# MDA-Based Architecture of a Description Logics Reasoner

Nenad Krdžavac[1], Dragan Đurić[2], Vladan Devedžić[2]
1Faculty of Electrical Engineering, University of Belgrade,
Bulevar Kralja Aleksandra 73, 11 000 Belgrade, Serbia and Montenegro
nenadkr@galeb.etf.bg.ac.yu
2FON - School of Business Administration,
Dept. of Information Systems and Technologies, Univ. of Belgrade
POB 52 Jove Ilica 154, 11000 Belgrade, Serbia and Montenegro
dragandj@gmail.com, devedzic@fon.bg.ac.yu

**Abstract.** The paper presents the architecture and design of a description logics (DLs) reasoner based on the Model Driven Architecture (MDA) methodology. The architecture relies on OMG's description logics metamodel, tableau metamodel, and model transformations using a language for model transformation. We show an example of DLs knowledge base using UML notation in context of MDA. The idea can be easily applied to implementation of a first-order logic theorem prover. The paper does not discuss implementation details of reasoning algorithms and the reasoner performance.

## 1 Introduction

Almost every software project needs an analysis of the range of problems that the software being developed should solve [13]. One way is to specify and build the system using modeling tools. For example, a modeling tool such as UML supports full development life-cycle of such software: design, implementation, deployment, maintenance, evaluation, and integration with other systems [16]. Model Driven Architecture (MDA) is an approach to IT system specification that separates the specification of functionality from the specification of implementation on a specific technology platform [10].

The basic notations in description logics (DL) are concepts (unary predicate) and roles (binary predicates) [2]. One of the most important constructive properties of DLs is their reasoning services, which can be applied for reasoning with ontologies. Some publicly available implementations of DLs reasoners [7], [6], [15] can reason

Please use the following format when citing this chapter:
Krdzavac, Nenad, Djuric, Dragan, Devedzic, Vladan, 2006, in IFIP International Federation for Information Processing, Volume 204, Artificial Intelligence Applications and Innovations, eds. Maglogiannis, I., Karpouzis, K., Bramer, M., (Boston: Springer), pp. 98–105

with ontologies, but the authors of those reasoners did not implement their reasoners using advanced model engineering techniques (such as MDA) and current software engineering standards.

The goal of this paper is to describe the architecture of a description logics reasoner in an MDA environment, and some benefits of using that methodology to implementant that kind of software. We suggest possible applications of such description logics reasoners as plug-ins to intelligent systems (we exemplify it by the AIR system [5]), or in intelligent analysis of students' solutions in Web-based intelligent tutoring systems (ITS).

In section 2, we describe basic concepts of Model Driven Architecture (MDA) and refer to basic research papers where readers can find useful information about that methodology. Section 3 is the main section in this paper and describes the architecture of a description logic reasoner. That section also explains some benefits of using MDA in the implementation of the reasoner. We show a method for implementing the reasoning algorithms for description logics based on the Atlas Transformation Language (ATL) [8] that is an alternative to OMG's standard [14] for model transformations. Within the scope of section 3, we describe basic concepts of description logics and give example of a DL based knowledge base in DL syntax and in UML notation.

## 2 Model Driven Architecture

Model Driven Architecture (MDA) is defined as a realization of model-engineering principles proposed by Object Management Group (OMG) [11]. According to [3], there are a few central properties of the MDA:

1. Four layer architecture and relationships among them (see Fig. 1).
2. Transformation among the models on the same layers M1 and M2
3. XML-based standard for sharing metadata, called XMI

The top-most layer (M3) is called meta-meta model layer and OMG has defined a standard at this layer as well - MOF (Meta Object Facility). According to [10], MOF is the language intended for defining meta-models at M2 layer.

MOF defines a set of reflective APIs consisting of reflective interfaces. Java Metadata Interfaces (JMI) 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 using Java language.

In terms of MDA, a metamodel makes statements about what can be expressed in the valid models described in a certain modeling language. Examples of a metamodels are UML metamodel (Fig. 1) and OMG's Description Logics metamodel (Fig. 2) [13]. The next layer is the model layer (M1) – the layer where we develop real-world models. In terms of UML, it means classes and relationships among them. MDA layers are called linguistic layers, but the concepts from the same linguistic layer belong to different ontological layers [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. 1).

