

A METHOD FOR IMPLEMENTATION DESCRIPTION LOGIC REASONER

Nenad Krdžavac, Dragan Gašević

This paper presents implementation details of an ALC (Attribute Language with Complement) 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 respect software engineering standards, and their authors did not describe models of the reasoners in a standard (i.e. Unified Modeling Language - UML) notation. In this paper we briefly describe basic characteristics of MDA, give the definition of syntax and semantics of ALC description logic, and describe basic reasoning techniques in that logic. Then we outline a method for development an ALC description logic reasoner using MDA standards.

1. INTRODUCTION

Almost every software project needs an analysis of the range of problems that the software being developed should solve [22]. One way is to specify and build the system using modeling tools. For example, modeling tool such as UML supports full life development cycle of such software: design, implementation, deployment, maintenance, evaluation, and integration with later systems [27]. 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 [21]. MDA achieves three primary goals through architectural separation of concerns: portability, interoperability and reusability [19]. According to MDA concepts, systems are developed via transformation of models. MDA defines two basic kinds of models: PIM (Platform Independent Model) and PSM (Platform Specific Model). Developers start by creating a PIM and transform the model step-by-step into a more platform specific model [14]. A possible application domain for using MDA standards are knowledge representation languages,

2000 Mathematics Subject Classification: 68N15

Keywords and Phrases: Description logic reasoner, attribute language with complement, ontology definition meta-model, model driven architecture, tableaux algorithm.

like languages for developing ontologies. For example, there are some efforts to develop MDA-based ontology languages, namely Ontology Definition Meta-model (ODM) and Ontology UML Profile (OUP) [12]. The core of the ODM structure is the Description Logics (DLs) meta-model (see Fig.3). 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) [3]. One of the most important constructive properties of DLs is reasoning services, which can be applied as reasoning with ontologies. Some implemented DLs reasoners [16], [13], [26], which are publicly available, can reason with ontologies, but the authors of those reasoners did not implement such reasoners by using advanced model engineering techniques (e.g. MDA) and the reasoners do not respect these software standards.

The goal of this paper is to survey implementation details of a description logic reasoner, based on Object Modeling Group’s (OMG’s) DL meta-model, that is a part of the ODM meta-model [22]. In section 2, we look at some basic concepts of ALC description logic and describe basic reasoning service. Section 3 describes MDA for ODM as well as the DL meta-model through the four-level MDA architecture. 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, description logics (DLs) evolved from semantic networks and frame systems, mainly to satisfy the need of giving a formal semantics to these formalisms [17]. As the name DLs indicates, one of characteristics of these languages is that they are equipped with formal logic-based semantics. A knowledge base (KB) developed using DLs comprises two components, TBox and ABox [3]. The terminology or schema (i.e. the vocabulary of the application domain) is called TBox. In our case, we use only unfoldable terminologies. More details about the terminologies can be found in [15], [17]. On the other hand, the name of assertions is ABox that represents named individuals expressed in terms of the vocabulary [22]. The definition of TBox and ABox is given in [3], [17]. One more distinguished feature is the emphasis on reasoning as a central service: reasoning allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base [3]. 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. ALC itself is a notational variant of the multi-modal logic K_{ω} [3], [28]. According to [17], for some extensions of the logic, like Description Logics with Concrete Domain (for example ALC(D)), there is no

counterpart in modal logic or other areas of mathematical logic. According to [17], syntax and semantic of ALC logic (with example) is shown in next definitions.

Definition 1. (Syntax of ALC language). *Let N_C and N_R be disjoint and countably infinite set of concepts and role names. The set of ALC-concepts is the smallest set, such that:*

- Every concept name $A \in N_C$ is an ALC concept.
- 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.

Definition 2. (Semantics of ALC language) *An ALC interpretation is pair (Δ^I, \cdot^I) where Δ^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 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*

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

EXAMPLE 2.1. 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 $(Human \sqcap \forall hasChild. Male)$ represents all persons that have only male children.

