*Zbornik radova 49. Konferencije za ETRAN, Budva, 5-10. juna 2005, tom III*
*Proc. 49th ETRAN Conference, Budva, June 5-10, 2005, Vol. III*

# A MDA-BASED DESCRIPTION LOGIC REASONER

Nenad Krdžavac, *Faculty of Electrical Engineering, University of Belgrade*
nenadk@galeb.etf.bg.ac.yu , www.goodoldai.org.yu
Dragan Gašević, *FON- School of Business Administration, University of Belgrade*
gasevic@yahoo.com, www.goodoldai.org.yu

**Abstract** – *This paper describes implementation details of an Attribute Language with Complement (ALC) description logic reasoner based on a model-engineering technology, called Model Driven Architecture (MDA). Some publicly available reasoners are successfully implemented in object-oriented technology or in LISP programming language, but reasoners do not adore state-of-the-art software engineering standards and their authors did not describe models of the reasoners in a standard (i.e. UML) notation.*

## 1. INTRODUCTION

Almost every software project needs an analysis of the range of problems that the software being developed should solve [18]. One way is to specify and build the system based on modeling techniques with e.g. Unified Model Language (UML) as it supports full life development cycle of such software: design, implementation, deployment, maintenance, evaluation, and integration with later systems [22]. Model Driven Architecture (MDA) is an approach that separates the specification of functionality from the specification of implementation on a specific technology platform [17]. According to MDA concepts, systems are developed via transformation of models. MDA defines two basic kinds of models: Platform Independent Model (PIM) and Platform Specific Model (PSM). Developers start by creating a PIM and transform the model step–by-step into a more PSM model [11]. A possible application domain for using MDA standards are knowledge representation languages like languages for developing ontologies. For example, there are some efforts to develop MDA-based ontology languages, namely Ontology Definition Metamodel (ODM) and Ontology UML Profile (OUP) [9].

In fact, DLs are the most recent name for a family of knowledge representation formalisms that represents the knowledge of the application domain (the "world") by defining the most relevant concepts of the domain (i.e. its terminology) and by using these concepts to specify properties of objects and individuals occurring in the domain (i.e. the world description) [2]. One of the most important constructive properties of DLs is reasoning services, which can be applied as reasoning with ontologies. Some implemented DLs reasoners [13], [10], [21], publicly available, can reason with ontologies, but the authors of those reasoners did not implement such reasoners by using MDA engineering techniques and the reasoners do not adore these software standards. The goal of this paper is to survey implementation details of a description logic reasoner based on DLs meta-model. In section 2 we desribe basic concepts of ALC description logic and describe basic reasoning service with DLs. Section 3 describes MDA for ODM. Section 4 outlines implementation details of the MDA-based (ALC) reasoner. In this section we also present some practical aspects of such an implementation and give some practical disadvantages of the proposed OMG' DL meta-model.

## 2. DESCRIPTION LOGIC PROPERTIES: A BRIEF SURVEY

Historically, DLs evolved from semantic networks and frame systems, mainly to satisfy the need of giving a formal semantics to these formalisms [14]. The basic notions in DLs are *concepts* (unary predicates) and *roles* (binary relations). A specific DL is mainly characterized by a set of *constructors,* it provides to build more complex concepts (concept expressions) and roles out of atomic ones. According to [14], syntax and semantic of the logic (with example) is shown in next definitions.

**Definition 1:** (*ALC syntax*)

Let $N_C$ and $N_R$ be disjoint and countably infinite sets 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 $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$ are ALC –concepts ◇

We use $\top$ as an abbreviation for some fixed propositional tautology such as $(A \sqcup \neg A)$, whereas $\bot$ as a concept $(A \sqcap \neg A)$. The meaning of the concepts is fixed by Tarski-style semantics [14].

**Definition 2:** (*ALC semantics*)

An ALC–interpretation *I* is a pair $(\Delta_I, \cdot^I)$ where $\Delta^I$ is non–empty set called *domain*, and $\cdot^I$ is an *interpretation function* that maps every concept name A to a subset $A^I$ of $\Delta_I$ and every role name R to a binary relation $R^I$ over $\Delta_I$.

