

**ON
knowledge
TO**

BOR: a Pragmatic DAML+OIL Reasoner

Kiril Simov, Stanislav Jordanov

OntoText Lab.

| | |
|----------------------------|-------------------------------------|
| Identifier | 40 |
| Class | Deliverable |
| Version | 1.0 |
| Version date | June 2002 |
| Status | Final |
| Distribution | Public |
| Responsible Partner | OntoText Lab., Sirma AI Ltd. |

On-To-Knowledge Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-1999-10132. The partners in this project are: Vrije Universiteit Amsterdam (VU) (co-ordinator), NL; the University of Searchrlsrue, Germany; Schweizerische Lebensversicherungs- und Rentenanstalt / Swiss Life, Switzerland; British Telecommunications plc, UK; CognIT a.s, Norway; EnerSearch AB, Sweden; AIdministrator Nederland BV, NL; OntoText Lab., Sirma AI EOOD, Bulgaria.

Vrije Universiteit Amsterdam (VU)

Faculty of Sciences, Division of Mathematics and
Computer Science
De Boelelaan 1081a
1081 HV Amsterdam, the Netherlands
Fax and Answering machine: +31-(0)20-872 27 22
Mobil phone: +31-(0)6-51850619
Contactperson: Dieter Fensel
E-mail: dieter@cs.vu.nl

University of Karlsruhe

Institute AIFB
Searchiserstr. 12
D-76128 Searchrlsrue, Germany
Tel: +49-721-608392
Fax: +49-721-693717
Contactperson: R. Studer
E-mail: studer@aifb.uni-searchrlsrue.de

Schweizerische Lebensversicherungs- und Rentenanstalt / Swiss Life

Swiss Life Information Systems Research Group
General Guisan-Quai 40
8022 Zürich, Switzerland
Tel: (41 1) 284 4061, Fax: (41 1)284 6913
Contactperson: Ulrich Reimer
E-mail: Ulrich.Reimer@swisslife.ch

British Telecommunications plc

BT Adastral Park
Martlesham Heath
IP5 3RE Ipswich, UK
Tel: (44 1473)605536, Fax: (44 1473)642459
Contactperson: John Davies
E-mail: John.nj.Davies@bt.com

CognIT a.s

Busterudgt 1.
N-1754 Halden, Norway
Tel: +47 69 1770 44, Fax: +47 669 006 12
Contactperson: Bernt. A. Bremdal
E-mail: bernt@cognit.no

EnerSearch AB

SE 205 09 Malmö, Sweden
Tel: +46 40 25 58 25; Fax: +46 40 611 51 84
Contactperson: Hans Ottosson
E-mail: hans.ottosson@enersearch.se

AIdministrator Nederland BV

Julianaplein 14B
3817CS Amersfoort, NL
Tel: (31-33)4659987, Fax: (31-33)4659987
Contactperson: Jos van der Meer
E-mail: Jos.van.der.Meer@aidministrator.nl

OntoText Lab.

Sirma AI EOOD - Artificial Intelligence Labs
38A Chr. Botev blvd, 1000 Sofia, Bulgaria
Tel: +359 2 981 23 38; Fax: +359 2 981 90 58
Contactperson: Atanas Kiryakov
E-mail: Atanas.Kiryakov@sirma.bg

Abstract

This report outlines the implementation of a DAML+OIL reasoner within the On-To-Knowledge project. The reasoner has to work in tight connection with the SESAME RDF(S) repository and the Ontology Middleware Module implemented within the project. SESAME supports the storage, management, and querying of ontologies and instances in RDF(S) repositories using the RQL language and set of APIs. The document extends section 3 of [Kiryakov et al, 2002] and provides more implementation details on top of it. The appropriate evaluation framework ([DLBMark 2002]) is also discussed.

Acknowledgements

The research reported here was carried out in the course of the On-To-Knowledge project. This project is partially funded by the IST Programme of the Commission of the European Communities as project number IST-1999-10132. The partners in this project are: Vrije Universiteit Amsterdam VUA (coordinator, NL), University of Karlsruhe (Germany), Swiss Life (Switzerland), BT plc (UK), CognIT a.s. (Norway), EnerSearch AB (Sweden), AIdministrators Nederland BV (NL), OntoText Lab. (BG). We wish to particularly thank to the Arjohn Kampman, Frank van Harmelen, Jeen Broekstra, Marin Dimitrov, and Michel Klein for the instant support and the fruitful discussions.