Standard reasoning services offered by DLs are [17]:

- Decide whether a concept C is satisfiable, i.e. whether it can have any instances (concept satisfiability).
- 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
- Decide whether a given ABox is consistent, i.e. whether a world as described by the ABox may exist (ABox consistency).

In literature [15], [13], authors have been considered other reasoning services, but according to [17], they can be reduced to the basic ones listed above. These reasoning problems are called “standard” reasoning tasks, but a non-standard reasoning service is, for example, unification of two concept patterns [17]. A survey of reasoning services in DLs, and formal definition of the some reasoning tasks is given in [9], [15]. Concept subsumption can be transposed into an equivalent satisfiability

problem [15], and validity of this transformation is clear from the semantics of subsumption. Satisfiability problem can be solved using algorithm based on tableaux calculus [15], [17].

2.1. Tableaux algorithm for ALC description logic

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 $D^I \neq \emptyset$ [29]. In general, tableaux algorithms decide whether a concept is satisfiable by trying to construct a model for it [20]. A tableau is a graph which represents such a model, with nodes corresponding to individuals and edges corresponding to relationships between individuals. Tableaux algorithm uses tree (T) to represent model being constructed [15]. Expansion rules for ALC logic are shown on Figure.1.

<p>\sqcap-rule: if $((C_1 \sqcap C_2) \in \mathcal{L}(x) \& \{C_1, C_2\} \not\subseteq \mathcal{L}(x))$ then $\rightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$;</p> <p>$\sqcup$-rule: if $((C_1 \sqcup C_2) \in \mathcal{L}(x) \& \{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset)$ then Begin 1. save T; 2. try $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C_1\}$, if that lead to clash, then restore T and 3. try $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C_2\}$; End.</p> <p>$\exists$-rule: if $((\exists R.C \in \mathcal{L}(x)) \& (\neg(\exists y) \text{ s.t. } \mathcal{L}(\langle x, y \rangle) = R \& C \in \mathcal{L}(y)))$ then Begin Create a new node “y” and edge $\langle x, y \rangle$ with $\mathcal{L}(y) = \{C\} \& \mathcal{L}(x, y) = \{R\}$; End.</p> <p>$\forall$-rule: if $((\forall R.C \in \mathcal{L}(x)) \& ((\exists y) \text{ s.t. } \mathcal{L}(\langle x, y \rangle) = R \& C \notin \mathcal{L}(y)))$ then Begin $\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{C\}$; End</p>
--

Figure 1: Expansion rules for reasoning in ALC description logic

The algorithm is guaranteed to terminate [15]. The \sqcap, \sqcup and \exists rules, can only be applied once to any given concept expression C in set of concept expressions called $L(x)$. The \forall rule, can be applied many times to a given $\forall R.C$ expression in $L(x)$ but only once to a given edge (x, y) . Applying a rule to a concept expression C extends the labeling with a concept expression which is always strictly smaller than C . The \sqcup is non-deterministic rule, but other rules are deterministic [15].

3. MDA FOR ODM

The basic principle “Everything is object” was important in the 80’s to set up object-oriented technologies [5]. Beside object-oriented technology known is model-engineering concept which basic principle is “Everything is model”. Using

DLs “terminology” we can say that model-engineering subsumes object-oriented technology. 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 [5]. The central part of