The interpretation function is extended to complex concepts as follows:

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

**Example 1**: Suppose that nouns *Human* and *Male* are concept names and *hasChild* is the role name, then ALC concept $(Human \sqcap \exists hasChild.\top)$ represents all persons that have a child while concept ($\text{Human} \sqcap \forall \text{hasChild.Male}$) represents all persons that have only male children◇

Standard reasoning services offered by DLs are [14]:

1. Decide whether a concept C is satisfiable, i.e. whether it can have any instances (*concept satisfiability*),
2. Decide whether a concept C is subsumed by a concept D, i.e. every instance of C is necessarily also an instance of D (*concept subsumption*), and
3. Decide whether a given ABox is consistent, i.e. whether a world as described by the ABox may exist (*ABox consistency*).

These reasoning problems are called "standard" reasoning tasks, but a non-standard reasoning service is, for example, unification of two concept patterns [14]. Concept subsumption can be transposed into an equivalent satisfiability problem [12], and validity of this transformation is clear from the semantics of subsumption. Satisfiability problem can be solved using algorithm based on tableaux calculus [12], [14].

Tableaux algorithms try to prove the satisfiability of a concept expression D, by demonstrating a model – an interpretation $I = (\Delta^I, \cdot^I)$ in which $D^I \neq \varnothing$ [12]. In general, tableaux algorithms decide whether a concept is satisfiable by trying to construct a model for it [16]. A tableau is a graph which represents such a model, with nodes corresponding to individuals and edges corresponding to relationships between individuals. According to [12], tableaux algorithm is guaranteed to terminate.

A knowledge base (KB) in DLs comprises two components, TBox and ABox [2]. The terminology or is called TBox. On the other hand, the name of assertions is ABox that represents named individuals expressed in terms of the vocabulary [18]. The definition of TBox and ABox is given in [2], [14]. Reasoning allows one to infer implicitly represented knowledge from such knowledge that is explicitly contained in the knowledge base [2].

## 3. MDA FOR ODM

The basic principle "*Everything is object*" was important in the 80's to set up object-oriented technologies [4]. Beside object-oriented technology known is model-engineering concept which basic principle is "*Everything is model*". Engineering models aim to reduce risk by helping us better understand both a complex problem and its potential solutions before undertaking the expense and effort of a full implementation [25]. MDA is defined as a realization of model-engineering principles around a set of OMG standards [4]. The central part of MDA is the four-layer architecture that has a number of standards defined at each of its layers (see Fig. 1). Most of MDA standards are developed as meta-models using meta-modeling. The top-most layer (M3) is called meta-meta-model and the OMG's standard defined at this layer is Meta-Object Facility (MOF). MOF is OMG's standard closely related to UML that enables metadata management and language definition [17]. According to [28], MOF is standard that specifies an abstract language for describing other languages. Actually, MDA layers are called linguistic layers. On the other hand, concepts from the same linguistic layer can be at different ontological layers [1]. The main advantages of the use of MDA concepts for implementation of our reasoner are the following:

1. Suitability for making further extension of the reasoner
2. Less mistakes during implementation as we use tools for transformation from model to code (in our case JMI interfaces)
3. Our future code (extension of the reasoner) can be integrated, easily, in such a software production environment.

The MDA's meta-model layer is usually denoted as M2 (Fig. 1). At this layer we can define a new meta-model (DL meta-model). The next layer is the model layer (M1) – the layer where we develop real-world models (or domain models). In terms of UML models, that means creating classes, their relations, states, etc. [9]. The bottom layer is the instance layer (M0). According to [9], there are two different approaches to explain this layer:

1. The instance layer contains instances of the concepts defined at a model layer (M1), e.g objects in some programming languages.
2. The instance layer contains things from our reality – concrete (e.g. Peter is instance of the class Professor, similar as individuals in description logics) and abstract (e.g. UML classes – Student, Persons etc). It is similar as concepts in description logics, but classes in UML have behavioral component).