Contents

| | | |
|----------|----------------------------------------------------------------------|-----------|
| 1 | Introduction | 5 |
| 1.1 | Reasoning in On-To-Knowledge Project..... | 5 |
| 1.1.1 | Discussion on Architecture with Separate Storage and Reasoning | 5 |
| 1.1.2 | Requirements Towards a Reasoning Service for On-To-Knowledge | 6 |
| 1.1.3 | DAML+OIL Reasoner Built-In the Repository..... | 6 |
| 1.2 | DAML+OIL and RDF(S) | 6 |
| 1.2.1 | The Incompatible Semantics | 7 |
| 1.2.2 | Incompatible Reasoning | 7 |
| 1.2.3 | Separation of DAML+OIL from RDF(S)..... | 8 |
| 1.3 | DAML+OIL as a Description Logic..... | 9 |
| 1.4 | Inference Services in Description Logics | 9 |
| 1.4.1 | Terminological Reasoning..... | 9 |
| 1.4.2 | Instance Reasoning..... | 9 |
| 2 | BOR: Instance Reasoning in On-To-Knowledge Project..... | 10 |
| 2.1 | Instance Reasoning Services in BOR | 9 |
| 2.1.1 | Realisation | 11 |
| 2.1.2 | Instance checking | 12 |
| 2.1.3 | Retrieval | 12 |
| 2.1.4 | Retrieval of components..... | 12 |
| 2.1.5 | Model Checking | 13 |
| 2.1.6 | Minimal Sub-Ontology Extraction..... | 13 |
| 2.2 | Functional Interfaces to a DAML+OIL Reasoner | 13 |
| 3 | Evaluation | 13 |
| 4 | References | 13 |

1 Introduction

In this section we describe the prerequisite for the implementation of DAML+OIL instance reasoner, called BOR, within On-To-Knowledge project - task "11.3 Reasoning Enhancements". The section starts with a bit more general discussion on the various types of reasoning, DAML+OIL and RDF(S). In the next section we describe the implemented inference services.

This document extends section 3 of [Kiryakov et al, 2002] and provides more implementation details on top of it. It is also important to mention that the name of the reasoning service described here changed twice due to different reasons – it was first called PRONTO, next LINRO and finally BOR. Any references to one of the previous names should be considered as references to BOR.

1.1 Reasoning in On-To-Knowledge Project

The represented knowledge allows (potentially) infinite number of possible uses, but within a practical system mainly the typical usage cases are implemented efficiently. In our view within the On-To-Knowledge project such typical usage cases are (1) Ontology development and (2) Ontology use. Ontology development requires the following basic tasks to be supported:

- Checking whether a class definition is consistent by itself or with respect to a set of other class descriptions;
- Checking whether a given class definition is more general than another class definition;
- Construction of explicit hierarchy of class names on the base of their class definitions.

We call these reasoning tasks "**Terminological Reasoning**". Ontology use involves first an already developed ontology in which the classes are defined and (possibly) the relations between them are explicitly represented (after some terminological reasoning) and next instance data of much higher magnitude, say, thousands or millions of instances. This usage case requires task such as:

- Find the most specific classes that describe a partially specified instance;
- Find all instances in the dataset which are instances of a given class definition;
- More complex queries involving instance data. Such could be getting all pairs of instances related in a way;
- Checking the consistence of the instance data with respect to the ontology.

We call these reasoning tasks "**Instance Reasoning**".

1.1.1 Discussion on Architecture with Separate Storage and Reasoning

Due to a number of different reasons the infrastructure initially developed under the On-To-Knowledge project separates the ontology and data representation from the reasoning over them. The storage, management, and querying of ontologies and instances is handled by the system SESAME – an RDF(S) repository which supports the RQL language. If more expressive reasoning is necessary (in more expressive language like OIL or DAML+OIL) then the corresponding information should be sent to an external reasoner (say, the FaCT system) that processes it and returns the answer back. Although such solution is appealing in terms of reuse of existing tools and compliance, in our view it can hardly provide good level of performance and interoperability in cases when the ontology and/or the instance data is very large. There are two possible approaches for implementation of such architecture:

- (i) Only the relevant parts of the ontology and the instance data are sent to the external reasoner. Such solution minimizes the exchange overhead between the two system, but imposes the question how these relevant parts are determined. In general the problem of fragmentation of an ontology and/or instance data into non-interacting chunks can require

already considerable amount of reasoning.