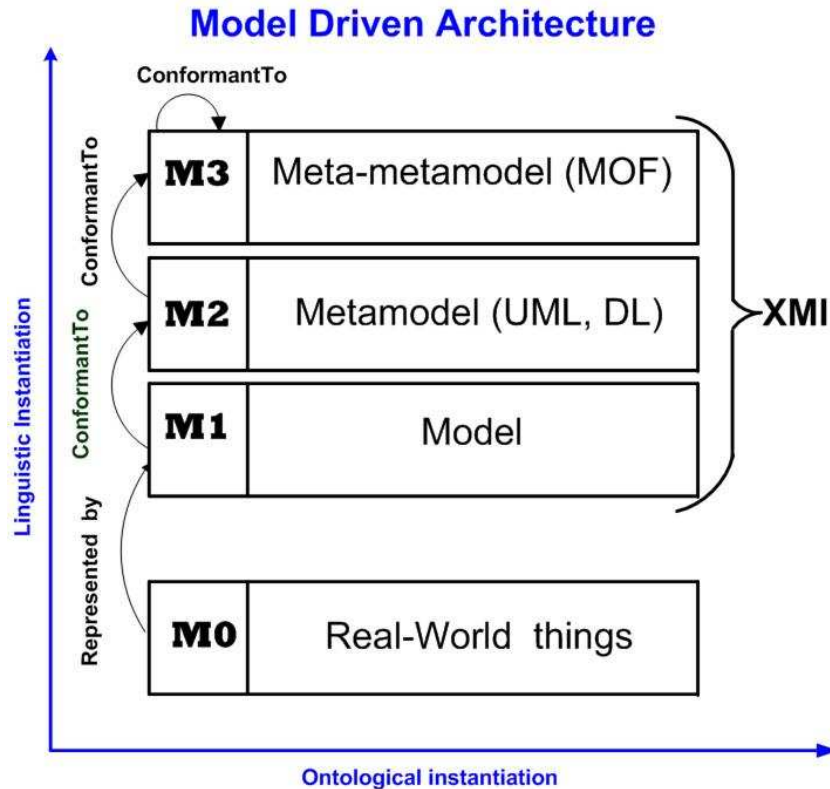


Figure 2. The four-layer MDA and its Ontological instance of relations (linguistics and ontological) (see [5])

MDA is the four-layer architecture that has a number of standards defined at each of its layers (see Figure 2). 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 meta-data management and language definition [21]. This is a self-defined language intended for defining meta-models. According to [5], the organization of the classical four-level architecture of OMG MDA can be more precisely named 3+1 (see Fig. 2). Actually, MDA layers are called linguistic layers. On the other hand, concepts from the same linguistic layer can be at different ontological layers [2]. MDA separates subject matters

so that application- oriented models are independently reusable across multiple implementations and vice versa [18]. The main advantages of the use of MDA concepts for implementation of our reasoner are the following:

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

According to [18], in an increasing number of application areas that use MDA, we can generate much of the application code directly from models. In terms of MDA a meta-model makes statements about what can be expressed in the valid models of a certain modeling language. In fact, a meta-model is a model of a modeling language [24]. Examples of the MDA's meta-models are UML, Common Warehouse Meta-model (CWM), etc. The MDA's meta-model layer is usually denoted as M2. At this layer we can define a new meta-model (e.g., a modeling language) that would cover some domain specific applications (e.g., ontology development). 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. [12]. There is an XML-based standard for sharing meta-data that can be used for all of the MDA's layers. This standard is called XML Meta-data Interchange (XMI) [23]. The bottom layer is the instance layer (M0). According to [12], there are two different approaches to explain this layer:

- The instance layer contains instances of the concepts defined at a model layer (M1), e.g objects in some programming languages
- 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.

In object-oriented paradigm we use “instanceOf” and “inherits” to establish relations between classes and objects. In model-engineering we use relations “representedBy” and “conformTo” [5], (see Fig. 2). Both model-engineering and object technologies approaches to software development can be viewed as complementary approaches [5]. In MDA technological space is defined ODM at M2 layer, as shown on Fig. 3. ODM in its structure includes several meta-models [22], proposed by OMG (www.omg.org). The core of this architecture is DL meta-model. MDA-based repositories can be used for storing models and meta-models. According to [22], the DL meta-model defines basic minimally constrained DL for use as a core, common component for the ODM and as a Knowledge Base in Knowledge