The object-oriented paradigm uses the terms "*instanceOf*" and "*inherits*" to describe relations between classes and objects. Model engineering uses the terms "*representedBy*" and "*conformantTo*" [3] to establish relations between the models in MDA layers (see Fig. 1). Model-based engineering and object technologies in software development can be viewed as complementary approaches [3].

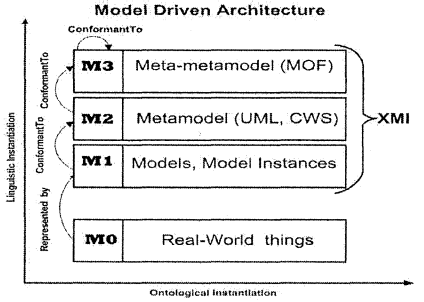


**Fig. 1.** Four layer architecture of MDA (see [3])

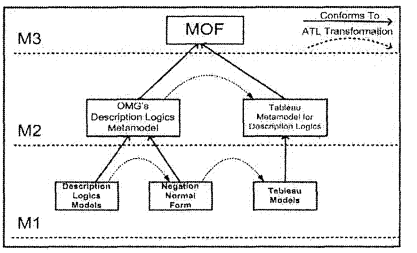


**Fig. 2.** Architecture of a Description Logic Reasoner

## 3 MDA-based Architecture of a Reasoner

Our DL reasoner is based on the DLs metamodel, proposed by OMG consortium (www.omg.org) [13], and on the tableau metamodel (Fig. 2).

Some publicly available DL reasoners are successfully implemented in object-oriented technology [15], or in the LISP programming language [6], [7] for very expressive description logics. The authors of such reasoners did not follow software engineering standards, i.e. they did not describe the models of the reasoners in the standard UML notation. The advantage of using such modeling tools is in supporting the full life cycle of software: design, implementation, deployment, maintenance, evaluation and integration. UML supports MDA concepts of software development. Implementation of a MDA-based reasoner includes a few steps:

1. Building the tableau metamodel for description logics;
2. Implementating the repository for OMG's DLs metamodel and the tableau metamodel;
3. Implementating the reasoning algorithms using model transformation.

The tableau metamodel can be built using UML. The tableau algorithm uses a tree (T) to represent the model being constructed [7], hence the Composite design pattern can be used in describing the tableau metamodel.

### 3.1 UML Model of Description Logics Knowledge Bases

The basic notations in description logics (DL) are concepts (unary predicate) and roles (binary predicates). Specific description logic is mainly characterized by a set of constructors that provides to build more complex concepts (concept expressions) and role expressions [2]. A knowledge base (KB) developed using DL consists of two components, TBox and ABox. Reasoning in description logics (such as satisfiability) is based on tableaux algorithm [7]. A plenty of details in the field of description logics and reasoning can be found at [2].
There are two ways to represent a DL-based knowledge base:

1. Using UML language to describe a model that conforms to DLs metamodel (Fig. 2)
2. Using the definition of the knowledge base and syntax of DLs languages.

Example describes a TBox of family relationships. The first part of the example describes family relationships using DL notations, but the second one describes them in UML and represents a UML model that conforms to DL metamodel proposed by OMG [13].

Example1: Suppose that nouns *Human*, *Male* and *Parent* are atomic concepts and *hasChild* is an atomic role, than means that every Male is Human but not vice versa (Formula 1). Concept definition represents all Fathers that have only male children (Formula 2).