- (ii) The external reasoner to support its own copy of the ontology and/or instance data. In this case the reasoner duplicates a lot of the functions of the repository, one way or another everything “known” to the repository should be passed to the reasoner.

1.1.2 Requirements Towards a Reasoning Service for On-To-Knowledge

Both approaches are in contradiction with the expectation for the SESAME usage – thousands of classes and millions of instances. Thus, the integration with external reasoner seems feasible only for small ontologies. Using only SESAME is also not satisfactory solution because it supports only RDF(S) which expressive power is insufficient to support ontologies and instance data for domains and applications that require the full inventory of ontological languages such as DAML+OIL.

The above arguments motivated our position that the reasoning services in the project has to satisfy the following requirements:

- To be aware of the semantics of DAML+OIL
- To be efficient in the typical usage cases
 - Ontology development – Terminological Reasoning, usually no instances are involved;
 - Ontology use – Instance Reasoning, stable ontology with huge instance data.
- To be in close integration with the RDF(S) repository.

1.1.3 DAML+OIL Reasoner Built-In the Repository

In order to satisfy the above stated requirements we have implemented the instance reasoning within the On-To-Knowledge project from scratch. The exact language and services to be supported are discussed here as well as the implementation itself. DAML+OIL represents an obvious choice for language to be supported. DAML+OIL is shortly commented and compared to RDF(S) and the description logics in the following sub-sections.

1.2 DAML+OIL and RDF(S)

DAML+OIL is a logical language whose syntax is built-in the RDF(S) and the semantics of the language is partially defined within the semantics of RDF(S) (for a definition of model-theoretical semantics of RDF(S) see [Hayes, 2001]). As a logical language DAML+OIL is furnished with a model-theoretical semantics presented in [Patel-Schneider, 2001]. Although this semantics is not a traditional one it is very close to the standard model-theoretical semantics in the classical logic. A sample¹ DAML+OIL statement follows:

¹ Taken from <http://www.daml.org/2001/03/daml+oil-ex.daml>

```

<daml:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>

  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasFather"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

```

1.2.1 *The Incompatible Semantics*

There are certain elements of RDF(S) on which DAML+OIL depends (see [Patel-Schneider and van Harmelen, 2001]): (1) RDF triple structure and (2) RDF Schema constructions. Thus one can consider a DAML+OIL ontology as an RDF(S) ontology and can assign to it the corresponding RDF(S) semantics. But DAML+OIL has its own model-theoretical semantics which is different from the one of RDF(S) and ignores certain RDF(S) constructions and features: (1) reification, (2) containers, and (3) meta-classes. Thus we have a language which is situated within another language, but which separate itself from the semantics of the surrounding language.

Let us clarify the point from the previous paragraph. Reification is missing from DAML+OIL because the semantics of the language excludes the possibility the same resources to be used for both a class and a property. Container is a notion outside the definition of DAML+OIL ontology. Meta-classes are forbidden in DAML+OIL. Of course one can use the same resource as a class name and as instance name, but such resource couldn't receive the intended semantics in DAML+OIL. Thus real meta-class hierarchy is impossible in DAML+OIL because if a resource is used in a syntactic construction requiring an instance then this resource is always interpreted as singleton. Thus, if the same resource is also used as a class then it is very special class having only one instance. In our view this is not an advantage of the definition of DAML+OIL, but one unfortunate consequence of its definition. We offer a refinement of the definition of DAML+OIL ontology in the next section defining the notion of well-defined DAML+OIL ontology².

A common misunderstanding is that DAML+OIL semantics is extension of the RDF(S) semantics. Although most of the primitives of the later have their cousins in the former, the formal interpretations are different. As mentioned earlier, meta-classes are not allowed in DAML+OIL. As a consequence DAML+OIL also does not support the flexible meta-modeling of RDF(S) (see [Pan and Horrocks, 2001] for details and criticism).