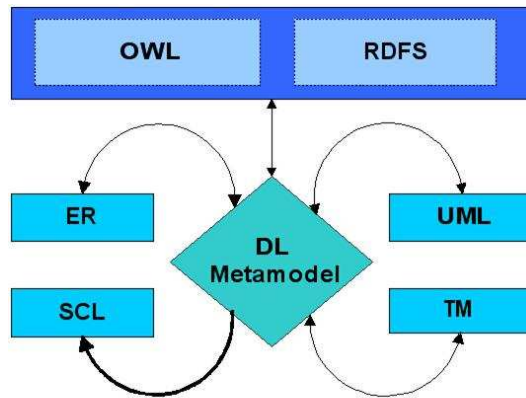


Figure 3. Structure of ODM (see [22])

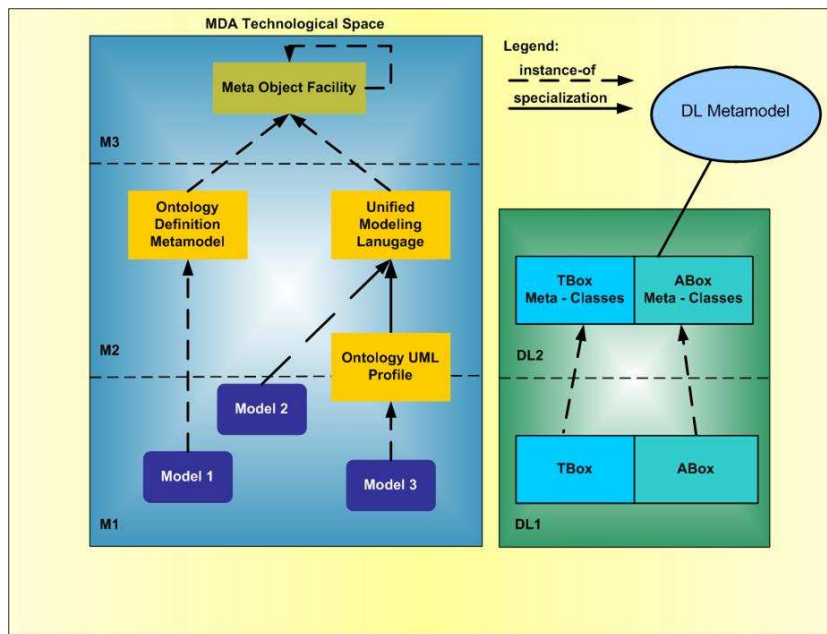


Figure 4. Description logic meta-model in MDA technological space

Representation System that use the ODM. DL meta-model is the core of all bi-directional mappings from the other meta-models (i.e. OWL and RDF(S), UML, etc.). The DL meta-model is not intended to be used for ontology development in its own right. One of main requirements for ODM is that it should be designed to comprehend common ontology concepts [8]. To be useful and effective a model must have a few characteristics such as: abstraction, understandability, accuracy,

predictiveness, and inexpensive [25]. The DL meta-model satisfy these characteristics and we decided to implement a reasoner based on that meta-model. In the definition of DL knowledge bases (KB) [8], [3], TBox and ABox are defined as two component of such a KB, and they are resided at two different ontological layers. In the context of MDA architecture, objects represented by TBox and ABox belong to the same linguistic layer (DL1 level in Fig. 4). ODM concepts are modeled by MOF [8], and the fundamental concept in ODM is class, which is an (linguistic) instance of MOF class (see Fig. 4). The goal of the MOF is to provide a framework and services to enable both model and meta-data driven systems.

MOF defines a set of reflective APIs consisting of reflective interfaces. JMI specification defines dynamic, platform neutral infrastructure that enables the creation, storage, access, discovery, and exchange of metadata [7]. 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 [12]. To generate JMI interfaces from the meta-model we used OMG' XML Metadata Interchange standard (XMI), which provides a mapping from MOF to XML [7].

