessible via XMI or JMI), the EMF with integrated ATL should also handle other kinds of models from different technological spaces (e.g. Java programs, XML documents, DBMS artifacts, etc.) [8]. This is important in case of using the DLs' reasoning services in other platforms like intelligent metamodelling frameworks, especially in the case of using such a reasoning machine to reason on UML models (not UML diagrams).

5.2.1 M2-transfromations of DL and Tableau metamodels

The first step in implementing the reasoning algorithm is bridging the DL metamodel and the Tableau metamodel in the M2 layer (see Fig. 3) using the ATL language. Basically, we identified mappings between concepts of two different models, and Table 1 describes the high-level transformations between elements of the DL and Tableau metamodel.

Table 1. Relationships between metaclasses of the DLs and Tableau metamodels

Tableau Metamodel	DL Metamodel
Formula	Term
Atomic Formula	Element
SubFormula	Expression
Concept	Concept
Role	Role
Constructor	Constructor

5.2.2 M1-transfromations of the DL and Tableau metamodels

In the M1 layer (Fig. 3), the DL model in NNF is transformed into the tableau model according to the reasoning expansion rules [7]. In ATL, we define a function for transforming a DL model to NNF of that model (see Fig. 9).

```
helper context DL!DL def : getNegations() : String =
      self.Identifier->collect(e | e.Identifier)->
            asSet()->
                  iterate(UniqueIdentifier; acc : String = '' |
                                          idn +
                                          if idn = '' then
                                             UniqueIdentifier
                                          else
                                             ' and ' + UniqueIdentifer
                                          endif
                        )
;
```

Fig. 9. An excerpt of the ATL transformation that transforms DL models into the negation normal form

The function collects all negations in a DL model. If negations are in front of a Term, we must delete such connections of the Term and connect negations to every atomic concept in the Expression (defined in the DL metamodel) that represents all SubExpressions in that Term (defined in DL metamodel). Basically, this function tries to implement DeMorgans's laws defined in Sect. 2.1.

Using the similar approach, we transform the DL model in NNF into its corresponding tableau model compliant to the Tableau metamodel. Actually, we define a few functions in the *helper* [8] section of the ATL code. The functions return all terms that consist of logical operation intersections and transform such Terms into two sub-Terms. A similar situation is the definition of other functions for other logical terms. In this solution, we use OCL functions integrated in the ATL language. According to our practical experiences all expansion rules may not be implemented using only the ATL language [8]. In that case, we use JMI interfaces and abstract classes which implement JMI interfaces. These interfaces must be implemented for both the DL metamodel and the Tableau metamodel. Combination of ATL and Java is the best way to implement all reasoning services, as some reasoning rules are easier for implementation in Java than in ATL.

Reasoning process is done during transformation from a DL model to a tableau model. Example 2, demonstrates a transformation a DL model of two atomic concepts into a tableau model. Fig. 9 represents a part of ATL code for the transformation.

Example 2. Let be Professor and Student are two atomic concepts. Intersection of two concepts is denoted by standard DLs notations as (Professor ⊓ Student).□

If we want to check satisfiability of the concept, according to [7], we must create an instance of the concept called "x". Applying the rule for intersection of two concepts, we can create a set of constraints as follows: L(x) = {Professor, Student, Professor ⊓ Student}.

There are no rules which can be applied to the constraint system and we can conclude that above concept is satisfiable.

The DL model (in the term of MDA) of the concept (Professor ⊓ Student) is described in Fig. 10, in the Ecore-based (http://www.eclipse.org/modeling/emft/) editor, and conforms to OMG's DLs metamodel [13]. Its XML representation is described at Fig. 11. In this example, we describe how to apply implemented transformation to the DL model.

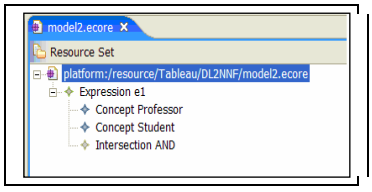


Fig. 10. Ecore-based model of the concept (Professor ⊓ Student)

Fig. 11. XML representation of the concept (Professor ⊓ Student)

Using the implemented transformations, a tableau model for the concept is generated as shown on Fig. 12. The XML representation of the model is presented at Fig. 13.

