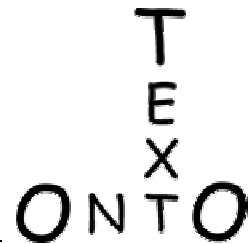


ONTOTEXT – A SIRMA AI LAB.

for Knowledge and Language Engineering

38A, CHRISTO BOTEV BLVD.
SOFIA 1000, BULGARIA
[HTTP://WWW.ONTOTEXT.COM](http://www.ontotext.com)

TEL: (359 2) 9810018, 9812338
FAX: (359 2) 9819058
E-MAIL: INFO@ONTOTEXT.COM



Ontology Middleware

<http://www.ontotext.com/omm>

System Documentation *Overview, Installation, Reference Guide*

Ver. 0.84, 04.09.2002

Atanas Kiryakov, Damyan Ognyanov, Borislav Popov



The Ontology Middleware Module subject of this document was developed as an extension of the Sesame RDF(S) repository under the On-To-Knowledge project (IST-1999-10132).



This document focuses mostly on the Ontology Middleware features. However it is just an extension of Sesame from administrator b.v. and for many issues the Sesame documentation have to be consulted at <http://sesame.administrator.nl/>



Sesame and the Ontology Middleware are open source and available for download and contributions at <http://sourceforge.net/projects/sesame/>

Abstract

Ontology Middleware Module (OMM) is an extension of the Sesame RDF(S) (<http://www.aidministrator.nl/>) repository that supports tracking changes, meta-information, fine-grained access control, and multi-protocol client access (RMI, SOAP). This document presents the implementation of OMM after the analysis and design presented in [Kiryakov et al, 2002]. It provides short overview of the principle definition of the problems as they are approached in OMM and the Knowledge Control System, which is a substantial part of it. The most part of the document provides information for the installation, sample knowledge bases, test cases and scenarios, implementation details, reference guide for the API, and performance considerations.

Acknowledgements

Big part of the Ontology Middleware presented 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), AIdministrator Nederland BV (NL), OntoText Lab. (BG). We wish to particularly thank to the Arjohn Kampman, Jeen Broekstra for the instant support and most especially for the timely implementation of the truth maintenance system of Sesame.

Availability

Most up to date version of the current document could always be found at <http://ww.ontotext.com/omm/OMMDocumentation.doc>. A PDF version, a zipped doc (.doc.zip), as well as a zipped PS (.ps.zip) are also available at the same location.

The most recent JAR files are available for download at <http://www.ontotext.com/omm/downloads/>. However, we recommend in principle such resources to be downloaded form the SorceForge site, <http://sourceforge.net/projects/sesame/>.

Contents

1. Introduction	5
1.1. Ontology Middleware Overview	5
1.2. Architecture and Interfaces.....	6
1.2.1. Overview of the SESAME Architecture	6
1.2.2. How OMM Fits in the Picture?.....	7
2. Installation	9
3. Skills Example	11
4. Major Test Cases.....	12
4.1. Working with a Previous Version	12
4.2. Play with the Security	12
4.2.1. Demo Security Setup.....	13
4.2.2. Repository Restriction Demo	14
4.2.3. Schema Restriction Demo	14
4.2.4. Classes Restriction Demo	14
4.2.5. Classes-Over-Schema Restriction Demo	15
4.2.6. Instances Restriction Demo.....	16
4.2.7. Properties Restriction Demo	17
4.2.8. Pattern Restriction Demo	17
4.2.9. Query Restriction Demo	18
5. Integration and Remote Access	19
5.1. The OMM API, Reference and Public Server.....	19
5.1.1. Correspondence with the Sesame's Java library for HTTP Clients	20
5.1.2. OMM API Reference	20
5.1.3. OMM Public Server	23
5.2. Built-in Usage of OMM and Sesame	24
5.2.1. Dependencies	24
5.2.2. Initialization	25
5.2.3. Access APIs	25
5.3. RMI Access	26
5.3.1. Client.....	26
5.4. SOAP Access.....	27
5.4.1. Server	27
5.4.2. Client.....	27

6. Implementation Details	28
6.1. Tracking Changes Implementation.....	28
6.2. Access Control Implementation	30
7. Import Demo Security Setup	31
7.1. Specify the security setup in the <code>system.conf</code> configuration file.....	32
7.2. Import security setup via <code>com.ontotext.omm.SecurityServices</code> interface.	33
8. Performance comments	35
9. References	37

1. Introduction

This document presents the implementation of the Ontology Middleware Module (OMM) – the appropriate analysis and design issues are presented in Deliverable 38, [Kiryakov et al, 2002], which is a necessary basis for proper understanding of this documentation.

A short overview of the definition and design of OMM is presented in this section followed by presentation of the architecture on a technical level. The second section provides information about the installation of OMM, the third section presents the example used for testing and evaluation purposes. The test cases and demonstration scenarios which demonstrate the most important features of OMM are presented in section 4. In sections 5,6 and 7, alternative methods for accessing Sesame are presented including standalone usage, and RMI and SOAP access.

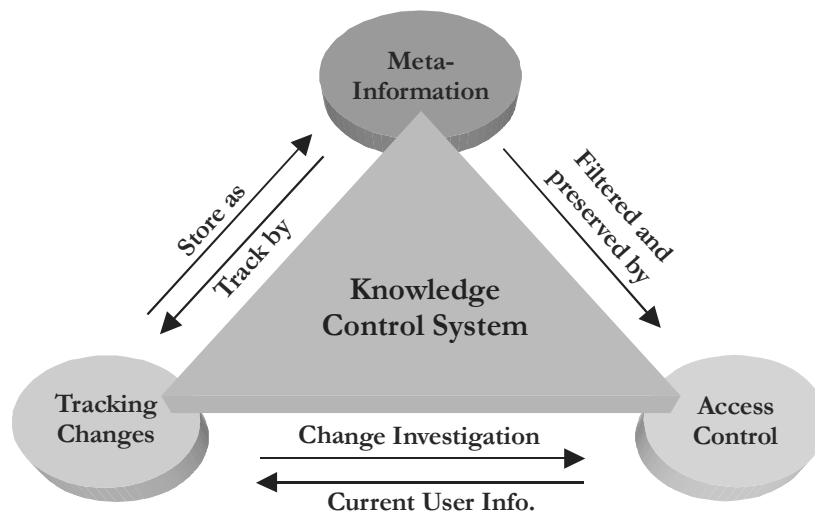
Comments on the implementation are available in section 8 followed by performance information in section 9.

1.1. *Ontology Middleware Overview*

The middleware can be seen as „administrative“ software infrastructure that makes the results of the On-To-Knowledge project easier for integration in real-world applications. The central issue is to make the methodology and modules available to the society in a shape that allows easier development, management, maintenance, and use of middle-size and big knowledge bases¹. In the light of these objectives the following main features are supported:

- Versioning (tracking changes) of knowledge bases;
- Access control (security) system;
- Meta-information for knowledge bases.

These three aspects are tightly interrelated among each other as depicted on the following scheme:



The composition of the three functions above represents a *Knowledge Control System (KCS)* that provides the knowledge engineers with the same level of control and manageability of the knowledge in the process of its development and maintenance as the source control systems (such as CVS) provide for the software. However, KCS is not only limited to support the knowledge

¹ A knowledge base can consist of ontology and/or instance data and application specific knowledge.

engineers or developers – from the perspective of the end-user applications, KCS can be seen as equivalent to the database security, change tracking (often called cataloguing) and auditing systems. The KCS is carefully designed so to support these two distinct use cases.

A fully-functional *Ontology Middleware* system should serve as a flexible and extendable platform for knowledge management solutions. It has to provide infrastructure with at least the following features:

- A repository providing the basic storage services in a scalable and reliable fashion. This role is already fulfilled by SESAME.
- Knowledge control – the KCS introduced above.
- Multi-protocol client access to allow different users and applications to use the system via the most efficient “transportation” media. This aspect is discussed in the next subsection.
- Support for pluggable reasoning modules suitable for various domains and applications. This ensures that within a single enterprise or computing environment one and the same system may be used for various purposes (that require different reasoning services and expressivity) so enabling easy integration, interoperability between applications, knowledge maintenance and reuse.

The design of the ontology middleware module presented here is just an extension of the SESAME architecture (see [Broekstra and Kampman, 2001b] and <http://sesame.aidadministrator.nl/>).

1.2. Architecture and Interfaces

Ontology Middleware Module (OMM) is designed to extend the existing functionality of the SESAME RDF(S) repository enriching it with support for versioning (tracking changes), meta-information, access control (security), and Description Logic (DL) reasoning.

Here we present an overview of the enhancements to Sesame – how the architecture was extended, which are the new and the modified modules.