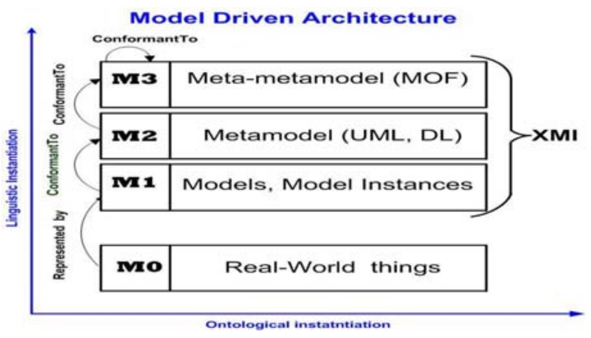

*Fig.1.. The four layer MDA and its ontological instances of relations-linguistics and ontological*

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) [19]. In MDA technological space is defined ODM at M2 layer (Fig.2). ODM in its structure includes several meta-models [18], proposed by OMG. The core of this architecture is DL meta-model. To be useful and effective DLs meta-model must have [25] a few characteristics such as: abstraction, understandability, accuracy, predictiveness, and inexpensive.

JMI specification defines dynamic, platform neutral infrastructure that enables the creation, storage, access, discovery, and exchange of metadata [6]. JMI also specifies the Java programming interface for manipulating MOF-based models and meta-models. Furthermore, JMI enables generation of programming interfaces based on such models

[9]. To generate JMI interfaces from the meta-model we used OMG' XML Metadata Interchange standard (XMI), which provides a mapping form MOF to XML [6]. In object oriented-paradigm we can not use benefits of such a meta-modeling approach.

## 4. IMPLEMENTATION DETAILS

The OMG DL meta-model is described in UML language (M2 layer, Fig. 1). According to the OMG API specification for ODM meta-models [18], the DLs meta-model has three packages shown in Fig. 3. The packages are generated in to Java implementation packages. The MOF reflective packages contains eight abstract interfaces that are extended by the generated interfaces [6]. By using some of present MDA-repositories we have a feature to produce JMI compliant code from the DLs meta-model.

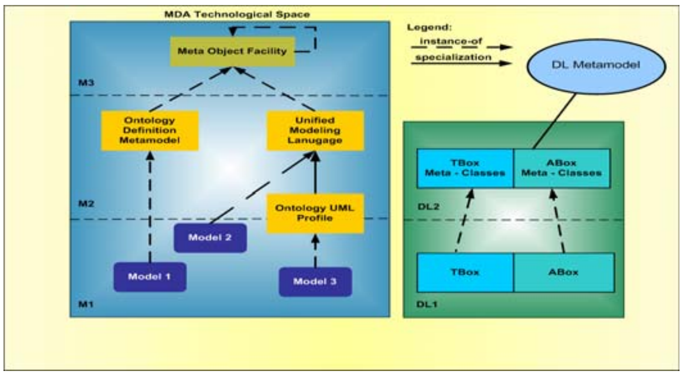

*Fig.2. Description logics meta-model in MDA technological space*

For the DL meta-model we describe two generated interfaces:

1. Instance interfaces

Instance interfaces contain methods for accessing instance-level attributes and references and invoking instance-level operations [6]. The JMI instance interface, which corresponds to meta-class Term [18], does not contain such methods and extends RefObejct abstract interface. Here is a part of the generated Java code for the metaclass Term:

```
package org.omg.odm.dl;
public interface Term extends javax.jmi.reflect.RefObject{
}
```

2. Class proxy interfaces

The interface ClassProxy extends *RefClass* and contains operations for creating instances of the corresponding metaclass. The implementation of the interface is in the package **org.omg.odm.dl.impl** and we add suffix "*Impl*" to all the implemented interfaces in that package.

```
package org.omg.odm.dl;
public interface ConceptClass extends javax.jmi.reflect.RefClass {
   public Concept createConcept();
   public Concept createConcept(java.lang.String uniqueidentifier);
}
```

For implementation of reasoning algorithms we use language called The Constraint Handling Rules (CHR) [23]. Our motivation is benefits of the language in implementation terminological reasoning**,** described in [27]. Inference rules for ALC logic [12], can be directly written in the language [24]. To integrate these rules in our MDA, we use The Java Constraint Kit (JCK) [24]. This is a package which contains a generic search engine to solve constraint problem, a high level language to write applications specific constraint solvers. Practical and theoretical aspects of Constraint Handling Rules are given in [23].