1.2.2 *Incompatible Reasoning*

The incompatibility between the DAML+OIL and RDF(S) semantics leads to principle problems with the compatibility of any reasoning services – each time when repository is interpreted and verified either one of the semantics should be use either the other. Although it is possible a sub-language of RDF(S) to be defined that is “forward-compatible” with DAML+OIL, there is no widely accepted dialect like this.

For the implementation of the DAML+OIL reasoner the strict semantic of the language will be used which means that the standard two-level model with disjoint sets of classes and instances will be supported. The appropriate distinctions are more precisely defined in the next sub-sections.

² Actually, the term “DAML+OIL repository” will be introduced.

1.2.3 Separation of DAML+OIL from RDF(S)

For the purpose of using DAML+OIL within the On-To-Knowledge project we can say that SESAME is appropriate for storing DAML+OIL ontologies and datasets because they are RDF(S) constructs. However, additional facilities are needed to process DAML+OIL in proper way – such, corresponding to the semantics of DAML+OIL.

In the rest of the document we are concerned with DAML+OIL ontologies and instance data represented as RDF(S) repositories in SESAME. Our separation of DAML+OIL from RDF(S) is based on the propositions made in the following two citations:

[van Harmelen et al. 2001], *"A DAML+OIL ontology consists of zero or more headers, followed by zero or more class elements, property elements, and instances."* and

[Patel-Schneider, 2001], *"A DAML-OIL knowledge base is a collection of RDF triples. Some of these triples are relevant to DAML-OIL and some are not. This semantics currently only treats the triples that are obviously relevant to DAML-OIL and ignores the rest."*

Thus, we will try to classify all the resources and statements in a SESAME repository containing a DAML+OIL knowledge into several categories. The resources are classified as follows:

- **Class** – all object classes defined within some class element of the DAML+OIL ontology (including the imported ones which minimally contain "#Thing" or "#Nothing" the two DAML+OIL standard classes for the universal and absurd concept). This set of resources contains all DAML+OIL classes defined in the ontology;
- **Property** – all object or datatype properties defined within some property element of the DAML+OIL ontology (including the imported ones) and thus this set of resources contains all DAML+OIL properties defined in the ontology;
- **Instance** – all resources that are used as instances in DAML+OIL elements. They can be used inside class element (by daml:hasValue element) or by instance statements.
- **Others** – all resources used in the DAML+OIL ontology outside the headers, which are not in the previous sets. Examples of such are the relational axioms.

In order one RDF(S) repository to be also a well-defined DAML+OIL repository we will require the sets Class, Property, and Instance to be pair-wise disjoint.

The statements in a DAML+OIL repository are classified into the following sets:

- **header** – containing all header statements;
- **ontology** – containing all class and property elements;
- **dataset** – containing all instance statements;
- **other** – containing all other statements.

Although the set 'other' is not necessary empty the statements in it don't have any impact on the DAML+OIL semantics of the ontology. The principle for classification of the statements is discussed in [Simov, 2002].

In the following text we will use the term 'Repository' to point to the SESAME RDF(S) repository which contains well-defined DAML+OIL repository. The inferences are concerned with the ontology and the dataset parts of such a repository. When it is necessary a distinction between the two types of knowledge (class and property versus instance knowledge) to be made, we use the term 'Ontology' for the ontological part of the repository and 'Dataset' for the instance part. All syntactic constructions defined in [van Harmelen et al. 2001] as elements are called in the following texts "elements." Instance statements are called instance statements.