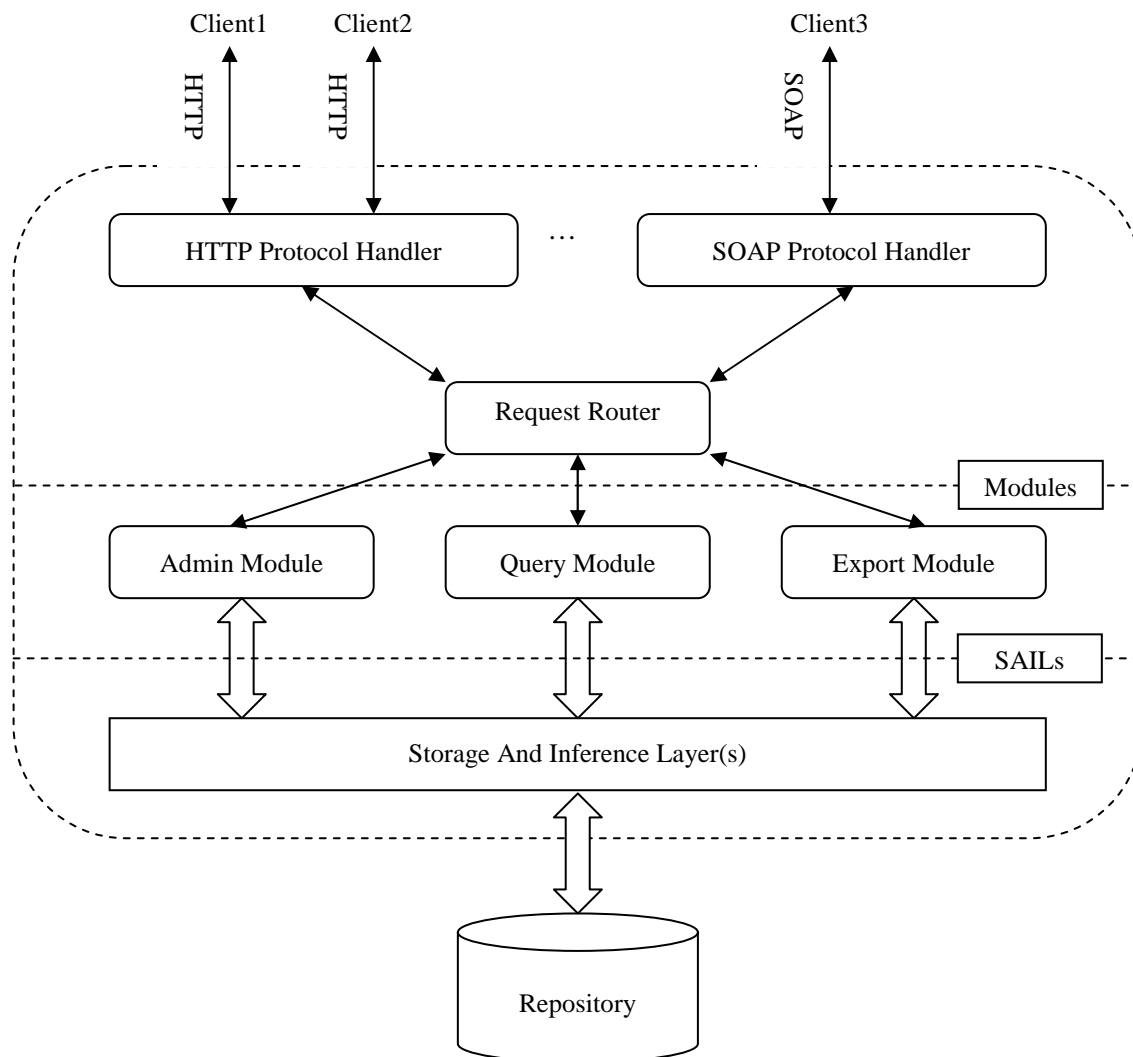
1.2.1. Overview of the SESAME Architecture

The SESAME architecture is composed of several layers. The access layer is responsible to provide the access functionality to be used by the front-end applications.

The Request Router manages the necessary marshalling of the calls from and to the appropriate protocols and routes the calls to the modules that provide the appropriate handling functionality – it was extended to meet the multi-protocol access requirements of OMM as well as to provide session handling in a manner desired by the versioning and security features of OMM.

A layer follows, consisting of various modules, each of which implements the basic functionality via calls to the storage and inference layers (SAIL).

The SAIL interface hides the specific implementation dependant to the underlying physical storage, features and formal semantics supported. A number of SAILS can be stacked on top of each other, each of them handling some of calls and transmitting the rest to the underlying one. The access to the SAILS as well as the communication between them goes through the SAIL interfaces. More about the SESAME architecture and features can be found in [Broekstra and Kampman, 2001b] and <http://sesame.aidadministrator.nl/>.



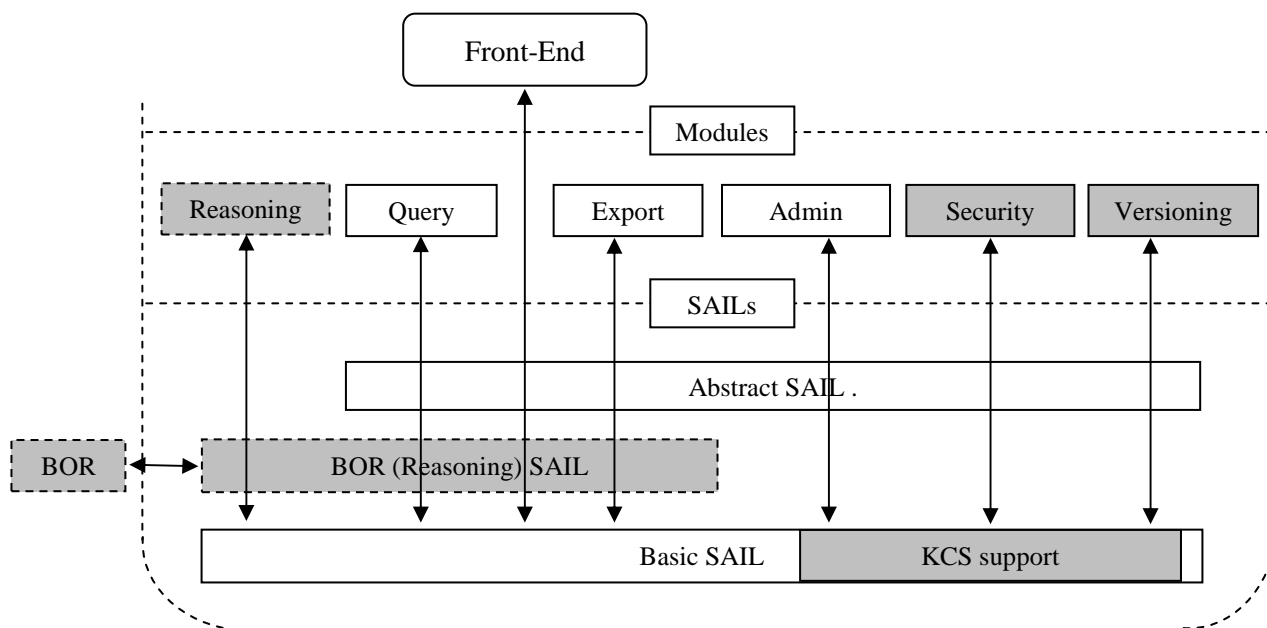
1.2.2. How OMM Fits in the Picture?

The interfaces of the OMM are designed to fit in the stacked SAIL architecture by extending it with their specific functionality. Some of the SESAME functional modules were modified so to benefit directly from the interfaces in the OMM SAIL extension. The implementation of BOR (the DAML+OIL reasoner, [Simov and Jordanov, 2002]) was integrated through an additional SAIL. Its full potential is accessible via a separate Reasoning module responsible for routing the appropriate calls to the reasoner. On the other hand the reasoner supports the SAIL interface so the rest of the modules can also interoperate with it like with the standard RDF(S) supporting SAIL. For instance, the Query module will be able to perform queries against DAML+OIL repositories without changes to its interfaces.

It is also the case that the existing basic SAIL (that implements the physical storage and the RDF(S) reasoning on top of a relational database) have been modified so to take care at the lowest level for some of the kinds of meta-information, especially those necessary for the knowledge control system (KCS). The reason for this design is that it is extremely important the information related to the access rights and “history” of the statements in the repository to be stored in the most efficient way.

This architecture allows transparent manipulation of the repository for the existing tools. Each application can either work with the repository through the existing SESAME modules or by changing the repository behavior working directly with the new reasoning, versioning, and security modules or gaining the access to the underlying low-level programming interfaces.

The scheme below represents how the lower part of the current SESAME architecture was changed. As mentioned earlier, the Request Router also undertook some modifications so to be able to (i) route the new interfaces and (ii) support more protocols. There were also important changes that take place in the router in order to enable the most convenient access control mechanism for each of the specific protocols.



The gray boxes denote the new modules or extensions developed. The arrows depict how the requests of the different modules are passing through the different SAILs. The boxes with dashed borders (Reasoning and BOR) are optional, i.e. not an obligatory part of the OMM/SESAME architecture. Those are only necessary for DAML+OIL reasoning.

Another new feature is that front-end applications (such as editors and viewers) are now able to communicate directly with the SAILs without using any of the functional modules. The requests of such application (as all the requests coming outside SESAME) are handled through the request router. It also means that such requests are subject of the standard access control and they are possible through all the supported protocols. This feature is still under development and testing.

2. Installation

The Ontology Middleware Module (OMM) is an open-source extension of Sesame. It is available together with its source code, examples, and documentation from the SourceForge at <http://sourceforge.net/projects/sesame/> and the Sesame page at <http://sesame.aidadministrator.nl/>.

Most up to date instruction as well as additional information is available at

<http://ww.ontotext.com/omm/OMMInstall.html>.

The most recent stable versions of all JAR and other files necessary for OMM can be found at:

<http://ww.ontotext.com/omm/download.html>.

The Ontology Middleware Module can be easily installed on top of a SESAME installation working with MySQL database (SQL92SAIL). Here are the major steps during the installation:

- Get a version of `sesame.jar` that contains all the `com.ontotext.omm` classes. Alternatively, one may just add these class files to an existing `sesame.jar`. The later one usually resides in folder `<TOMCAT_HOME>/webapps/sesame/WEB-INF`.
- Extend the `web.xml` file (again in `<TOMCAT_HOME>/webapps/sesame/WEB-INF`) with registrations and mappings for two more servlets: `createVersion` and `selectBranch`. This requires the following text to be added for the first one:

```
<servlet>
  <servlet-name>createVersion</servlet-name>
  <servlet-class>com.ontotext.omm.http.CreateVersion
  </servlet-class>
  ...
</servlet>
```

and

```
<servlet-mapping>
  <servlet-name>createVersion</servlet-name>
  <url-pattern>/servlets/createVersion/*</url-pattern>
</servlet-mapping>
```

Similar sections are also necessary for the `selectBranch` servlet.

- Create a new repository in the `system.conf` file (the same directory), with the following `sail` elements in the `sailstack`:

```
<sail class="com.ontotext.omm.security.SecuritySail">
  <param name="jdbcDriver" value="org.gjt.mm.mysql.Driver"/>
  ...
</sail>
<sail class="com.ontotext.omm.versioning.VersioningTmsMySQLSail">
  <param name="jdbcDriver" value="org.gjt.mm.mysql.Driver"/>
  <param name="jdbcDriver" value="org.gjt.mm.mysql.Driver"/>
  ...
</sail>
```

Setting up the JDBC parameters according to your database installation (should the same as for the `SQL92SAIL`).