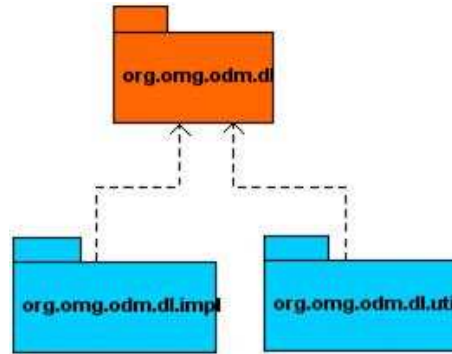


Figure 5. Packages for generated Description logic reasoner

4. IMPLEMENTATION DETAILS

The OMG DL meta-model is described in UML language (M2 layer, Fig.4). In UML language we can define packages (see Fig. 5). Following that package modeling organization we develop Java implementation packages. By using some of present MDA-repositories we have a feature to produce JMI compliant code for the DL meta-model. According to the OMG API specification for ODM meta-models (<http://sylvester.va.grci.com/codipsite/odm/draft/>) the DL meta-model has three “packages” shown in Fig. 5. The MOF reflective package contains eight abstract interfaces that are extended by the generated interfaces [7]. The resulting JMI interfaces allow users to create, update, and access instances of the meta-model using a Java program. JMI interfaces are a proof of how we can automatically transform a model-driven developed system, from a high-level abstract application subject matter models into a running system. In object oriented-paradigm we can not use benefits of such a meta-modeling approach.

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

- Instance interfaces (example: Concept, see Fig. 6)
Instance interfaces contain methods for accessing instance-level attributes and references and invoking instance-level operations [7]. In the DL meta-model

on the top of hierarchy is the metaclass Term. The JMI instance interface, which corresponds to Term meta-class, does not contain such methods and extends RefObject 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 {
}
```

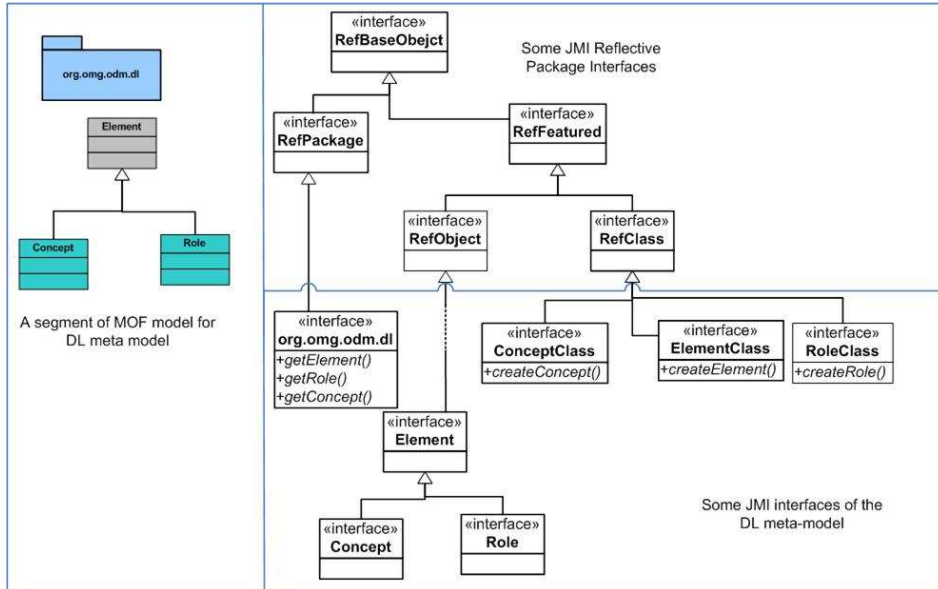


Figure 6. Some JMI interfaces generated for DL meta-model

- Class proxy interfaces (example: ConceptClass, see Fig. 6)
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);
}
```