1.3 DAML+OIL as a Description Logic

In this sub-section we discuss DAML+OIL as a kind of description logic in order to be able to reuse known algorithms and results on the tractability of such languages. The closest description logic known in the literature is *SHOQ(D)*, described in [Horrocks and Sattler, 2001]. The *SHOQ(D)* language primitives are presented below together with their DAML+OIL equivalents:

| <i>SHOQ(D)</i> | DAML+OIL | <i>SHOQ(D)</i> | DAML+OIL |
|--------------------|---------------------|----------------------|---------------------|
| Concepts | Classes | Disjunction | daml:unionOf |
| Roles | Properties | Negation | daml:complementOf |
| Individuals | Instances | Exists restriction | daml:hasClass |
| Nominals | daml:hasValue | Value restriction | daml:toClass |
| Concrete datatypes | Datatypes | At-least restriction | daml:maxCardinality |
| Conjunction | daml:intersectionOf | At-most restriction | daml:minCardinality |

Reverse role operator is missing in *SHOQ(D)* and thus (daml:inverseOf element) cannot be represented in *SHOQ(D)*. The rest of the primitives of DAML+OIL can be represented in *SHOQ(D)* as role inclusion axioms, transitivity axioms, generalized concept inclusion axioms, equality axioms, inequality axioms.

In [Horrocks and Sattler 2001] an inference procedure for terminological reasoning in *SHOQ(D)* is presented. We implemented it in our instance reasoner BOR in order to support some necessary terminological reasoning.

1.4 Inference Services in Description Logics

The fact that DAML+OIL could be considered as a description logic allows us to use the extensive investigations of these logics. A knowledge representation and reasoning system based on DLs is assumed to provide the number of reasoning facilities that can be separated in two groups: terminological and instance reasoning.

1.4.1 Terminological Reasoning

The four tasks below are the basics of the terminological reasoning – they are implemented in most of the systems, including FaCT.

- **Subsumption check** – checks if one class is a subclass of another;
- **Consistency check** – checks for inconsistent class definitions;
- **Taxonomy construction** – computes all subclass relations (including those which are not explicitly stated but that are implied by the given definitions);
- **Classification** – determines the classes that immediate subsume or are subsumed by a given class;

1.4.2 Instance Reasoning

The basic two tasks involving individuals are:

- **Realisation** – given a partial description of an individual (instance), finds the most specific concept that describes it;

- **Instance checking** – given a partial description of an individual (instance) and a class description, finds whether the class describes the instance;
- **Individual retrieval** – finds all individuals (instances) that are described by a given concept;

In description logics with negation most terminological reasoning problems are reduced to (in)consistency of a class definition. The most widely used technique for the (in)consistency checking is based on the notion of a **tableau**. On a very informal level a tableau is an abstract structure representing all explicit atomic knowledge about a set of instance variables (or names) following from a given ontology. A tableau is a base of construction of an interpretation in which the corresponding knowledge is satisfiable. Thus the decision procedure tries to show that a tableau corresponding to the current ontology and/or dataset exist. Sometimes the corresponding tableau is infinite and its existence is proven indirectly by construction of a completion. A **completion** is an abstract structure, which ensure the existence of a tableau. Instance Reasoning can be realized in a similar way but including the instance names from the dataset in the completion.

The reasoning procedure is designed as a set of rules. The rules are applied to an (incomplete) abstract structure explicating the knowledge stored in the structure and trying to construct a completion. The rules are classified in two different ways:

Deterministic versus non-deterministic

A deterministic rule modifies the current structure in one possible way. A non-deterministic rule could modify the structure in more than one way and thus leaves a choice to the system.

Extending versus non-extending

A rule is extending if its application introduce at least one new instance variable in the structure.

Obviously deterministic, non-extending rules are preferable in one implementation. The application of the rules is governed by meta-rules and control information:

A meta-rule could be as:

apply first the non-extending rules then extending ones

Control information could concern the typical problems that are expected and thus could require different meta-rules.

In the worst case for expressive languages like DAML+OIL (*SHOQ(D)*) the inference is very hard - usually **NP** and more. Thus in the systems usually some optimisations for the average case problems are implemented. Such optimisations includes heuristics to avoid “expensive” checks or incomplete inferences like *pseudo models* - sound, but incomplete inference based on structures similar to completions [Haarslev V., Mueller R., Turhan A.-Y., 2001].

In our work we are using all of these treasure of experience trying to adopt it to the typical cases of instance reasoning within the On-To-Knowledge project.

2 BOR: Instance Reasoning in the On-To-Knowledge Project

It was accepted that in order to make the reasoning tasks feasible within the time constraints we had to follow one graduate approach to the tasks that we had to implement. First we classified the possible datasets with respects to their complexity. On one end of the scale we put the datasets in which instance statements of an arbitrary complex form are presented. On the other end of the scale we put datasets which contain only ground instance statements. Such datasets are called ground datasets.