Having the installation finished, OMM services can be configured on a per-repository basis – within single installation there could be one repository with access control, another just with tracking and third one using both or none of this services.

Making use of OMM depends on the exact feature. For instance, for a repository which was set up to use the `VersioningTmsMySQLSail` the changes will be tracked without any action on the user or application side. As another example, for repositories using the `SecuritySail`, the security policy²

² Or just the default security policy if there is no such defined.

will be enforced, again without any change in the APIs – every regular request to OMM/SESAME will be controlled by the security sail. For instance, in an RQL query, only those statements that fit into the users permissions will be returned as a result.

3. Skills Example

For the purpose of testing and verification of OMM we extended the skills management schema from the case study presented in Deliverable 20, [Novotny and Lau, 2001]. It is a relatively simple Skills Management knowledge base structured as follows:

- **Enterprise.rdfs** – a general organizational ontology, covering the basic types of organizations, various employment and management relations, as well, as structural relations in complex organizations. No ownership issues modeled. This model is derived from the Upper Cyc Ontology, [Cycorp 1997]. It includes classes such as `Organization` and `Position` and properties such as `hasPosition`;
- **Skills.rdfs** – a general ontology about personal skills, training courses, and competence requirements for certain positions, expressed via the required skills. Includes classes such as `Skill` and properties such as `hasSkill`;
- **Sirma_enter_kb.rdf** – description of the companies, people and positions within Sirma Group. The knowledge is simplified but correct view of a real organization;
- **Sirma_skills_hierarchy.rdfs** – non-exhaustive hierarchy of skills relevant for Sirma Group. The top skills are `BusinessSkill`, `TechnicalSkill`, and `LanguageSkill`;
- **Sirma_skills_kb.rdf** – information about the skills of specific people within Sirma Group.

Overall, the sample knowledge base has about 140 classes and 600-700 statements. The RDF(S) files are available at <http://www.ontotext.com/otk/2002/05/>. It is also uploaded in the "Sirma Skills KB" repository (with ID "mysql-sskb") of the public server, as presented in sub-section 5.1.3.

In order to be able to reproduce the test cases, the files should be uploaded in this order in a clear repository with base URIs like `http://www.ontotext.com/otk/2002/05/<file_name>`. Both `VersioningTmsMySQLSail` and `SecuritySail` should be included in the stack. The user account used should have the appropriate rights – in order to avoid mistakes and confusion, we recommend the default administrator account (admin/admin) to be used for the upload.

4. Major Test Cases

Test cases based on the *Skills example* from the previous section are presented here, demonstrating the key features of the Ontology Middleware Module.

One could, either install and run its own installation of Sesame, either use it online through web interface at <http://omm.ontotext.com/>, choosing “Sirma Skills KB” repository with security and tracking of changes.

The steps in each of the case descriptions presuppose that the user has been logged into the SESAME system, often with requirements for the user account being used. The most basic user interface to SESAME currently is a web-interface that can be started just putting a URL such as `http://your_server:a_port/Sesame` - the exact details depend on the SESAME and Tomcat installations. Once getting to the entry page of SESAME, one could log in as a specific user if the [Log In](#) link is followed.

4.1. Working with a Previous Version

After loading the sample into the database, one can check how the uploading transactions were tracked. The first six updates (UIDs 2 to 7) correspond to the upload of each of the files. In order to see one of them it has to be first labeled as a version (this is due to the specifics of the current GUI – there is no such requirement in principle.)

1. Select **Set Working Version** from the **Available Actions** page. In the **Update ID** combo-box, you see the valid Update Ids – this is, actually, the list of the states of the repository.
2. Select an **update ID** (say 2), put a version name in the **Label** field (say “ver2”), and press **create**. In case of successful labeling/creation of a version you will be sent back to the **Available Actions** page.
3. Select **Set Working Version** again. The new version should appear in the **Repository Versions** list.
4. Follow the link for the newly created version, say “ver2”. In this moment, a dummy branch of the current repository is being created and populated with the state of the repository at the selected version. This process depending on the size of the repository can take few seconds or a minute. You are automatically getting switched to work with the newly-created branch in **read-only mode**. After successful branching, you are sent to page, where you can see text like “Available actions on **branch of 2 repos_name** [select other] are:...”. This is an indication that you are currently working with the branch of your initially selected repository, populated with its state at the selected version – any requests and queries will be interpreted with respect to this state.
5. From the **Available Actions** page, select **Evaluate an RQL Query**, and evaluate the “class” query – the classes introduced after this update (after the upload of the corresponding file) will not appear in the result. For instance, if you set as a working version, Update ID 2, this means that all the specific skill classes (such as `http://.../sirma_skills_hier.rdfs#Java`) should be missing.

4.2. Play with the Security

This set of demos concerns the fine-grained Security and Access Control functionality over RDF(S) repositories. Its aim is to present (via reasonable examples and explanations) the types of security restrictions covered by the current implementation. In order to obtain better understanding

on the formal representation of the security policy, one should read section 3 in [Kiryakov et al, 2002].

4.2.1. Demo Security Setup

This section describes the *security setup* used in this demo, by presenting the permissions given to each user.

Creating a security setup in RDF(S) and importing it is explained in [section 8](#).

The **Admin** is granted full permissions (*read, add, remove, admin, history*) over the whole repository via a *repository restriction*, which is used in a *do-everything-with-the-repository rule*.

The **Anonymous** user is not given any permission, which means - unable to perform any of the access actions, even *read*.

The **testuser** is allowed to read the entire *schema* via a *Schema Restriction* used in a rule, specifying *Read* right.

The **HRManager** is allowed to *add, read, remove* all the subclasses of *Skill*. This part of the repository is restricted via a *Classes Over Schema Restriction*. *Class Restriction* is used to allow the user to *read, add, remove* all instances of Skills.

The **RandDManager** has the *read, add, remove* rights for the *TechnicalSkill* hierarchy and its instances. These permissions are set in the same manner as for the *HRManager* : via *Class Restriction* and a *Class Over Schema Restriction* used in security rule.

The **Employee** has the most complex security setup in the demo. Its permissions are set via three *Pattern Restrictions* formed by three *Classes Restrictions*, three *Properties Restrictions* and one *Classes Over Schema Restriction*. The security rules formed on the basis of the *Pattern Restrictions* grant *Read* right to the *Employee*.

- Pattern restriction $\langle iPerson, pHasPosition, _ \rangle$, where: *iPerson* is a *Classes Restriction* over the instances of *Person*. *pHasPosition* is a *Properties Restriction* over the sub-properties of *hasPosition*. This restriction specifies the triples that state the position which a particular person has.
- Pattern restriction $\langle scSkill, pSubClassOf, _ \rangle$, where: *scSkill* is a *Classes Over Schema Restriction* over the subclasses of *Skill*; *pSubClassOf* is a *Properties Restriction* over the sub-properties of *subClassOf*. This restriction specifies all triples that define the Skills hierarchy.
- Pattern restriction $\langle iPerson, pHasSkill, iTechanicalSkill \rangle$, where: *iPerson* is a *Classes Restriction* over the instances of *Person*; *pHasSkill* is a *Properties Restriction* over the sub-properties of *hasSkill*; and *iTechnicalSkill* is a *Classes Restriction* over the instances of *TechnicalSkill*. This restriction specifies all triples that state the belonging of a certain technical skill to a particular person.

The **QueryTester**'s permissions are defined via a *Query Restriction*. The following query is used :
*select * from*

{X} http://www.ontotext.com/otk/2002/05/enterprise.rdfs#hasPosition {Y}

The restriction is associated with a security rule and *Read* permissions are granted to the user. Thus all the statements, where the subject is one of the query results, are considered accessible.

The user **Dimitar** is used to demonstrate *Instances Restrictions*. The resource :

http://www.ontotext.com/otk/2002/05/sirma_enter_kb.rdf#DimitarManov

is assigned to the *Instances Restriction*. The restriction is used to construct a security rule which enables *Reading* of the statements where the specified resource is subject.

4.2.2. Repository Restriction Demo

Restricts the whole repository. Certain rights are applicable only for this restriction type (such as Admin and History).

This section will present the access to the repository when there is (or isn't) present a repository restriction for the user being used. The **admin** user has a **do-everything-with-the-repository** rule associated. This rule is based on a **repository restriction**. The **anonymous** user does not have assigned a rule based on repository restriction, neither any other rule.

1. Log in³ to Sesame as [**admin** | **admin**].
2. Choose **Evaluate an RQL Query**.
3. Type in the query : **select * from {X} @p {Y}**. All the triples in the repository are displayed.
4. Log out.
5. Log in as [**anonymous** |] .
6. Choose **Evaluate an RQL Query**.
7. Type in the query : **select * from {X} @p {Y}**. No triples are displayed.
8. Log out.

4.2.3. Schema Restriction Demo

Restricts all the resources and statements that constitute the schema of the repository.

The **testuser** does have a rule that allows *reading* and is restricted by a schema restriction. On the other hand, the **anonymous** is not allowed to view the schema.

1. Log in to Sesame as [**testuser** | **opensesame**].
2. Choose **Evaluate an RQL Query**.
3. Type in the query : **select * from {X} @p {Y}**. All the triples forming **the schema** are displayed (e.g. 476 results found, while the whole repository consists of 608 triples.)
4. Log out.
5. Log in as [**anonymous** |] .
6. Choose **Evaluate an RQL Query**.
7. Type in the query : **select * from {X} @p {Y}**. No triples are displayed.
8. Log out.

4.2.4. Classes Restriction Demo

Restricts all the resources (instances) of specific classes, including the statements where those are subjects. Disjunction logic applicable in case of multiple classes. Resources that are instances of sub-classes also considered. Defined via set of classes.

This example demonstrates that the **HRManager** is able to *add, read and remove instances* under the *Skills* hierarchy (Instances of <http://www.ontotext.com/otk/2002/05/skills.rdfs#Skill> and its subclasses). While the **Employee** has permissions assigned, as explained in the [Demo Security Setup](#) section.

For this purpose a new **DemoSwingSkill** class will be added as an instance of the **Swing** skill. It will be associated with an existing employee and the level of this skill for this employee will be

³ Each time “Log in” is suggested, please also select the database with the Skills Hierarchy Example loaded and with the Security Sail set for it as one of the stacked sails.

specified (<http://www.ontotext.com/otk/2002/05/SecurityDemo/newSwingSkill.rdf>).