4.1 Practical experience during implementation

Beside all the advantages of the metamodel-based approach to the DL reasoner implementation, we have faced 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 (<http://www.gentleware.com/>) to save the DL meta-model in an XMI file. With the tool **uml2mof.jar** (www.mdr.netbeans.org) we generated a MOF XMI file. Using Java NetBeans 3.5.1 (<http://mdr.netbeans.org/mdrexplorer.html>) we generated the JMI interfaces. Here are some experiences with using the DL meta-model:

- 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, that for different metaclasses have the same name, so we had to change them manually.
- Association ends between the metaclasses *Term* and *Expression* do not have names, so we have to name them, as we could not generate JMI interfaces.
- The DL meta-model cannot support a very important class of DLs called description logics with concrete domain and functional dependences, as it does not have metaclasses defined in their definitions of both syntax and semantics [17], [20]. Accordingly, the meta-model would not be able to support reasoning on a knowledge base in such logics.
- During the generation of JMI interfaces all the OCL constraints were ignored and we had to implement the constraints manually.

For implementation reasoning rules, can be used, language called *Constraint Handling Rules* (CHR) [11]. The inference rules (see Fig. 1) can be directly written in the language [1]. To integrate these rules in our meta-model environment, can be used *The Java Constraint Kit* (JCK) [1]. This package contains a generic search engine to solve constraint problems. Practical and theoretical aspects of Constraint Handling Rules are given in [11].

5. CONCLUSION AND FUTURE WORK

In this paper we analyzed a method for implementation a description logic reasoner (The LoRD) using model-engineering paradigm. We give some aspects of description logics, especially ALC description logics, and give references where readers can find more information about the logic and its extensions. We hope that readers of this paper can find useful information, how to apply model-engineering approach in implementation such sort of software, especially for implementation theorem provers. Our future work will be focused on extensions of the reasoner to support ontology reasoning, especially OWL ontologies and implementation tableaux algorithms for description logic with concrete domains and functional dependencies, in model-engineering paradigm.

REFERENCES

1. S. ABDENNADHER, E. KRÄMER, M. SAFT, M. SCHMAUSS: *JACK: A Java Constraint Kit*. Electronic Notes in Theoretical Computer Science, **64** (2000).
2. C. ATKINSON, T. KÜHNE: *Model-Driven Development: A Metamodeling Foundation*. IEEE Software, **20**, No. 5 (2003), 36–41.
3. F. BAADER, D. CALVANESE, D. MCGUINNESS, D. NARDI, P. PATEL-SCHNEIDER: *The Description Logics Handbook-Theory, Implementation and Application*. Cambridge University Press, (2003).
4. D. BERARDI, A. CALI, D. CALVANESE, G. GE DIACOMO: *Reasoning on UML Class Diagrams*. Technical Report 11-03, Dipartimento di Informatica e Sistemistica, Università di Roma, "La Sapienza", [Online]: <http://www.inf.unibz.it/calvanese/teaching/03-ESSLLI/>, (2003).
5. J. BÈZIVIN: *In Search of a Basic Principle for Model Driven Architecture*. The European Journal for The Informatics Professional, Vol. V, No. 2, April (2004).
6. J. BÈZIVIN, F. JOUAULT, P. VALDURIEZ: *On the Need for Megamodels*. In Proceedings of the International Workshop Best Practices for Model Driven Software Development, Vancouver, Canada, (2004).
7. R. DIRCKZE (spec. leader): *Java Metadata Interface (JMI) Specification Version 1.0*, [Online]: <http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html>, (2002).
8. D. DJURIĆ, D. GAŠEVIĆ, V. DEVEDŽIĆ: *A MDA-based Approach to the Ontology Definition Metamodel*. In Proceedings of the 4th International Workshop on Computer Intelligence and Information Technology, Niš, Serbia and Montenegro, (2003), 51–54.
9. F.M. DONINI, M. LANZERINI, D. NARDI, A. SCHAEFER: *Reasoning in Description Logics*. In Gerhard Brewka editor, *Foundation of Knowledge Representation*, CSLI-Publication, (1996), 191–236.
10. 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).
11. 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*, **37** (1-3), October (1998), 95–138.
12. D. GAŠEVIĆ, D.DJURIĆ, V.DEVEDŽIĆ, V.DAMJANOVIĆ: *Approaching MDA and OWL Ontologies Through Technological Spaces*. In Proceedings of the 3rd International Workshop on Software-Model Engineering, Lisbon, Portugal,(2004).
13. V. HAARSLEV AND R. MOLLER: *Description of the RACER System and its Applications*. In Proceedings International Workshop on Description Logics (DL-2001), Stanford, USA, (2001).
14. P. HNETYNKA, M. PIŠE: *Hand-written vs. MOF-based Metadata Repositories: The SOFA Experience*. In Proceedings of the 11th IEEE international Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04), Brno, Czech Republic, (2004).