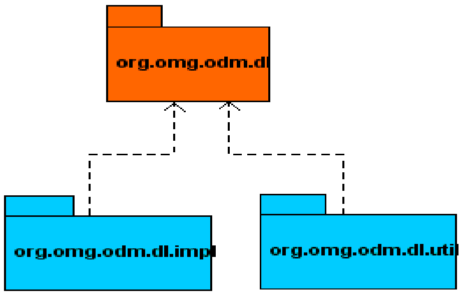

*Fig. 3. Packages for generated description logic reasoner*

Beside all advantages of the meta-model-based approach to the implementation DL reasoner, we have faced the some shortcomings in the DL meta-model during the generation of JMI interfaces. Our motivation to describe them is to provide readers of the papers with some useful information about practical problems in implementation any software in model-engineering paradigm using current tool support. The first tool we used is Poseidon for UML to save the DL meta-model in an XMI file. With the tool *uml2mof.jar* we generated a MOF XMI file. Using Java NetBeans 3.5.1 we generated the JMI interfaces. Here are some experiences with using the DL meta-model:

1. Association ends of the composition relation between metaclasses Assertion and Instance have the same name. When we generate JMI interfaces in the association proxy interface we found objects of different metaclasses that have the same name, so we had to change them manually.
2. Association ends between the metaclasses Term and Expression did not have any name, so we had to named them, because we could not generate JMI interfaces according to JSR-40 [6].
3. The DL meta-model cannot support a very important class of DLs called description logics with concrete domain and functional dependences.
4. During the generation of JMI interfaces all the OCL constraints were ignored and we had to implement the constraints manually.

## 5. CONCLUSION AND FUTURE WORK

In this paper we analyzed implementation details of ALC description logic reasoner using MDA concept.

We hope that readers of this paper will find useful information how to apply MDA approaches in implementation such sort of software especially theorem provers. Our future work will be focused on extensions of the reasoner to support ontology, especially OWL ontologies and implementation tableaux algorithms for description logic with concrete domains and functional dependencies, in model-engineering paradigm. We will develop a graphical user interface for the reasoner.

## LITERATURA

[1] C. Atkinson, T. Kühne, Model-Driven Development: A Metamodeling *Foundation, IEEE Software*, Vol. 20, No. 5, pp. 36-41, 2003.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, The Description Logics Handbook-Theory, Implementation and Application, *Cambridge University Press*, 2003.

[3] D. Berardi, A. Calĺ, D. Calvanese, G. Ge Diacomo, Reasoning on UML Class Diagrams, *Technical Report 11-03,* Dipartimento di Informatica e Sistemistica, Universitá di Roma, »La Sapienza [Online].Avaliable: http://www.inf.unibz.it/~calvanese/teaching/03-ESSLLI/ , 2003.

[4] J. Bézivin, In Search of a Basic Principle for Model Driven Architecture, *The European Journal for The Informatics Professional,* Vol. V, No. 2, April 2004.

[5] J. Bézivin, F. Jouault, P. Valduriez, On the Need for Megamodels, *In Proc. of the Int. Workshop Best Practices for Model Driven Software Development*, Vancouver, Canada, 2004.

[6] R, Dirckze, (spec. leader) Java Metadata Interface (JMI) Specification Version 1.0, [Online].Available: http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html, 2002.

*[7]* D. Djurić, et al: A MDA-based Approach to the Ontology Definition Metamodel, *In Proc. of the 4th Int. Workshop on Comp. Intel. and Inf. Tech.*, Niš, Serbia and Montenegro, 2003, pp. 51-54.