A. Add

1. Log in as [**Employee | E**].
2. Choose *Add data from the world wide web*.
3. As a *URL of the data set*:
<http://www.ontotext.com/otk/2002/05/SecurityDemo/newSwingSkill.rdf>. Press **Add Data**.
The status should display that the addition of statements is impossible since access is denied. To check this:
4. Go back and choose *Explore the Repository*.
5. Paste this URI: http://www.ontotext.com/otk/2002/05/sirma_enter_kb.rdf#DemoSwingSkill and select *Explore*. No results should be displayed.
6. Log out.
7. Log in as [**HRManager | HR**].
8. Choose **Add data from the world wide web**.
9. As a *URL of the data set* :
<http://www.ontotext.com/otk/2002/05/SecurityDemo/newSwingSkill.rdf>
10. Press **Add Data**.
11. The status should display the number of added statements.
12. In order to list all the triples which subject is *DemoSwinSkill* go back and select *Evaluate and RQL Query*. Paste the following query:
13. `select * from {S} @p {O}`
14. where S = http://www.ontotext.com/otk/2002/05/sirma_enter_kb.rdf#DemoSwingSkill
15. Three (3) results should be displayed, where one of the triples states that *DemoSwingSkill* is instances (rdf:type) of the *Swing* skill.
16. Log out.

B. Remove

1. Log in as [**Employee | E**].
2. Choose **Remove statements**. We will try to remove the triple that defines *DemoSwingSkill* as instance of the *Swing* skill.
3. Specify *subject, predicate, object* as follows:
subject : http://www.ontotext.com/otk/2002/05/sirma_enter_kb.rdf#DemoSwingSkill
predicate : <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
object : http://www.ontotext.com/otk/2002/05/sirma_skills_hier.rdfs#Swing
4. Proceed by pressing the *Remove statements* button. The notification should say that *no statements have been removed* since the *Employee* has no permission to remove instances under the *Skills Hierarchy*.
5. Now the *remove attempt* will be held under the *HRManager* login.
6. Log in as [**HRManager | HR**].
7. Choose **Remove statements**. We'll try to remove the triple that defines *DemoSwingSkill* as instance of the *Swing* skill.
8. Specify only the *subject* :
http://www.ontotext.com/otk/2002/05/sirma_enter_kb.rdf#DemoSwingSkill
9. Proceed by pressing the *Remove statements* button. The notification should say that the statement (one statement) *has been removed*.

Note: Using **RandDManager** instead of **HRManager** and **TechnicalSkill** instead of **Skill** could additionally demonstrate the scenario demonstrated in this section.

4.2.5. Classes-Over-Schema Restriction Demo

Restricts all the classes and properties that are sub (class/property) of the resource stated in the restriction's definition. Is a restriction over the schema.

This section presents the granted permission of the HRManager to edit the Skills Hierarchy via a security rule using Classes Over Schema Restriction.

A. Add

1. Log in as [**Employee|E**].
2. Choose *Add data from the world wide web* and set the following URL : <http://www.ontotext.com/otk/2002/05/SecurityDemo/skillClass.rdfs> - it contains a new subclass of *BusinessSkill* - *BusinessEspionage*. The employee should not be able to add such a statement.
3. Press *Add Data*.
4. Go back and choose *Evaluate an RQL query*.
5. Paste the following query:

```
select * from {X} @p {Y}
where X = http://www.ontotext.com/otk/2002/05/sirma_skills_hier.rdfs#BusinessEspionage
```

No results should be found which indicates that indeed the statements have *not been added*.
6. Log out.
7. Log in as [**HRManager|HR**].
8. Repeat steps 2-5 above, three statements will be returned after step 5.

B. Remove

1. Log in as [**Employee|E**].
2. Choose *Remove Statements* and Set the *subject* to:
http://www.ontotext.com/otk/2002/05/sirma_skills_hier.rdfs#BusinessEspionage. This should remove all statements which subject is *BusinessEspionage*.
3. Press *Remove Statements*. The status should report that *no statements have been removed*.
4. Log out.
5. Log in as [**HRManager|HR**].
6. Repeat steps 2 and 3. After step 3 the status should report that 3 statements have been removed, which demonstrates the *remove* permissions granted to the *HRManager* under the *Skills Hierarchy*.
7. Log out.

Note: Using **RandDManager** instead of **HRManager** and **TechnicalSkill** instead of **Skill** could additionally demonstrate the scenario demonstrated in this section.

4.2.6. Instances Restriction Demo

Restricts a set of specific resources, including the statements where those are subjects. Is defined via a set of resources.

The *Instances Restrictions* are presented via the user *Dimitar*.

1. Log in as [**Dimitar|mitac**].
2. Choose *Explore the Repository*.
3. Set the following URI:

http://www.ontotext.com/otk/2002/05/sirma_enter_kb.rdf#DimitarManov

4. Press *Explore*.

All the statements having the specified resource as a subject are displayed.

5. Click on :

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>

from the result set. Only statements that are with subject :
http://www.ontotext.com/otk/2002/05/sirma_enter_kb.rdf#DimitarManov
 should be returned.

6. Log out.

4.2.7. *Properties Restriction Demo*

Restricts all the statements with specific properties as predicates. Disjunction logic applicable in case of multiple properties specified. The sub-properties are also considered. Defined via a set of properties.

The *Properties Restrictions* are demonstrated in the following section as part of the *Pattern Restrictions* demo, since each of the presented *Pattern Restrictions* has a *Properties Restriction* as a constituent.

4.2.8. *Pattern Restriction Demo*

Restricts all the statements that conform to patterns defined via restrictions of type Classes or Instances over the subject and/or object and restriction of type Properties over the predicate;

The **Employee** user will be used to demonstrate its permissions defined via three Pattern Restrictions.

< iPerson, pHasPosition, >

1. Log in as [**Employee|E**].
2. Choose *Explore the Repository*.
3. Set the following URI: <http://www.ontotext.com/otk/2002/05/enterprise.rdfs#hasPosition> and Press *Explore*. Only the 5 statements that comply with this restriction should be displayed, the others are filtered.
4. Log out.

< scSkill, pSubClassOf, >

1. Log in as [**Employee|E**].
2. Choose *Explore the Repository*
3. Set the following URI: <http://www.w3.org/2000/01/rdf-schema#subClassOf> and Press *Explore*. Only the triples from the Skills hierarchy definition are displayed having subClassOf as predicate, the others are filtered. Compare via the same query using the admin user (follows).
4. Log out.
5. Log in as [**admin|admin**].
6. Repeat steps 2 and 3, now it displays the triples that contain as a constituent *subClassOf*, either as subject, predicate or object.
7. Log out.

< iPerson, pHasSkill, iTechanicalSkill >

1. Log in as [**Employee|E**].
2. Choose *Explore the Repository*
3. Set the following URI: <http://www.ontotext.com/otk/2002/05/skills.rdfs#hasSkill>. Only the triples with TechnicalSkill as an object are displayed, the others are filtered. Compare via the same query using the admin user (follows).
4. Log out.
5. Log in as [**admin|admin**].
6. Repeat steps 2 and 3, now it displays all the triples having *hasSkill* as predicate. The result

should contain at least one more triple than the previous result, because there is at least one `aPerson,hasSkill, aSkill` triple, in which `aSkill` is not a *TechnicalSkill*, but is a *LanguageSkill*, for example.

7. Log out.

4.2.9. Query Restriction Demo

Restricts the set of statements to be returned by an RQL query that could take the current user as a parameter.

The *QueryTester* user is used to demonstrate the usage of *Query Restrictions*.

1. Log in as [**QueryTester|qu**].
2. Choose *Explore the Repository*.
3. Set the URI to: `http://www.ontotext.com/otk/2002/05/enterprise.rdfs#hasPosition`
4. Press *Explore*.

Only the triples with subjects from the query result are returned. These are the statements that specify the positions assigned to a person.

5. Log out.

Repeating the same steps for the *Admin* user demonstrates the differences.

6. Log in as [**admin|admin**].
7. Choose *Explore the Repository*.
8. Set the URI to : `http://www.ontotext.com/otk/2002/05/enterprise.rdfs#hasPosition`
9. Press *Explore*.

All the triples with subject, object or predicate from the query result are returned.

10. Log out.

.

5. Integration and Remote Access

One of the important features that OMM adds to Sesame is the multi-protocol client access. At present the following options for integration of and remote access to OMM/Sesame are supported:

- HTTP requests – this option is the one originally supported by Sesame. It lies underneath the standard web-interface to Sesame. Any applications capable in performing HTTP requests can use this option, according to the request specification presented in <http://sesame.aidadministrator.nl/publications/communication.html>. Further, the Java applications can make use of the client library wrapping the HTTP requests within Java classes and methods. Documentation of this library can be found at: <http://sesame.aidadministrator.nl/publications/api/client/>. The HTTP access is further discussed within this document;
- Built-in use – Sesame can be directly built (or embedded) into Java applications. In this case there is no remote access protocol involved, both Sesame and the application using it work on one and the same machine within one and the same process. Obviously, this scenario for Sesame use is the most efficient one as far as there are no communication overheads. The Built-in usage of Sesame is addressed in a sub-section below;
- RMI calls – Sesame can also be accessed from Java applications via the RMI protocol. In contrast to HTTP, RMI is a well-developed RPC protocol. The communication overhead is much lower compared to HTTP or SOAP, but it is only feasible for Java applications. It is discussed in detail in a sub-section below;
- SOAP messages – Sesame can be used as a SOAP service from any platforms and programming languages. SOAP is an XML based RPC protocol that can work on top of HTTP. It has the advantages of both HTTP (being cross-platform) and RMI (being a true RPC protocol) combined with wide industry support. It has to be considered that the communication overhead for SOAP is even bigger than for HTTP. The SOAP access to Sesame is discussed in the last sub-section here.