15. I. HORROCKS: *Optimising Tableau Decision Procedures for Description Logics*, PhD Thesis, University of Manchester, (1997).
16. I. HORROCKS: *The FACT System*. In. H. de Swart, editor, Automated Reasoning with Analytic Tableaux and Related Methods, International Conference Tableaux'98, number 1397 in Lecture Notes in Artificial Intelligence, Springer-Verlag, May (1998), 307–312.
17. C. LUTZ: *The Complexity of Description Logics with Concrete Domains*. PhD thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, (2002).
18. S. MELLOR, A. CLARK, T. FUTAGAMI: *Guest Editors'Introduction: Model-Driven Development*. IEEE Software **20**, No. 5, Sep/Oct (2003), 14–18.
19. J. MILLER, J. MUKERJI (eds), MDA Guide Version 1.01, OMG Document Number: omg/2003-06-01, [Online]: <http://www.omg.org/docs/omg/03-06-01.pdf>, (2003).
20. M. MILIČIĆ: *Description Logics with Concrete Domains and Functional Dependencies*. Master's Thesis, Technical University Dresden, Dresden, May (2004).
21. *Meta Object Facility (MOF) Specification v1.4*. OMG Document formal/02-04-03 [Online]: <http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf>, (2002).
22. *Ontology Definition Meta-Model*. Preliminary Revised Submission to OMG RFP ad/2003-03-04, Vol.1, [Online]:<http://codip.grci.com/odm/draft/>, August (2004).
23. *OMG XMI Specification, v1.2*, OMG Document formal/02-01-01 [Online]: <http://www.omg.org/cgi-bin/doc?formal/2002-01-01>, (2002).
24. E. SEIDEWITZ: *What Models Mean*. IEEE Software, **20** (5), (2003), 26–32.
25. B. SELIĆ: *The Pragmatics of Model-Driven Development*. IEEE Software, **20** (5), (2003), 19–25.
26. E. SIRIN, B. PARSIA: *An OWL DL Reasoner*. Proceedings International Workshop on Description Logics (DL2004), British Columbia, Canada, 6. - 8. June (2004).
27. R. SOLEY: *MDA, An Introduction*, [Online]: <http://www.omg.org>, (2004).
28. M. SCHMIDT-SCHAUSS, G. SMOLKA: *Attributive concept descriptions with complements*. Journal of Artificial Intelligence, **48** (1), (1991), 1–26.
29. F. BAADER, U. SATTLER: *An Overview of Tableau Algorithms for Description Logics*. Studia Logica, **69** (2001), 5–40.

1. Faculty of Electrical Engineering,
Bulevar Kralja Aleksandra 73,
11000 Belgrade,
Serbia and Montenegro
E-mail: nenadkr@galeb.etf.bg.ac.yu

(Received February 11, 2005)

2. Scholl of Interactive Arts and Technology,
Simon Fraser University Surrey,
2400 Central City,
10153 King George Hwy,
Surrey, BC V3T 2W1,
Canada
E-mail: dgasevic@sfu.ca