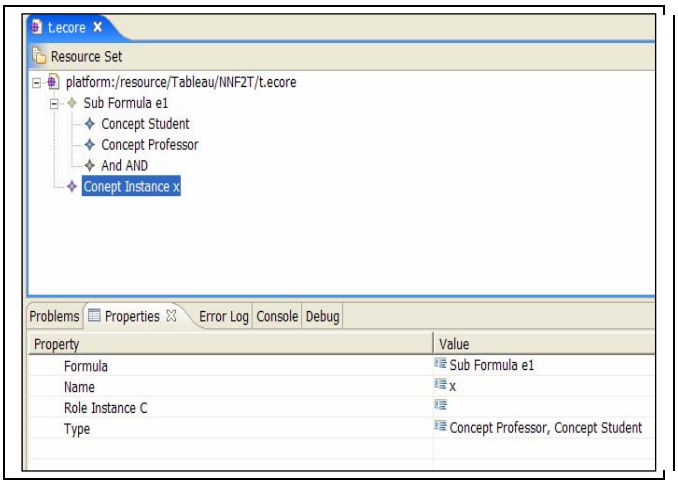


Fig 12. The Tableau model of the concept (Professor ⊓ Student)

Example 3: Suppose that we defined three concepts A, B and C. We want to check satisfiability of next concept: (A ⊓ ¬A) ⊓ (B ⊓ C).

Fig. 14 represents tableau model for concept defined in example 3. This concept is not satisfiable. Model shown on fig. 14 represents unsatisfiable points colored with blue. Also, as in previous example, one variable “x” is generated during this transformation.

Constraint system as beginning point in reasoning processes can be presented as finite set of classes (models), as follows:

L(x) = { A ⊓¬A ⊓ (B ⊓ C) }	(1)

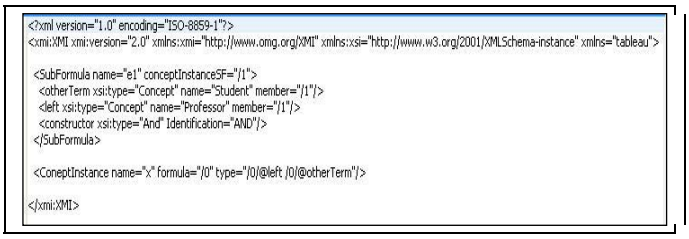


Fig. 13. XML representation of the tableau model for the concept (Professor ⊓ Student)

Individual “x” (See Figure 14 above) is an instance of all subconcepts in this set. Using reasoning rules (in this case it is intersection rule for ALC logic) [7], this constraint system, described by formula (4), can be extended in new ones:

$L(x) = \{ A, \neg A\} \cup L(x)$	(2)

Constraint system consists of clash and formula (3) is not satisfiable, which implies that question subsumes answer and student give true answer. Unsatisfiable points of the beginning model is represented by blue color on Figure 14.

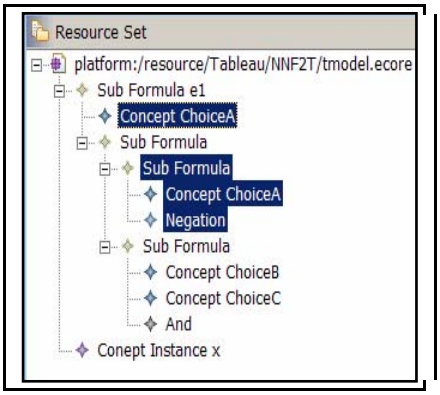


Fig. 14. Tableau model for concept defined in example 3

Example 3 will help us to explain application of the reasoner in intelligent analysis of students' solutions. It means using the reasoner in checking satisfiability of students' answers in case of multiple choice.

If system offers three options on a question, marked with A , B and C. If student checked answer A, reasoner will generate tableau model (shown on fig. 14) for student's answer with respect to given answers. For given example 3, reasoner checks weather student's answer(A) is subsumed by given (possible) answers (A, B, C). Implemented system can not solve more complex unsatisfiable problems then example 3.

6. Related work

Some existing reasoners for description logics [6], [7], [15] are de facto standards in the world of DLs reasoners. Their authors implemented all of the known DLs reasoning services, while some of them have the support for well-known ontology tools. For example, RACER [6] has support for Protégé (http://protege.stanford.edu/) that is the leading ontology editor.

Besides all advantages that all of these reasoners have, they still have some disadvantages that we have tried to address by applying MDE techniques to develop a DL reasoner. We list them below:

1. The present reasoners are implemented in programming languages without explicitly defined model that can be easily extended with new functionalities and re-implementation for new platforms.
   It is important in the case of an integrated model of a reasoner in a more complex model of the software that needs some specific DL reasoning services or the use of a specific DL type.
2. The classical reasoners may not be extended by software developers and may not fulfill their requirements.
   The current reasoners can not be integrated as plug-in architecture in some software development platform such as Eclipse. The importance of such integration is checking consistency of UML models during software development. For example, Eclipse supports a plug-in for Together.[2]
   Our reasoner can be extended using abstract classes that implement JMI interfaces. The user of our reasoner can extend functionality of the reasoner according to their requirements, but without changing core of the reasoner. Our reasoner is plug-in architecture to any software that supports such ability.
3. The present reasoners use data structures of classical programming languages to represent the tableau.
   In our solution we use the XMI format for saving tableau model. It is more reliable to search and update the tableau model using JMI interfaces or ECore classes than using data structures of programming languages.
4. Some current [15] reasoners depend on language specific parsers, e.g., Jena (http://jena.sourceforge.net/) for OWL in implementation reasoner functionality.
   For search, update throughout DL models and tableau models we use JMI interfaces and ECore classes, while all of them can be imported into the repository by using XMI and the ATL's joint framework for managing model metadata (i.e., AMMA) [26].
5. Reasoners like PELLET [15] and FACT [7] are the YES/NO sort of software. Their reasoning algorithm generates only Yes or No answers when checking the consistency of ontology. So, for example, it is difficult to use them, to analyze the semantics of users' answers in intelligent question/answer systems (e.g., in intelligent educational systems) [9].

2 http://www.borland.com/us/products/together/index.html

Using JMI interfaces and ECore classes, generated from the Tableau metamodel, the tableau model may be analyzed to find useful information about the users' answers in cases when the users give wrong answers.

Besides the present DL reasoners, the recent OMG's ODM [13] initiative is very relevant to the proposed model-driven implementation of the DL reasoner. Although, the reasoner is based on the DL metamodel that is non-normative part of the ODM specification, still the reasoner can be used for the future development of the model-driven reasoner for OWL and RDF(S) languages. The core metamodels of the ODM specification, namely the OWL and RDF(s) metamodels, can easily be transformed into the DL metamodel by using the QVT-like language, and thus our Tableau metamodel can be used as a basis for the implementation of reasoning services of the OWL and RDF(s) languages. Furthermore, the algorithm can be extended to support reasoning over UML models as well.

Considering the AIR framework for developing intelligent systems using MDE concepts [5], the proposed model-driven engineering of the tableau algorithm has a full compatibility with AIR. First, they are both based on the same model repository, i.e., JMI-based NetBeans MDR (http://mdr.netbeans.org/). Second, as the AIR workbench is based on the Eclipse plug-in architecture, the used tools (e.g., ATL) for developing the tableau algorithm are also compatible with the AIR workbench. Since we have developed an AIR plug-in for the ODM [5], the proposed tableau algorithm can be used to reason over ODM-based ontologies developed in AIR.

7. Conclusion and future work

Applying the MDE principles, we have proposed the development of a DL reasoner-that is to say, the tableau algorithm that is used for the computation of the satisfiability reasoning service on which all other reasoning services rely. Employing the DL metamodel of the OMG's ODM specification we have shown the practical value of that metamodel for implementing DL reasoners, so that metamodel is a non-normative part of the specification. The definition of the Tableau metamodel for the tableau algorithm enabled the implementation of the satisfiability reasoning service, and thus the tableau algorithm, to be done only by a model transformation from the DL metamodel into the Tableau metamodel. Besides achieving the expected goals to have a DL reasoner (e.g. to be efficiently extended or retargeted to a new platform), the use of metamodelling and model repository provided us with an additional advantage over the current DL reasoners. In fact, the current DL reasoners can only give "yes" or "no" as an answer when checking consistency of a DL model, while our approach can also detect the cause of an inconsistency.

We are currently working on the implementation of other DL reasoning services that are based the tableau algorithm presented in this paper. We are also developing a specialized GUI for the reasoner that has a plug-in architecture. We will also further extend the reasoner, so that it can support

DL typeson which the standard Semantic Web ontology languages rely (i.e., OWL) (http://www.w3.org/2007/OWL/wiki/OWL_Working_Group). We plan to develop QVT transformations from the OWL ODM metamodel [13] into the DL metamodel in order to provide a reasoner for the OMG's ODM specification [13]. Finally, we plan to test the developed algorithm in practical Semantic Web applications such as intelligent educational systems, for example, to analyze the semantics of the students' answers.