5.1. The OMM API, Reference and Public Server

The OMM API is a combination of three groups of methods:

- Standard Sesame interfaces – those which are available in Sesame via HTTP, namely: `addDataFromUrl(...)`, `clearRepository(...)`, `evalRdqlQuery(...)`, `evalRqlQuery(...)`, `extractRDF(...)`, `removeStatements(...)`, `selectRepository(...)`, and `uploadData(...)`;
- SAIL methods – methods from the SAIL API that provide lower level access to the repositories. Those are methods like `getSubClassesOf(...)`, `isProperty(...)`, etc. This methods provide more direct access to the repositories with respect to its basic RDF(S) semantic and graph representation, without dependence on any query languages. See section 1.2. for better understanding of the SAIL role or the Sesame documentation for reference;
- OMM specific methods – such related to versioning and security sub-systems, tracking changes' meta information, and other aspects of the knowledge control system (KCS). For instance, `branchState(...)`, `revertToState(...)`, `getMetaInfo(...)`. Those methods are presented in section 5 of [\[Kiryakov et al. 2002\]](#).

The OMM API is currently partially supported. There is support implemented for all the standard Sesame interfaces, part of the SAIL methods, and part of the OMM-specific methods. The standard interfaces are supported for all integration options. The Built-In use option provides easy access to all the SAIL methods – however, only part of those are available through RMI and SOAP, and non

of them via HTTP.

5.1.1. *Correspondence with the Sesame's Java library for HTTP Clients*

There are two major differences between the Java Client Library for HTTP (JCLH) access to Sesame developed by Administrator, and the OMM API:

- JCLH covers only the standard Sesame interfaces (the first group of methods, mentioned above);
- JCLH is available only in Java, and only for HTTP. In contrast, the OMM API is designed and implemented so to be efficient across different platforms and protocols.

Currently there are differences between the Client Library and the OMM API even where they overlap and can be aligned. The methods in OMM are equivalent to those in the Client Library, as far as both comply with the standard HTTP interface to Sesame. However, there are some differences in the parameters and result types and structures. Here the major differences are discussed:

- The OMM API is much more session-oriented than the JCLH and thus it counts more on the session context. As far as each session works on behalf of a single user with single repository in each moment, this information is kept in the session context rather than passed as parameters during each call. So, most of the JCLH methods take the repository ID as a parameter, while the OMM methods do not.
- The JCHL methods are implemented so to allow more information about the request execution, error messages, and transactions – information available through the `AdminListener` interface. In contrast, the OMM API is using fairly simple error codes, which make the communication faster and simpler, but provide less control and information.
- The JCHL is designed to allow streaming the results of query execution –feature extremely important for queries which execution takes a lot of time as well as in cases when the data processing is organized in a pipeline. In contrast, OMM is designed so to support in the most efficient way possible “normal” queries, returning reasonable amount of data within reasonable time. For instance, in JCLH, the RQL query execution can be performed via:

```
QueryResultsTable evalRqlQuery(String query, String repository)
```

while in the OMM API it comes like:

```
String[] [] evalRqlQuery(String query)
```

OntoText Lab, the developer of OMM, will keep working on better alignment of the two interfaces, which however will be subject of extensive consulting and co-ordination with administrator b.v., the developers of Sesame and the JCHL. Ideally, at the end, both JCHL and the OMM API should be as much compatible as possible, providing all the benefits of the different design approaches, so, to allow the application programmer to choose the proper method or structure for the specific task and application.

5.1.2. *OMM API Reference*

Here follows list of the methods currently supported for Built-in use, RMI and SOAP access. For some of the protocols, some of the methods may have slightly different parameters and return types, please, refer to the documentation for the specific protocol.

The methods are given in a JavaDoc-like notation in alphabetical order. Those, which are part of the standard Sesame interface are marked with `STANDARD`, those coming from the SAIL interface are marked with `SAIL`, those related to the OMM (actually the, Knowledge Control System), with `KCS`, and finally there are some auxiliary methods marked with `AUX`.

WARNING! The KCS methods related to tracking of changes are only available for repositories with tracking changes support. If, in addition, the repository is subject of access control, only the users with `history` permissions can successfully execute these methods. In all other cases the execution will fail or return no result.

Method Summary	
<code>int</code>	<u><code>addDataFromUrl</code></u> (String dataUrl, String baseUrl) Adds RDF(s) given URL of the RDF(s) document and a base URL. Returns the number of the statements added. STANDARD
<code>String</code>	<u><code>branchState</code></u> (long stateUID) Branch the repository at given state for further operations, returns the ID of the branched repository. KCS
<code>boolean</code>	<u><code>clearRepository</code></u> () Clears the repository. STANDARD
<code>void</code>	<u><code>continueCounterIncrement</code></u> () Continues with the normal increment of the update counter on each modification made in the repository. See <code>pauseCounterIncrement()</code> . KCS
<code>String[] []</code>	<u><code>evalRdqlQuery</code></u> (String query) Evaluates an RDQL query and returns the result. STANDARD
<code>String[] []</code>	<u><code>evalRqlQuery</code></u> (String query) Evaluates an RQL query and returns the result. STANDARD
<code>String</code>	<u><code>extractRDF</code></u> (boolean ontology, boolean instances, boolean explicit) Extracts the ontology and/or instances from the repository. STANDARD
<code>Vector</code>	<u><code>getClasses</code></u> () Retrieve the list of all the classes defined in the repository. SAIL
<code>Vector</code>	<u><code>getClassesOf</code></u> (String anInstance, boolean mostSpecific) Retrieve the list of classes to which <code>anInstance</code> belongs. According to the flag <code>moreSpecific</code> flag, the most specific ones can only be retrieved. SAIL
<code>Vector</code>	<u><code>getInstancesOf</code></u> (String aClass, boolean proper) Retrieve a list of URI's of an instances of a specific class. If the <code>proper</code> flag is set to <code>true</code> , the instances of its sub-classes are not retrieved. SAIL
<code>Map</code>	<u><code>getMetaInfo</code></u> (String subj, String pred, String obj) Retrieves the meta-info for a given statement. The statement is specified by the URIs of it's subject, predicate and object. The resulting map consists of keys mapped to the actual meta-info values. Generally, the keys are URIs from the <u>KCS schema</u> . E.g. meta-info key-value pair stating that a particular statement has vanished from the repository at a specific update (id): key : <code>http://www.ontotext.com/otk/2002/03/kcs.rdfs#bornAt</code> value : 3 KCS

Vector	<u>getProperties</u> () Retrieve the list of all the properties defined in the repository. SAIL
Vector	<u>getStatements</u> (String subj, String pred, String obj, boolean explicitOnly, boolean objIsLiteral) Retrieve a list of the statements from the repository, according to the pat. SAIL
Vector	<u>getSubClassesOf</u> (String resource, boolean direct) Retrieve the list of the sub-classes of a class. SAIL
Vector	<u>getSubPropertiesOf</u> (String resource, boolean direct) Retrieve the list of the sub-properties of a property. SAIL
Vector	<u>getSuperClassesOf</u> (String resource, boolean direct) Retrieve the list of the super-classes of a class. SAIL
Vector	<u>getSuperPropertiesOf</u> (String resource, boolean direct) Retrieve the list of the super-properties of a property. SAIL
long []	<u>getUpdateIds</u> () Retrieve the list of the ids of all Updates of the repository, sorted in chronological order. The latest, i.e. the current, state is last. KCS
Map	<u>getUpdateMetaInfo</u> (String updateId) Retrieves the meta-info for a given update, specified by its id. The resulting map consists of keys (URIs from the <u>KCS schema</u>) mapped to the actual meta-info values. KCS
Vector	<u>getVersionIds</u> () Retrieves a Vector over the version ids of all the versions of the repository. KCS
Map	<u>getVersionMetaInfo</u> (String versionId) Retrieves the meta-info for a given version, specified by its id. The resulting map consists of keys (URIs from the <u>KCS schema</u>) mapped to the actual meta-info values. KCS
boolean	<u>hasStatement</u> (String subj, String pred, String obj, boolean explicitOnly, boolean objIsLiteral) Query the repository for particular statement. SAIL
boolean	<u>isClass</u> (String resource) Check that a URI is a class. STANDARD
boolean	<u>isInstanceOf</u> (String anInstance, String aClass, boolean proper) Check that a URI is an instance of a class. SAIL
boolean	<u>isPausedCounterIncrement</u> () Check if the update counter is paused. KCS
boolean	<u>isProperty</u> (String resource) Check that a URI is a property. SAIL
boolean	<u>isSubClassOf</u> (String subClass, String superClass, boolean direct)

	Query for subsumption of two classes. SAIL
boolean	<u>isSubPropertyOf</u> (String subProperty, String superProperty, boolean direct) Query for subsumption of two properties. SAIL
void	<u>labelCurrentState</u> (String label) Create a labeled version of the current repository state. KCS
void	<u>labelState</u> (long stateUID, String label) Create a labeled version for a state of the repository assigning the necessary meta-information. KCS
String[]	<u>listRepositories</u> () Lists all the repositories. KCS
boolean	<u>login</u> (String userID, String pass) Logs in given a user and a password. AUX
void	<u>pauseCounterIncrement</u> () Stop the increment of the update counter, this way multiple updates will appear to be done “simultaneously”. KCS
int	<u>removeStatements</u> (String subjURI, String predURI, String objURI, boolean bObjectIsLiteral) Removes statements corresponding to the <subject, predicate, object> pattern. Returns the number of deleted statements. The appropriate implicit statements are also removed, i.e. those inferred from and “supported” only by the statements being removed with this call. STANDARD
Void	<u>revertToState</u> (long stateUID) Restore the repository to previous state removing all statements added after the value of the update counter and revive all remover ones. KCS
boolean	<u>selectRepository</u> (String repository) Selects a repository to work with. AUX
int	<u>uploadData</u> (String data, String baseURL) Uploads RDF(s) in Sesame. Returns the number of statements added. STANDARD
void	<u>workWithState</u> (long stateUID) Switch the repository to a given previous state for the consequent read operations. The state is identified by update ID (UID) – like those returned from getUpdateIDs() method. This operation could be slow for big repositories, because an auxiliary repository (say, AUXR1), which is a *read-only* branch of the current one (say R1), is being created and automatically selected. To start working again with the current state, select the R1 repository via selectRepository() . KCS

5.1.3. OMM Public Server

OntoText Lab. supports a public OMM/Sesame server, which can be used for evaluation and research purposes. It can be accessed via the standard web-interface (or programmatically with HTTP requests) directly at <http://omm.ontotext.com>. This address is being currently redirected to <http://62.213.161.156:8888/sesame> - this address can change through the time, but on the

hand its use will be faster and more reliable. SOAP and RMI access is also possible to this server. In case the server is out of service, new repository is needed, or other problems, feel free to contact us at info@ontotext.com.