$$\text{Male} \sqsubset \text{Human} \tag{1}$$

$$\text{Father} \equiv \text{Male} \sqcap \forall \text{hasChild.Male} \tag{2}$$

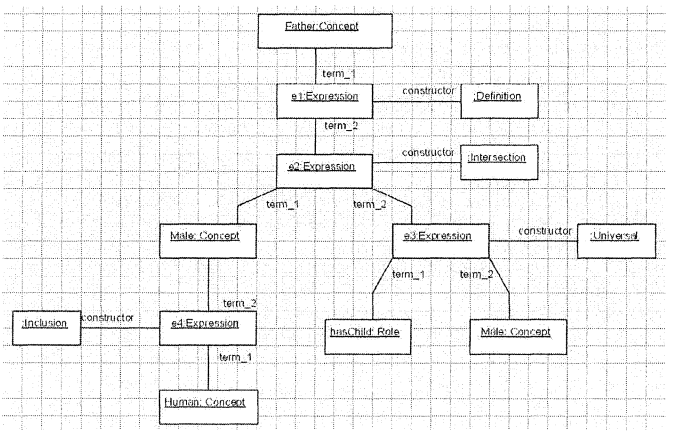


**Fig. 3.** The "Father" model in the MDA technological space [Online]. Available: http://www.gentleware.com/index.php

The same knowledge base can be expressed using UML language but in MDA environment, Fig. 3. The model shown in Fig. 3 belongs to the M1 layer of the 4-layer MDA (Fig. 2 and Section 3) and conforms to the OMG's DL metamodel.

The semantics (Tarski style) of formulas (1), (2), and semantics of DL model (Fig. 2 and Fig. 3) are the same. According to OMG's ODM proposal [13], the concepts Father, Male and Parent, represented in Fig. 3 in UML notation, are instances of the metaclass Concept. Term_1 and Term_2 are association ends in association relation between the metaclasses Term and Expression. Term_1 represents dyadic constructor, but Term_2 represents monadic constructor [13].

### 3.2 Implementation of the Repository for Description Logics and Tableau Metamodels

The repository is used for storing and retrieving the models that conform to both DLs and tableau metamodels. It is built using the Metadata Repository (MDR) for NetBeans project (www.mdr.netbeans.org) and Java metadata interfaces (JMI). According to [4], the JMI specification defines Java mappings for MOF. The repository provides standard services for creating, accessing, updating, deleting and validating metadata. There are a few steps in implementing the repository for both metamodels:

1. Converting the UML metamodels into MOF metamodels;
2. Generating JMI interfaces for both metamodels;
3. Instantiating the models in the repository. Imported models conform to both metamodels (Fig. 2).

Using Poseidon for UML tool (www.gentleware.com), the metamodels can be saved in the XMI format. The **uml2mof.jar** tool (www.mdr.netbenas.org) can be used to generate MOF metamodels.

In spite of all the advantages of OMG's DL metamodel [13] and the defined tableau metamodel, there are also several practical problems related to the implementation of the repository for both metamodels:

1. Association ends of the composition relation between the metaclasses *Assertion* and *Instance* [13] have the same name. When we generate JMI interfaces in the association proxy interface we find objects that for different metaclasses have the same name, so we have to change them manually.
2. Association ends between the metaclasses *Term* and *Expression* [13] do not have names, so we have to name them, as we could not generate JMI interfaces.
3. The DL meta-model cannot support a very important class of DLs called description logics with concrete domain [2], as it does not have metaclasses defined in their definitions of both syntax and semantics. Accordingly, the meta-model would not be able to support reasoning with a knowledge base in such logics.
4. During the generation of JMI interfaces all the OCL constraints were ignored and we had to implement the constraints manually.

There are a few advantages of using the MDA methodology to implement the reasoner, wrt. classical object-oriented programming or LISP programming language:

1. Automatic transformation of the model-driven system, from higher-level abstract models to object-oriented models, to a running system.

The metamodels are built using UML. MDR NetBeans Explorer (www.mdr.netbeans.org) enables generating JMI interfaces according to JSR040 [4] standard. These interfaces allow for handling the models at the M1 level (Fig. 2). It cannot be done using object-oriented methodology or LISP programming.

2. Some publicly available reasoners [15] implemented in object-oriented technology used ontology parsers. In our implementation, parsers are not used.
3. Reasoning algorithms, such as checking for consistency, are implemented using data structures like trees or hash tables [15]. In this solution, tableau can be represented using XMI.

Tableau model can be described using XML Metadata Interchange. XMI has a tree structure - every tableau model is a tree [2]. Because of this similarity, it is easy to describe a tableau model (which conforms to the tableau metamodel) using XMI. JMI interfaces generated for tableau metamodel are used for creating, accessing, updating, deleting and validating tableau models according to JSR040 standard [4].