[8] F.M. Donini, M. Lanzerini, D. Nardi, A. Schaerf, Reasoning in Description Logics, In Gerhard Brewka editor, *Fundation of Knowledge Representation, , CSLI-Publication,* 1996, pp.191-236.

***[9]*** D. Gašević, et al: "Approaching MDA and OWL Ontologies Through Technological Spaces," *In Proc. of the 3rd Int. Workshop on Software-Model Engineering*, Lisbon, Portugal, 2004.

[10] V. Haarslev and R. Moller, Description of the RACER System and its Applications, *Proceedings International Workshop on Description Logics (DL-2001)*, Stanford, USA, 2001.

[11] P. Hnětynka, M. Píše, Hand-written vs. MOF-based Metadata Repositories: The SOFA Experience, *In Proc. of the 11th IEEE int. Conf. and Workshop on the Engineering of Computer-Based Systems (ECBS'04*), Brno, Czech Republic, 2004.

[12] I. Horrocks, Optimising Tableaux Decision Procedures for Description Logics, *PhD Thesis*, Univesity of Manchester, 1997.

[13] I. Horrocks, The FACT System, In. H. de Swart, editor, Automated Reasoning with Analytic Tableaux and Related Methods, *Int. Conf. Tableaux'98, number 1397 in Lecture Notes in Artificial Intelligence*, Springer-Verlag, May 1998, pp.307-312.

[14] C. Lutz, The Complexity of Description Logics with Concrete Domains, *PhD thesis*, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2002.

[15] J. Miller and J. Mukerji, (eds), MDA Guide Version 1.01, *OMG Document Number: omg/2003-06-01*, [Online].Available from: http://www.omg.org/docs/omg/03-06-01.pdf , 2003.

[16] M. Miličić, Description Logics with Concrete Domains and Functional Dependencies, *Master's Thesis*, Technical Univesity Dresden, Dresden, May 2004.

[17] Meta Object Facility (MOF) Specification v1.4. OMG, *Document formal/02-04-03,* [Online]. Available: http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf, 2002.

[18] Ontology Definition Meta-Model, Preliminary Revised, *Submission to OMG RFP ad/2003-03-04, Vol.1* [Online]. Available from: http://codip.grci.com/odm/draft/, August 2004.

[19] OMG XMI Specification, v1.2., *OMG Document formal/02-01-01*, [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2002-01-01, 2002.

[20] E. Seidewitz, What Models Mean, *IEEE Software*, Vol. 20(5), pp. 26-32, 2003.

[21] E. Sirin and B. Parsia, An OWL DL Reasoner, Proceedings *International Workshop on Description Logics (DL2004),* British Columbia, Canada, 6. - 8. June 2004.

[22] R. Soley, MDA, An Introduction [Online].Available: http://www.omg.org/mda/mda_files/Soley-MDA/MDA-Seminar-Soley.htm, 2004.

[23] T. Frühwirth, Theory and Practice of Constraint handling rules, Special Issue on Constraint Logic Programming (P. Stuckey and K. Marriot, Eds.), *Journal of Logic Programming*, Vol 37(1-3), October 1998, pp.95-138.

[24] S. Abdennadher, E. Krämer, M. Saft and M. Schmauss, JACK: A Java Constraint Kit, *Electronic Notes in Theoretical Computer Science,* Vol. 64, 2000.

[25] B. Selic, The Pragmatics of Model-Driven Development, *IEEE Software*, Vol.20 (5), pp. 19-25, 2003.

[26] S. J. Mellor, A. N. Clark, T., Futagami, Guest Editors'Introduction: Model-Driven Development, *IEEE Software,* Vol. 20, No. 5, pp.14-18, Sep/Oct, 2003.

[27] T. Frühwirth, P. Hanschke, Terminological Reasoning with Constraint Handling Rules, *Chapter in Principles and Practice of Constraint Programming (P. Van Hentenryck and V.J. Saraswat, Eds.)*, MIT Press, 1995.

[28] M., Matula, NetBeans Metadata Repository, [Online].Available: http://mdr.netbeans.org/MDR-whitepaper.pdf , 2003.