For test purposes one may use the "Sirma Skills KB" repository (with ID "mysql-sskb"), which is populated with the example presented in section 3. .

5.2. Built-in Usage of OMM and Sesame

This section explains how to build-in/embed the *OMM* and *Sesame* in another project or front-end Java application.

Since in this mode it is possible to access all the methods of the SAIL interfaces a complete JavaDoc reference is needed to use it. It is available at <http://www.ontotext.com/OMM/StandAloneSesame/completeJavaDoc/index.html> . All the classes that implement the **nl.aidadministrator.rdf.sail.Sail** interface could be used in a stacked SAIL architecture defined in the configuration file used.

A class demonstrating this kind of usage (**builtin.BuiltInSesame**) is available at <http://www.ontotext.com/OMM/StandAloneSesame/StandAloneSrc.zip>. In the same archive there is an example configuration file (**system.conf**) which we suggest to be placed in the work folder of the project. In a set of experiments it demonstrates the usage of OMM/Sesame by performing a connection to our demo public server. The first example demonstrates usage of Sesame through the **RequestRouter** class; the second experiment uses directly the SAIL interface and the third one uses the **VersionManagement** interface to demonstrate the versioning and tracking changes capabilities added to Sesame by the OMM.

The class `com.ontotext.omm.util.SesameStartup` is used in the first experiment in order to initialize the stacked interfaces and to provide access via the **RequestRouter** class. **SesameStartup** provides a set of static initialization methods that should be used to initialize Sesame/OMM. The technical documentation of the stand-alone startup class is available at <http://www.ontotext.com/OMM/StandAloneSesame/SesameStartupDoc/index.html>.

The initialization via `com.ontotext.omm.util.SesameStartup` requires one of the following (i) a configuration file name, (ii) configuration file handler class and map of initialization parameters, (iii) a configuration file name and a parser class for it. It has been demonstrated with the first option.

5.2.1. Dependencies

This section specifies the dependencies of Sesame/OMM on third party libraries. In order to use Sesame\OMM in a built-in mode, the following libraries are required:

- `sesame.jar` – the classes of Sesame/OMM
- `jena.jar` – RDF(S)/DAML parser
- `rdf-api-xxx.jar` – RDF API
- `xerces.jar` (or XML parser)
- the JDBC drivers allowing access to the repository (in the example source this is `mm.mysql-2.0.4-bin.jar`).

All the third party libraries related to Sesame/OMM are available at <http://www.ontotext.com/omm/downloads.html>.

5.2.2. *Initialization*

A static invocation of one of the `initialize(...)` methods of the

```
com.ontotext.omm.utils.SesameStartup
```

class initializes the stacked SAILs. For instance:

```
SesameStartup.initialize("C:\\tomcat\\webapps\\sesame\\WEB-INF\\system.conf");
```

Afterwards the class `nl.administrator.rdf.config.SystemConfigCenter` (java doc reference available at <http://www.ontotext.com/OMM/StandAloneSesame/SystemConfigCenterDoc/index.html>) is used to gain access to the first stacked SAIL in a specified repository. This is demonstrated in the second example in the `builtin.BuiltInSesame` class available at <http://www.ontotext.com/OMM/StandAloneSesame/StandAloneSrc.zip>.

5.2.3. *Access APIs*

The access to Sesame/OMM is based on the

```
nl.administrator.rdf.req_router.RequestRouter
```

and

```
nl.administrator.rdf.config.SystemConfigCenter
```

 interfaces. From which one could retrieve any of the stacked SAILs.

5.3. RMI Access

RMI is supported as an alternative access method to Sesame. A couple of interfaces and their according server side implementations are used to provide an RMI access to Sesame, those reside in the `com.ontotext.omm.rmi` package. The entry point to Sesame is `FactoryInterface`. Its only method is used to gain access to the `ServicesInterface` through which the rest of the Sesame services are available. A standalone or servlet-based Sesame application can bind an instance of the `FactoryInterfaceImpl` only if an `rmiRegistry` (an executable, part of the JDK distribution) is running on the server. Both interfaces should be compiled with the `rmcp` utility so the proper stub and skeleton code to be generated. Refer to the RMI documentation for full coverage of the interface compiling issue. Once the stub/skeleton code is present it could be packed with the factory and services interfaces in a `jar`. The `jar` should be added to the class path of the `rmiRegistry` application in order to bind/unbind, create, and use the Sesame RMI interfaces.

5.3.1. Client

This section contains a sample use case of Sesame via RMI. The first thing is to get a reference to the RMI Registry running on the server. This is performed via the call:

```
java.rmi.registry.Registry reg = LocateRegistry.getRegistry(RMI_HOST);
```

Where `RMI_HOST` is pointing to the server on which the RMI Registry is running. E.g. our demo server `dell.sirma.bg`

Afterwards, reference to the `FactoryInterface` should be retrieved from the newly located registry:

```
com.ontotext.omm.rmi.FactoryInterface f =
    (com.ontotext.omm.rmi.FactoryInterface)
    reg.lookup("FactoryInterface");
```

The next step is to get access to the `ServicesInterface` via the retrieved `FactoryInterface`:

```
com.ontotext.omm.rmi.ServicesInterface si = f.getService();
```

After this the services available could be accessed in the client application. A sample application, which could be seen as an example in the `testrmisesame.TestSesameRMI` class at <http://www.ontotext.com/OMM/RMI/RMIClientSrc.zip>.

The lifetime of a session is the same as the lifetime of the `ServicesInterface` instance returned from `getServices()` from the `FactoryInterface`. The JavaDoc documentation of the `ServicesInterface` is available at <http://www.ontotext.com/OMM/RMI/ServicesInterface/index.html>.

5.4. SOAP Access

OMM allows Sesame to be accessed through SOAP. The methods (or calls or messages) supported are analogous to those supported for the rest of the protocols, for price references check the Java client documentation. This section presents instructions for OMM/Sesame server installation as well as for SOAP access from client applications.

5.4.1. Server

In order to access Sesame via SOAP a fresh copy of the apache SOAP implementation should be retrieved from <http://xml.apache.org/soap> (or alternative implementation of the SOAP API). Afterwards the installation instructions should be followed in order to get SOAP running on a previously installed servlet server (e.g. Jakarta/Tomcat). We experienced problems to access the running Sesame servlet based application via SOAP interface with versions of Jakarta/Tomcat server less than 4.0, so, these instructions concern versions above 4.0.

Some changes to the installation of Sesame are required, in order to make it available via the SOAP interface. Instead of placing `sesame.jar` in the `$TOMCAT_HOME\webapps\sesame\WEB_INF\lib` folder⁴, place it in the `$TOMCAT_HOME\common\lib` or in the folders mentioned in the `-Djava.endorsed.dirs` system property. This makes all the Sesame classes available for the SOAP running context. Once Sesame and SOAP have been installed and started, the Sesame SOAP service should be deployed on the server. Refer to the SOAP documentation for that issue and use the prepared `DeploymentDescriptor.xml` (<http://www.OntoText.com/OMM/SOAP/DeploymentDescriptor.xml>) to deploy the Sesame SOAP service.

5.4.2. Client

Client applications from different platforms written in all most popular programming languages can access OMM/Sesame through SOAP. Here we present SOAP access to OMM from Java application, including instructions, client library, and a sample application.

The following libraries are required:

- `soap.jar` – from the Apache SOAP site (<http://xml.apache.org/soap>);
- `mail.jar` – from <http://java.sun.com/products/javamail/>;
- `activation.jar` – from <http://java.sun.com/products/beans/glasgow/jaf.html>;
- An XML parser is required too.

In order for an application to be able to make SOAP calls to Sesame, the `com.ontotext.omm.soap.SoapClient` wrapper class should be instantiated, using the specific server URL and service ID. The client should login on the server and select a working repository.

The sources of the simple client library (the above mentioned `soapClient` class) and a sample application (the `soapTest.Testsoap` class) that connects to a hard-coded server and repository are available at <http://www.ontotext.com/OMM/SOAP/SoapClientSrc.zip>. The documentation of the SOAP Client is available at <http://www.ontotext.com/OMM/SOAP/SoapClientDoc>.

⁴ In more recent Jakarta versions, $\geq 4.0.1$, `TOMCAT_HOME` has been changed to `CATALINA_HOME`

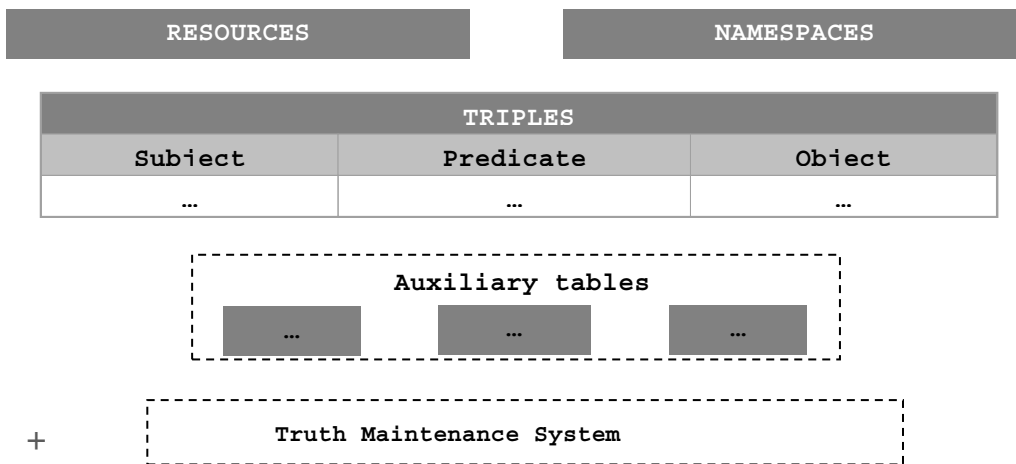
6. Implementation Details

More important implementation details and alternations in the design (with respect to Deliverable 38, [Kiryakov et al, 2002]) are presented here for the major sub-systems of the ontology middleware – tracking changes and access control. For good understanding of the following subsections understanding respectively of sections 2 and 3 of Deliverable 38 is required.