4. Existing publicly available reasoners [7], [6] may not be integrated in today's intelligent metamodeling frameworks, like AIR [5] or intelligent web-based education systems as plug-ins, especially if such a reasoner should be plugged into the Eclipse Modeling Framework (EMF) (www.eclipse.org).
5. Suitability for making further extensions of the reasoner.

Reasoners like PELLET [15] or FACT [7] are YES/NO sort of software. Their reasoning algorithm for consistency answers only Yes or No when checking the consistency of ontology. It is difficult to use them that way in, e.g., intelligent analysis of the semantics of students' solutions in intelligent web-based educational systems [9]. Using JMI interfaces, generated from the tableau metamodel, the tableau model may be analyzed to find useful information about the students' solutions in cases when the students give wrong answers.

### 3.3 Implementation of the Reasoning Algorithms

A transformation from one model to another is the key technology in the MDA paradigm [3]. OMG proposed a standard [14] for model transformation. Three vital subjects of the proposal that ensure the full realization of MDA are:

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

The Atlas Transformation Language (ATL) [8] is an answer to the **OMG's QVT RFP** [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 Eclipse Modeling Framework (EMF) (www.eclipse.org/gmt) as a plug-in.

ATL is a declarative and hybrid language. 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 the language are described in [8].

3. During the generation of JMI interfaces, all OCL constraints were ignored. We have to implement them manually.

The ATL language is implemented with respect to the OCL standard.

4. ATL is compatible with JMI interfaces.

ATL transformation model is first read using the ATL parser and loaded into Java meta-data repository which is based on a JMI compliant repository. The generated JMI interfaces for both metamodels 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 description logics. Although mainly intended to deal with MDA models (based on MOF meta-models and accessible via XMI or JMI), the EMF framework 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 reasoner in other platforms like intelligent metamodeling frameworks, especially in case of using such a reasoning machine to reason on UML models (not UML diagrams).

The first step in implementing the reasoning algorithms is bridging the DL metamodel and the tableau metamodel at the M2 level (Fig. 2), using the ATL language. At the M1 level (Fig. 2), the DL model in negation normal form must be transformed into the tableau model according to the reasoning expansion rules [7].

## 4 Conclusion and Future Works

The paper proposed the architecture of a description logic reasoner based on the OMG's DL metamodel. We also proposed a method for implementation of reasoning algorithms for description logics using the ATL language. Such an implementation methodology for the reasoner is flexible in practical uses of the reasoner in today's intelligent metamodeling framework or web-based intelligent tutoring systems. In the future we will try to test our reasoner in various Semantic Web applications.

## References


1. Atkinson C., Kuhne T., Model-Driven Development, A Metamodeling Foundation, *IEEE Software* **20** 5 (2003) 36-41
2. Baader F., Calvanese D., D. McGuinness, Nardi D., Patel-Schneider P., The Description Logic Handbook-Theory, Implementation and Application, Cambridge University Press (2003)
3. Bezivin J., In Search of Basic Principles for Model Drive Architecture, *The European Journal for The Informatics Professional,* **5** 2 (2004)
4. Dirckze R., (spec. leader): Java Metadata Interface (JMI) API Specification ver. 1.0 (2002) [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr040/
5. Djuric D., Gasevic D., Damjanovic V., AIR-A Platform for Intelligent Systems, In Proceedings of AIAI 2004: First IFIP International Conference on Artificial Intelligence Applications and Innovations, Toulouse France (2004)
6. Haarslev V., Moller R., RACER Systems Description, *Lecture Notice in Computer Science* **20083** (2001)
7. Horrocks I., Optimising Tableaux Decision Procedures for Description Logics, PhD Thesis, University of Manchester (1997)
8. Jouault F., Kurtev I., Transforming Models with ATL, In Proc. of the Model Transformations in Practice Workshop at MoDELS, Jamaica (2005). Available: http://sosym.dcs.kcl.ac.uk/events/mtip/submissions/jouault_kurtev__transforming_models_with_atl.pdf
9. Krdzavac N., Gasevic D., Devedzic V., 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. Mukerji J., Miler J., 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. Sirin E., Parsia B., An OWL DL Reasoner, Proceedings on International Workshop on Description Logics (DL2004), British Columbia, Canada 6. - 8. June (2004)
16. Soley R., MDA, An Introduction, [Online]. Available: http://www.omg.org, (2004)