8. References

1. C. Atkinson, T. Kuhne, "Model-Driven Development, A Metamodelling Foundation", *IEEE Software,* vol.20, No. 5, 2003, pp. 36-41.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi , P. Patel-Schneider, *The Description Logic Handbook-Theory, Implementation and Application*, Cambridge University Press, 2003.
3. J. Bezivin, "In Search of a Basic Principle for Model-Driven Engineering," *Upgrade,* vol. 5, No. 2, 2004, pp. 21-24.
4. R. Dirckze, (spec. leader): "Java Metadata Interface (JMI) API Specification ver. 1.0", 2002 [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr040/
5. D. Djuric, D. Gasevic, V. Damjanovic, "AIR-A Platform for Intelligent Systems", *In Proceedings of 1st IFIP International Conference on Artificial Intelligence Applications and Innovations*, Toulouse France, 2004, pp. 383-392.
6. V. Haarslev, R. Moller, RACER System Description, *In Proceedings of the International Joint Conference on Automated Reasoning*, Siena, Italy, 2001, pp. 701–705.
7. I. Horrocks, *Optimising Tableaux Decision Procedures for Description Logic*s, PhD Thesis, University of Manchester, 1997.
8. F. Jouault, I. Kurtev, "Transforming Models with ATL", In Proceedings of the Model Transformations in Practice Workshop at MoDELS, Montego Bay, Jamaica, 2005.
9. N. Krdzavac, D. Gasevic, V. Devedzic, "Description Logic Reasoning in Web-based Education Environment", *In Proceedings of the Workshop on Adaptive Hypermedia and Collaborative Web-based Systems (4th International Conference on Web Engineering)*, Munich, Germany, 2004.
10. Meta Object Facility (MOF) Specification, v1.4, [Online]. Available: http://www.omg.org/docs/formal/02-04-03.pdf
11. J. Mukerji,J. Miler, "MDA Guide Version. 1.0.1", [Online]. Available: http://www.omg.org/docs/omg/03-06-01.pdf
12. "OMG XMI Specification, ver. 1.2", OMG Document Formal/02-01-01, 2002. [Online.] Available: http://www.omg.org/cgi-bin/doc?formal/2002-01-01.pdf
13. "Ontology Definition Metamodel", Preliminary Revised Submission to OMG RFP ad/2003-03-40 1, 2004. [Online]. Available: http://codip.grci.com/odm/draft
14. "Request for Proposal: MOF 2.0 Query / Views /Transformations RFP", OMG Document: ad/2002-04-10 (2002) [Online]. Available: http://www.omg.org/docs/ad/02-04-10.pdf
15. E. Sirin,B. Parsia, "An OWL DL Reasoner", *In Proceedings on International Workshop on Description Logics (DL2004)*, Whistler, BC, Canada, June 2004.

16. R. Soley, “MDA, An Introduction”, 2004. [Online]. Available: http://www.omg.org
17. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable of Ebject-Oriented Software*, Addison-Wesley, Reading, MA, 1995, ISBN: 0201633612
18. F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg Siekmann on the Occasion of His 60th Birthday*, LNAI 2605, Springer, Berlin, 2005, pp. 228-248
19. C. Lutz: The Complexity of Description Logics with Concrete Domains. PhD thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, (2002).
20. E. Seidewitz, “What Models Mean,” *IEEE Software*, vol. 20, no. 5, 2003, pp. 26-32.
21. B. Selic, “The Pragmatics of Model-Driven Development,” *IEEE Software*, vol. 20, no. 5, 2003, pp. 19-25.
22. NetBeans Metadata Repository (MDR), http://mdr.netbeans.org, 2003.
23. Poseidon for UML, http://www.gentleware.com, 2006.
24. “Ontology Definition Metamodel,” OMG Document ad/06-05-01, http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf, 2006.
25. Berners-Lee, T., Hendler, J., Lassila, O. “The Semantic Web,” *Scientific American*, vol. 284, no. 5, 2001, pp. 34-43.
26. Bézivin, J., Jouault, F., Rosenthal, F., and Valduriez, P. (2005), “The AMMA platform support for modeling in the large and modelling in the small,” *LINA Technical Report No. 04.09*, University of Nantes, France, 2005.
27. Eclipse Modeling Framework, http://www.eclipse.org/emf, 2006.
28. N. Krdžavac, V. Devedžić, “A Tableau Metamodel for Description Logics”, *In Proceedings of Workshop on Automated Reasoning Bridging the Gap between Theory and Practice*, University of Bristol, UK, April 2006.

Nenad Krdzavac is currently PhD student at the FON-School of Business Administration, University of Belgrade, Serbia. So far, he has authored several research papers. He is a member of the GOOD OLD AI research group. His research interests are Description Logics

Dragan Gasević is an Assistant Professor in the School of Computing and Information Systems at Athabasca University and an Adjunct Professor at Simon Fraser University. He is a recipient of Alberta Ingenuity's 2008 New Faculty Award. His research interests include semantic technologies, software language engineering, and learning technologies. He has (co-)authored around 190 research papers published. He has been serving on editorial boards of three international journals and has edited special issues in journals such as IET Software and IEEE TSE. He has been the organizer, chair, and member of program committees of many international conferences.

Vladan Devedzić is a Professor and Head of the Department of Software Engineering at FON-School of Business Administration, University of Belgrade, Serbia. His research interests focus on the practical engineering aspects of developing intelligent educational systems for the Web, while his

long-term goal is to merge ideas from intelligent systems and software engineering. He has authored/co-authored more than 280 research papers, published in international and national journals or presented at international and national conferences, as well as six books on intelligent systems and software engineering.

Received: December 05, 2006; Accepted: August 05, 2008.