General instance statements

- a : C** - class statement: **C** is an arbitrary complex class expression
- (a,b) : R** - property statement: **R** is a property name

- $a = b$ - equality statement
 $a \neq b$ - inequality statement
 $\forall x.C(x)$ - universal statement : C is an arbitrary complex class expression

a, b are instance names, x is an instance variable

The class and the universal statements provide high expressivity within the datasets. They allow one to state additional ontological restrictions over a particular dataset. Usually these statements impose great computational problems to the inference procedure. Here are examples of such statements, $a : (\text{Mother} \cap (\geq 3 \text{ has-child}))$ states that a is a mother with more than two children. $\forall x.(\neg \text{Mother} \cup (\geq 3 \text{ has-child}))(x)$ states that each mother in the current domain has more than two children.

Ground instance statements

- $a : A$ - concept statement: A is a class name
 $(a,b) : R$ - property statement: R is a property name
 $a = b$ - equality statement
 $a \neq b$ - inequality statement

Datasets containing only ground instance statements allow very efficient implementation of inference procedures. Also ground datasets are typical for the case studies within the On-To-Knowledge project. It is important to mention that universal statements are consequences from the ontological knowledge and thus they can be expected to be always presented as constraints over the instance data.

In the rest of this section we describe the inference services that we have implemented as an extension of SESAME system. In addition to the instance reasoning we have implemented also the terminological reasoning because it is a prerequisite for the instance reasoning. Because of our approach to implementation discussed above we implemented terminological reasoning for ontologies that doesn't include general inclusion concept axioms, which results in universal statements in the datasets.

2.1 Instance Reasoning Services in BOR

We have implemented the following reasoning services within BOR.

2.1.1 Realisation

The task can be formally defined as follows:

Given: A - a set of instance statements, a - an instance resource in A , and C_1, \dots, C_n - class resource in the ontology

Find: C_{i1}, \dots, C_{im} the most specific classes that describe a (realisation of a)

The decision procedure can require consistency check of the augmented sets $A \cup \{a : \neg C_j\}$.

Our goal here was to minimise the number of the classes C_j for which this check will be done. We defined a special classes C_a for instance a in the following way:

Let a be an instance. Let R_{Null} be a set of properties such that for each property R in R_{Null} , there exists a class C_R such that for each instance b with $(a R b)$ in the dataset it follows that $(b \text{ type } C_R)$ is in the dataset. Let $R_{\text{N>}}$ be a set of properties such that there exists an instance b such that $(a R b)$ is in dataset and R is not in R_{Null} . For each property R in $R_{\text{N>}}$, let $S_{>R}$ be the set of classes such that for some instance b , $(a R b)$ is in the dataset and $(b \text{ type } C)$ is in the dataset. Let R_{Some} be a set of properties such that there exists an instance b such that $(a R b)$ is in the dataset, R is not in $R_{\text{N>}}$ and R is not in R_{Null} . For each property R in R_{Some} , $S_{\text{Some}R}$ is the set

of classes such that for some instance **b**, (**a R b**) is in the dataset and (**b type C**) is in the dataset. For each class **C** and each property **R**, let $n\langle R, C \rangle$ be the number of the different instances **b** such that (**a R b**) is in the dataset and (**b type C**) is in the dataset. Let **A** be the dataset. Then we defined the following concept:

$$\begin{aligned} \mathbf{Ca} = & (\cap \{C \mid (\mathbf{a} \text{ type } C) \text{ in } A\}) \cap \\ & (\cap \{((\forall R.CR) \cap (\geq n\langle R, CR \rangle R.CR)) \mid R \text{ in } RN_{all}\}) \cap \\ & (\cap \{(\geq n\langle R, C \rangle R.C) \mid R \text{ in } RN_{>} \text{ and } C \text{ in } S_{>}R\}) \cap \\ & (\cap \{(\exists R.C) \mid R \text{ in } RN_{some} \text{ and } C \text{ in } S_{some}R\}) \end{aligned}$$

First the class **Ca** is classified in the taxonomy. Then we restrict ourselves to the subclasses of this class. First we search for the most specific subclasses that still describe the instance **a**. It could happen that some of the immediate superclasses of **Ca** is element of the realization of **a**. If this is the case then some of the paths in the taxonomic graph under **Ca** will not contain any classes describing the instance **a**. Thus after the exploration of the subclasses under **Ca**, we check whether there are such paths and then we check the superclasses of **Ca**, which are not superclasses of some elements of the realization of the instance **a**.