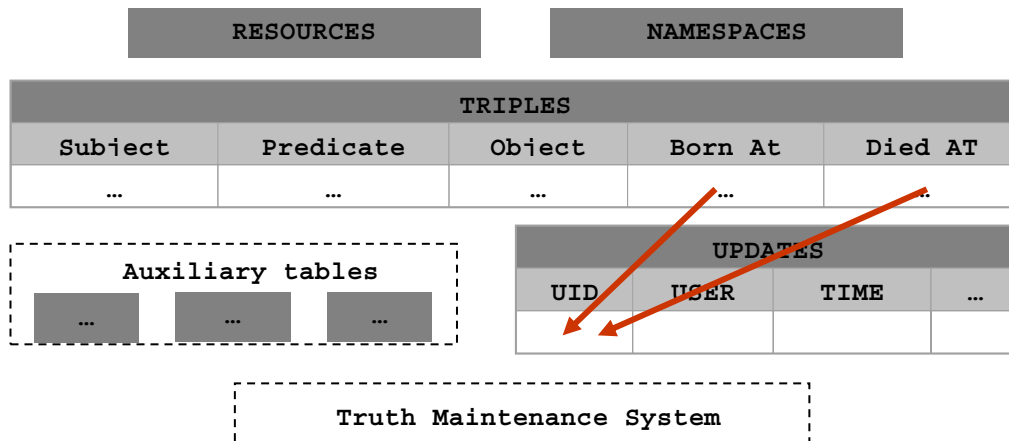
6.1. Tracking Changes Implementation

The only significant change in the implementation plans was related to the necessary extension of the DB schema of SESAME for the purposes of the ontology middleware. However, first it is important to acknowledge that after the requirements for a truth maintenance system (TMS) in SESAME, outlined during the analysis and design for the OMM, such TMS was developed by administrator b.v. on a fairly short time scale, taking into account its complexity.

The database schema used from the SQL92SAIL (and respectively the MySQL implementation) :



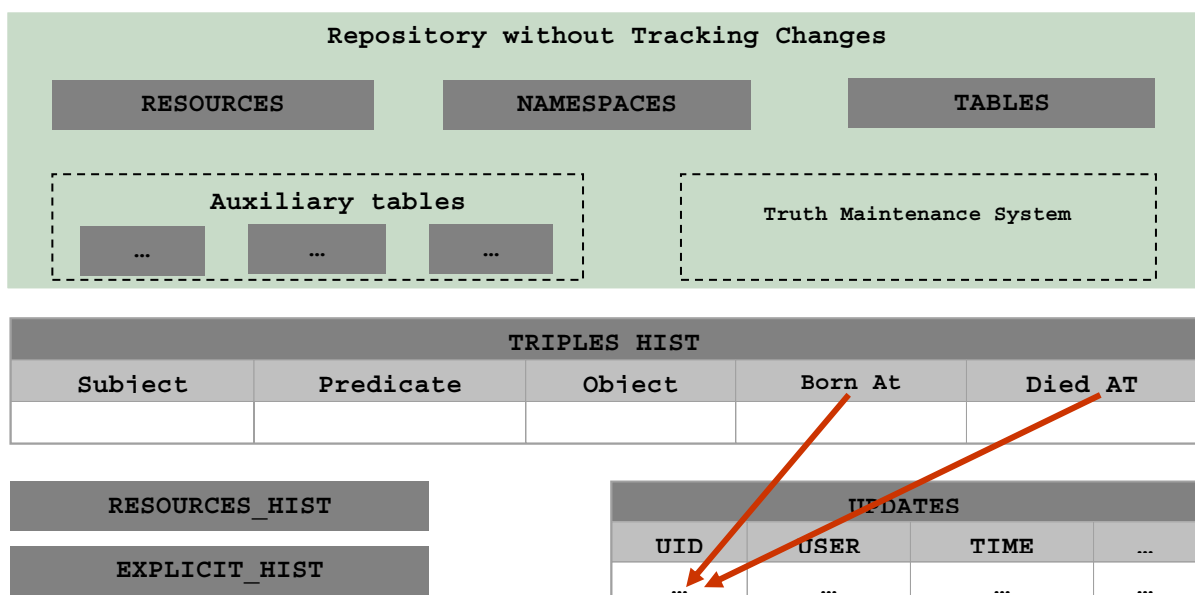
The original plan and the first implementation of the ontology middleware extensions of SESAME was build on the following database schema:



The above schema was based on the idea to extend the existing TRIPLES table with two more columns with references to the updates which caused the adding and deletion of the corresponding statement. Under this schema, the triples that are “valid” in certain state of the repository can be queried via the appropriate restrictions on the `BornAt` and `DiedAt` columns. This idea appeared to be problematic due to a number of reasons:

- There was a need for re-implementation of the basic SAIL in order to add methods that can work with previous versions of the repository without making use of the auxiliary tables (that always correspond to the current state.)
- This schema fails to account for the transitions between implicit and explicit states of the statement during its lifetime. This is a major problem as far as many methods in the SAIL interface the query module count on this distinction.
- Finally, the performance when working with the current version was affected due to need of additional conditions on each query including the TRIPLES table.

After analysis of different design alternatives and their impact on the performance, we implemented the following schema:



The core idea is that the original tables remain unchanged and additional (in a sense parallel) tables are used to preserve the tracking information. When a user needs to be able to work with a previous version, the appropriate state is branched into a separate auxiliary repository which is read-only and has no support for tracking changes. This schema has the following advantages and disadvantages:

- some information is duplicated (-);
- to start work with old version a dummy branch is created, which takes both considerable time and space (-);
- working with the current version is as fast as without tracking, the implementation is much simpler (+);
- working with old version is as fast as working with the current one – the same optimizations work (+);

6.2. Access Control Implementation

There was a single major modification necessary for the implementation of the security sub-system. After more careful analysis of different use cases and sample security policies, we uncovered that one additional restriction type is necessary, namely **classes-over-schema** – it is specified by a set of classes and determines the set of all resources which represent their sub-classes⁵. In contrast, the **classes** restriction type determines set of resources which are instances of the specified classes.

An example usage of both the *Classes and Classes-over-schema* restriction types is available in sub-section 4.2.1. above.

⁵ As well as the statements which has such resources as subject – no change with respect to the **Classes** restriction type in this.

7. Import Demo Security Setup

The import of security setup data is provided from RDF(S). The schema for the setup is at

www.ontotext.com/otk/2002/03/KCS.rdfs. The security setup itself consists of instances of the classes in the schema (e.g. instances of <http://www.ontotext.com/otk/2002/03/kcs.rdfs#Restriction>, <http://www.ontotext.com/otk/2002/03/kcs.rdfs#SecurityRule>, etc.) and statements defining the relations of these instances to other resources (both classes and instances).

It is recommended the security setup to be defined in RDF(S) according to the `kcs.rdfs` schema, and afterwards loaded in one of the following two ways. The security setup used in the demo is accessible at <http://www.ontotext.com/omm/SecurityDemo/DemoSecuritySetup.rdf>. In order to define a desired accessibility over a set of resources one should first decide what kind of restriction to be used over the repository. A sample restriction definition follows:

```
<rdf:Description rdf:about='http://www.ontotext.com/otk/2002/03/kcs.rdfs#Persons'>
  <NS0:id>7</NS0:id>
  <NS0:name>Persons</NS0:name>
  <NS0:description>Persons</NS0:description>
  <NS0:restrictionType>ClassesRestriction</NS0:restrictionType>
  <rdf:type rdf:resource = http://www.ontotext.com/otk/2002/03/kcs.rdfs#ClassesRestriction' />
  <NS0:includeResource>http://www.ontotext.com/otk/2002/05/enterprise.rdfs#Person
  </NS0:includeResource>
</rdf:Description>
```

Where NS0 is the namespace defined by <http://www.ontotext.com/otk/2002/03/kcs.rdfs#>.

This restriction (*ClassesRestriction*) is a sub class of *ResourceRestriction*. All these restrictions are defined via a set of resources. The resources included are specified via the `<NS0:includeResource>` tag. Similar properties are used to define the other types of restrictions. The properties consisted in a *PropertiesRestriction* are defined via an `<includeProperty>` tag. The *subject/predicate/object restrictions* which define a *PatternRestriction* are defined accordingly via the properties *subjectRestr*, *predicateRestr*, *objectRestr*.

Then a *security rule* is defined and the restriction is assigned to the rule via the property *ruleRestriction*. The *rights* associated with this rule are specified via a *rightsGranted* property. Example of a security rule definition follows:

```
<rdf:Description rdf:about='http://www.ontotext.com/otk/2002/03/kcs.rdfs#PersonHasPositionRead'>
  <rdf:type rdf:resource='http://www.ontotext.com/otk/2002/03/kcs.rdfs#SecurityRule' />
  <NS0:id>7</NS0:id>
  <NS0:name>PersonHasPositionRead</NS0:name>
  <NS0:ruleRestriction>http://www.ontotext.com/otk/2002/03/kcs.rdfs#PersonHasPosition
  </NS0:ruleRestriction>
  <NS0:description>PersonHasPositionRead</NS0:description>
  <NS0:rightsGranted>READ</NS0:rightsGranted>
</rdf:Description>
```

The created *security rule* should be associated with a set of *users* or with a set of *roles*.