2.1.2 Instance checking

The task can be formally defined as follows:

Given: **a** - an instance statements, and **C** - a class element

Find: whether **a** is an instance of by **C**

In this case we first check the consistency of the following class $\mathbf{Ca} \cap \mathbf{C}$ if it is inconsistent then **a** is not an instance of **C**. If it is consistent, then we realised the instance **a** with respect to the subclasses of **C** and if there exists a realization of **a**, then **a** is an instance of **C**.

2.1.3 Retrieval

The task can be formally defined as follows:

Given: **A** - a set of instance statements, and **C** - a class element

Find: **a1**, ... **am** - the instance resources in **A** that are instances of **C**

The decision procedure can require consistency check of the augmented sets $\mathbf{A} \cup \{\mathbf{a} : \neg\mathbf{C}\}$ for all instances in **A**.

First we calculate the realization each instance **a** in the dataset **A**. In fact we consider this to be supported all the time during the use of the ontology and the dataset. Thus we have all the instances in the dataset already connected with their realisations. In this way for each instance we know the most specific classes in the ontology that describe it. Then classify class **C** in the taxonomy. Then all instances of classes that are more specific then **C** are also its instances. The only other instances for which we need to do the above check are the instances of the classes that immediate superclasses of **C** in the taxonomy.

2.1.4 Retrieval of components

Besides the individuals one can require retrieval of their components as well. A component of an instance consists of all instances connected to the given instance by some DAML+OIL property. This sort of inference is important when some of the instances in the dataset are represented by numbers or other kind of ids and the important information is given via the properties defined on these ids. Of course some restrictions on the size and content of component of instance have to be imposed in practice.

This is done in two steps. First we retrieve the instances that are described by the query class. Then

we are using user-defined patterns which other instances to be retrieved from the dataset. These patterns are given by lists of properties and chains of properties which to be followed during the retrieval.

2.1.5 Model Checking

The task can be formally defined as follows:

Given: **A** - a set of instance statements, and **O** - an ontology

Find: whether **A** is a model of **O**

Here one of the important thing is the information presented in **A**. Important assumption is that it already contains all instances for a tableau and there is no need to add new instances. Only checking for consistency of the statements with ontology and between them. This means that we will add only very limited type information to the dataset and we will check whether it is a tableau.

This task is efficiently decidable. We modified the completion rules to the restriction that no new instances can be added to the data. The only task performed here is explication of the implicit relations between instances that follow from the ontology and the dataset.

2.1.6 Minimal Sub-Ontology Extraction

The task can be formally defined as follows:

Given: **A** - a set of instance statements, and **O** - an ontology

Find: ontology **O'** - a minimal sub-ontology with a model **A**

A minimal ontology is defined as a minimal sub-taxonomy, but also such ontology will need to include some non-hierarchical knowledge (because of generalised concept inclusion axioms). In our implementation we are working with ontologies without generalised concept inclusion axioms (as it was discussed above) and we don't extract non-hierarchical knowledge. The implementation relies on the realization, which determines the most specific classes that are modeled by the dataset and then via graph traversing we extract all the connected class and property definitions.

2.2 Functional Interfaces to BOR

In the moment we have implemented the standard interfaces to description logics defined in [KRSS 1993]. This allows us to use for evaluation the benchmarks developed for evaluation of description logic systems (see next section).

At the moment we are implemented the functional interfaces described in our previous document and connect them with the SESAME implementation of RDF repositories.

3 Evaluation

A framework for evaluation of a reasoner should address at least the following criteria:

- **Correctness:** whether the supported reasoning services operate in a sound and complete fashion.
- **Efficiency:** time/space requirements of the reasoner measured with respect to “typical” (average complexity) and “extreme” (worst case complexity) data samples.
- **RDF(S) connectivity:** reasoner has to be able to work with ontologies represented as an **RDF(S)** repository in SESAME. Also it has to be able to “co-operate” with SESAME so that the later one to be able to partially benefit from the DAML+OIL semantics in its RQL queries and SAIL interface.
- **Functionality:** what services are supported.