The *parent roles* of the role being defined could be specified via the *superRole* property. A *security rule* is associated with a *role* in a statement with property *includeRule*. An example is given through the following role definition:

```
<rdf:Description rdf:about='http://www.ontotext.com/otk/2002/03/kcs.rdfs#SoftwareEngineerRole'>
  <rdf:type rdf:resource='http://www.ontotext.com/otk/2002/03/kcs.rdfs#Role' />
  <NS0:id>1</NS0:id>
  <NS0:name>SoftwareEngineerRole </NS0:name>
  <NS0:includeRule>http://www.ontotext.com/otk/2002/03/kcs.rdfs#Rule1 </NS0:includeRule>
  <NS0:includeRule>http://www.ontotext.com/otk/2002/03/kcs.rdfs#Rule7 </NS0:includeRule>
  <NS0:superRole>http://www.ontotext.com/otk/2002/03/kcs.rdfs#ParentRole1 </NS0:superRole>
  <NS0:description> SoftwareEngineerRole </NS0:description>
</rdf:Description>
```

Finally, the user should be assigned a set of roles or/and a set of rules. The roles are assigned via the property *hasRole*, and the rules are assigned using the property *hasRule*. The *id* property of the user instance should be set, too. Since the users are defined in the *configuration file* (*system.conf*) it is necessary to synchronize the ids used in the security setup and these defined in the *configuration file*. The following example presents assigning a role and a rule to a user :

```
<rdf:Description rdf:about='http://www.ontotext.com/otk/2002/03/kcs.rdfs#User_Haiasko'>
  <rdf:type rdf:resource='http://www.ontotext.com/otk/2002/03/kcs.rdfs#User' />
  <NS0:id>17</NS0:id>
  <NS0:hasRule>http://www.ontotext.com/otk/2002/03/kcs.rdfs#Rule_UniversalKnowledge
</NS0: hasRule>
  <NS0:hasRole>http://www.ontotext.com/otk/2002/03/kcs.rdfs#Role_Chief_Galaxy_Librarian
</NS0: hasRole >
</rdf:Description>
```

7.1. Specify the security setup in the *system.conf* configuration file

For use of the OMM through the web interface it is most appropriate to set a parameter in the *system.conf*, which specifies the URL of the security setup to be loaded. This parameter is called *security_setup* and appears in the parameters of a *security sail*. In order to be able to import/export the security setup, a user should have *Admin* right over the repository. This right should be defined in a rule in the *current* security setup (not in the one being imported). Sample definition follows:

```
<sail class="com.ontotext.omm.security.SecuritySail">
  <param name="jdbcDriver" value="org.gjt.mm.mysql.Driver"/>
  <param name="jdbcUrl" value="jdbc:mysql://localhost:3306/testdbver"/>
  <param name="user" value="sesame"/>
  <param name="password" value="sesame"/>
  <param name="security_setup"
    value="http://www.ontotext.com\otk\2002\05\SecurityDemo\DemoSecuritySetup.rdf"/>
</sail>
```

7.2. Import security setup via `com.ontotext.omm.SecurityServices` interface.

This approach is applicable if using Sesame and OMM from within another project. The `com.ontotext.omm.SecurityServices` interface provides methods for import/export of the security setup. The methods require a reader/writer to the security setup. Only users, with *Admin* right set over the whole repository, are able to export the security policy. Example usage of this functionality follows:

```
securitySail.exportSecurityPolicy2RDF(
    new PrintWriter(new FileOutputStream("c:/sesame/out/SecuritySetup.rdf" ));

securitySail.importSecurityPolicy(
    new FileReader("c:/sesame/out/SecuritySetup.rdf"));
```

In order to get the full picture, an example main method follows. It initializes a couple of SAIL layers, loads the configuration file (e.g. `system.conf`), creates an appropriate *session context* (e.g. `admin`), exports the setup to RDF(S) and then loads it back:

```
public static void main(String[] args) {
    VersioningTmsMySQLSail verSail= new VersioningTmsMySQLSail();
    SecuritySail securitySail = new SecuritySail();
    securitySail.setBaseSail(verSail);

    HashMap map = new HashMap();
    map.put("jdbcDriver", "org.gjt.mm.mysql.Driver");
    map.put("jdbcUrl", "jdbc:mysql://localhost:3306/testdbver");
    map.put("user", "sesame");
    map.put("password", "sesame");

    try {
        SessionContext context = new SessionContext();
        context.user = "admin";
        context.pass = "admin";
        context.userID=1;
        context.baseUrl = "http://www.ontotext.com/otk/2002/05/SecurityDemo/newSwingSkill.rdf";
        context.repository = "mysql-dbver";
        SessionContext.put("", context);
        SessionContext.setContext(context);

        verSail.initialize(map);
        securitySail.initialize(map);

        context = SessionContext.getContext();
        Map pm = new HashMap();
        pm.put("systemConfigFile",
```

```
"C:/JBuilder6/jakarta-tomcat-4.0.1/webapps/Sesame/WEB-INF/system.conf");

System.setProperty("org.xml.sax.parser", "org.apache.xerces.parsers.SAXParser");

SystemConfigFileHandler ch = new SystemConfigFileHandler();
ch.setParameters(pm);
SystemConfig cfg = ch.loadConfig();
SystemConfigCenter.refresh(cfg);

securitySail.exportSecurityPolicy2RDF(
new PrintWriter(
    new FileOutputStream("c:/sesame/out/SecuritySetup.rdf"));

securitySail.importSecurityPolicy(new FileReader("c:/sesame/out/SecuritySetup.rdf"));

} catch (Exception e) {
    e.printStackTrace(System.out);
    System.out.println(e.getMessage());
}
securitySail.shutdown();
}
```

8. Performance comments

During the current phase of the project we concentrated on implementation of the basic functionality of the OMM leaving more serious optimization and performance evaluation plans for the next phase, Integration. Here follow some preliminary comments on various aspects of the OMM. The Security sub-system is still under optimization and performance evaluation.

Among the most important issues about the performance of the OMM is time and space overhead caused by the mechanisms for tracking of the changes. Few measurements shared below:

- The time overhead when adding statements (uploading ontologies) within a repository with tracking changes support (as compared to such without) is below 5%. This means that the price for supporting all this additional information related to the tracking is fairly low.
- The space overhead (in the database) caused by the additional tracking information depends on the number of updates performed over each of the statements in the repository. In a hypothetical situation, when 100 resources and 500 statements are initially added, than half of them are deleted and another 50 resources and 250 statements were uploaded, the overhead (in terms of volume in the database) is some additional 150%. It can be explained as follows: the historical data for each statement or resource that ever appeared in the repository is roughly as much as the “basic”/“regular” information preserved for it. This for instance, means that immediately after upload in an empty repository with tracking support, the space overhead will be something like additional 100%.
- The time for branching of (certain state of) a repository is comparable with the time for uploading the explicit statements in a new repository.
- As it was expected, there is no impact on the non-modifying transactions with the repository – all queries on the current and previous versions work without any delay as compared to repository without tracking support.

Another important performance issue is the difference in the access speed through the various access methods (RMI, SOAP, HTTP, Built-In use). All these methods are presented in [Section 5](#). For each of these access methods, identical test cases have been performed, separated in the following three phases:

- *Initialization Phase* in which the respective access method is being initialized. This phase also includes login, password and database specifications, as well as access-method-specific setup information (e.g. URL of the server, configuration file, etc.)
- *Query Phase* in which two RQL queries are sequentially executed. The first one, being a simple *give-me-all-triples* query, and the second one a more complex *path* query.
- *SAIL API Phase* in which invocation of some of the SAIL methods takes place. This phase is not performed for the HTTP protocol, since the SAIL API is not available through it. The SAIL methods used in this phase are *getClasses()* and *getStatements(...)*.

The sources of this access methods performance measurement experiment are available at <http://www.ontotext.com/omm/accessmetrix/accessmetrix.zip>. The experiment could be easily started remotely since it uses the <http://omm.ontototext.com> demo server and has a configuration file (*system.conf*) included in it. The configuration file (*system.conf*) should be placed in the project’s working folder. All the required third party libraries could be found at

<http://www.ontotext.com/omm/downloads.html>.

In the following table, the results of the performance experiment are being presented in a normalized form. The metrics are not in any particular time unit, but instead use as a basis the access time for the respective Built-In phase, which is considered equal to 1 (one) virtual time unit. These results are formed on the average metrics after three identical executions of the experiment.

Access Methods Performance Metrics

	Built-In	HTTP	RMI	SOAP
Query Phase	1.00	1.04	1.09	1.78
SAIL API Phase	1.00		1.33	4.87

It is easily seen that the fastest way to use Sesame/OMM is the Built-In use, which is intuitively expected. As for using RQL queries HTTP and RMI perform in a similar manner. SOAP is significantly slower in the Query Phase, which is also expected because of the XML serialization.

The Built-In use performs even better in the direct SAIL API usage phase, where even RMI is way behind, not to mention SOAP.

9. References

- [Brickley and Guha, 2000] W3C; Dan Brickley, R.V. Guha, eds.
Resource Description Framework (RDF) Schemas.
<http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>
- [Broekstra and Kampman, 2001a] Jeen Broekstra, Arjohn Kampman
Query Language Definition.
Deliverable 10, On-To-Knowledge project, May 2001.
<http://www.ontoknowledge.org/download/del10.pdf>
- [Broekstra and Kampman, 2001b] Jeen Broekstra, Arjohn Kampman
Sesame: A generic Architecture for Storing and Querying RDF and RDF Schema.
Deliverable 9, On-To-Knowledge project, October 2001.
<http://www.ontoknowledge.org/download/del10.pdf>
- [Cycorp 1997] Cycorp. Cyc Public Ontology, 1997.
<http://www.cyc.com/cyc-2-1/>
- [Ding et al, 2001] Ying Ding, Dieter Fensel, Michel Klein, Borys Omelayenko
Ontology management: survey, requirements and directions.
Deliverable 4, On-To-Knowledge project, June 2001.
<http://www.ontoknowledge.org/download/del4.pdf>
- [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>
- [Lassila and Swick, 1999] W3C; Ora Lassila, Ralph R. Swick, eds.
Resource Description Framework (RDF) Model and Syntax Specification
<http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>
- [Simov and Jordanov, 2002] Simov K. Iv., Jordanov S. *BOR: a Pragmatic DAML+OIL Reasoner.* Deliverable 40, On-To-Knowledge project, June 2002.
<http://www.ontoknowledge.org/download/del40.pdf>
- [Novotny and Lau, 2001] Novotny B., Lau T. *Organizational Memory Description of Case Study Prototypes.* Deliverable 20, On-To-Knowledge project, June 2002.
<http://www.ontoknowledge.org/download/del20.pdf>