The last two criteria were already discussed in the report and does not require a complex evaluation strategy. We have envisaged an evaluation of our reasoner comprising several levels of experiments. The correctness requirements are tested on small, human observable ontologies. The efficiency is tested over ontologies with increasing complexity.

In our work we was trying to reuse the benchmarks for evaluation of Description logic systems [DLBMark 2002] as much as possible. The DL Benchmark Suite includes tests for the following problems:

- Concept satisfiability tests
- Synthetic TBox classification tests
- Realistic TBox classification tests
- Synthetic ABox tests

In all cases there are positive and negative tests. In direct use for evaluation of our reasoner is the last problem. It is designed to check the performance of the reasoner with respect to the *realization* reasoning service. Because in our system we have also implemented terminological reasoning in order to facilitate the instance reasoning we also use problems included in the first three categories of problems.

Up to now we check our implementation with respect to *satisfiability check* with and without terminological axioms presented, *TBox classification* in both cases - realistic ontologies and generated ones.

The present results show that our implementation is correct with respect to these reasoning tasks. The efficiency is still under the performance of the state-of-the-art implementation, as far as, BOR satisfies the time requirement only the for the first three problems in each group.

Thus in our further work within the project we will target on integration, evaluation of the other reasoning services and optimizatoin of the performance.

The BOR reasoner is available for downlaod at <http://www.ontotext.com/bor> together with more instructions on its installation, tests, and usage.

4 References

- [DLBMark 2002] *The Description Logic Benchmark Suite*. With contributions from Volker Haarslev, Ian Horrocks, Ralf Möller and Peter Patel-Schneider.
<http://kogs-www.informatik.uni-hamburg.de/~moeller/dl-benchmark-suite.html>
- [Haarslev et al, 2001] Haarslev V., Mueller R., Turhan A.-Y. *Exploiting Pseudo Models for TBox and ABox Reasoning in Expressive Description Logics*. IJCAI2001
- [Hayes, 2001] Patrick Hayes. *RDF Model Theory*. W3C Working Draft.
<http://www.w3.org/TR/rdf-mt/>
- [Horrocks and Sattler, 2001] Horrocks I. and Sattler U. *Ontology Reasoning in the SHOQ(D) Description Logic*, IJCAI2001
- [KRSS 1993] *Working Version (Draft): Description Logic Specification from the KRSS Effort*. Peter F. Patel-Schneider, co-chair, Bill Swartout, co-chair. 1993.
- [Kiryakov et al, 2002] Kiryakov A., Simov K. Iv., Ognyanov D. *Ontology Middleware: Analysis and Design*. Deliverable 38, On-To-Knowledge project, March 2002.
<http://www.ontoknowledge.org/download/del38.pdf>
- [Pan and Horrocks, 2001] Pan, Jeff; Horrocks, Ian
Metamodeling Architecture of Web Ontology Languages.
In the Proc. of International Semantic Web Working Symposium (SWWS), July 30 -

August 1, 2001, Stanford University, California, USA.

- [Patel-Schneider, 2001]** Peter Patel-Schneider, editor. *A Model-Theoretic Semantics for DAML+OIL (March 2001)*. November 2001.
<http://www.daml.org/2000/12/model-theoretic-semantics.html>
- [Patel-Schneider and van Harmelen, 2001]** Patel-Schneider P., and van Harmelen Fr. *Coordination points between RDF(S) and DAML+OIL*.
<http://www.daml.org/2001/07/RDFS-DAML+OIL-coordination.html>
- [Simov, 2002]** Kiril Iv. Simov. *DAML+OIL Syntax Overview and Analysis*. Technical Report, OntoText Lab, February 2002.
<http://www.ontotext.com/publications/daml-oil-syntax.htm>
- [van Harmelen et al. 2001]** Frank van Harmelen, Peter F. Patel-Schneider and Ian Horrocks, editors. *Reference description of the DAML+OIL (March 2001) ontology markup language*. <http://www.daml.org/2001/03/reference.html>