Integrated Ontology Development Toolkit

Version 1.1.2

May 2006

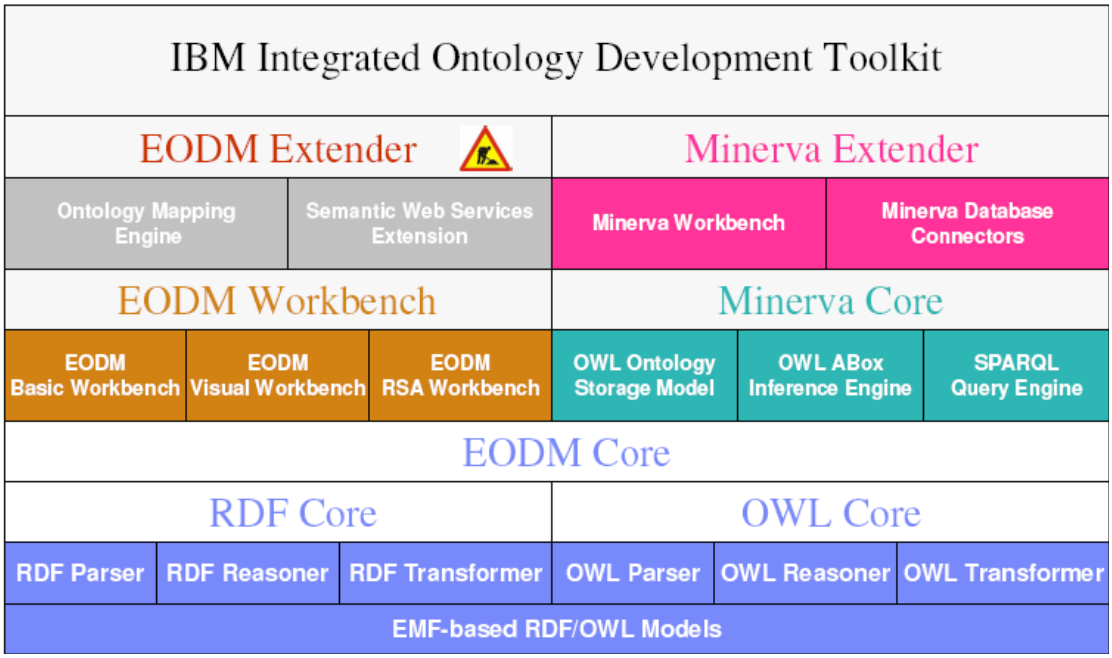
IBM

Table of Contents

Introduction

IODT is a toolkit for ontology-driven development, including EMF Ontology Definition Metamodel (EODM), EODM workbench, an OWL Ontology Repository (named Minerva) and some extenders to the core components. EODM is derived from the OMG's Ontology Definition Metamodel (ODM) and implemented in Eclipse Modeling Framework (EMF). To facilitate software development and execution, EODM includes RDFS/OWL parsing and serialization, reasoning, and transformation between RDFS/OWL and other data-modeling languages. These functions can be invoked from the EODM Workbench, Minerva, or called by the applications that use ontologies.

EODM is also an open source project of Eclipse.org (http://www.eclipse.org/emft/projects/eodm/). EODM Workbench is an Eclipse-based, integrated, ontology-engineering environment that supports ontology building, management, and visualization. Minerva is a high-performance OWL ontology storage, inference and query system based on RDBMSes. Following is the system architecture of IODT (EODM RSA Workbench, Minerva Database Connectors and EODM Extender are under developing and would be available in the future release).



IODT includes the following components, which can be downloaded and used independently or cooperatively:

- EODM is the run-time library that allows the application to put in and put out an RDFS/OWL ontology in RDF/XML format; manipulate an ontology using Java objects; call an inference engine and access inference results; and transform between ontology and other models.
- EODM Workbench is an Eclipse-based, integrated, ontology-engineering environment that supports ontology building, management, and visualization.
- OWL Ontology Repository (named Minerva) is a high-performance OWL storage and query system based on relational DBMSes. It supports DLP (Description Logic Program), a subset of OWL DL, and conjunctive query, a subset of the SPARQL language.

What is EODM

EODM (EMF Ontology Definition Metamodel) is built on top of EMF and conforms the ODM(Ontology Definition Metamodel) standard of OMG(www.OMG.org). It provides a set of programming APIs for programmers and IT specialists. User can create, modify, and navigate RDF/OWL models using EODM, just like other programming implementations of semantic models (such as Jena).

Compared with others tools, the most important differentiation of EODM is its ability of bi-direction transiformation between RDF/OWL model and ECore Model, which gives EODM the interoperationability between RDF/OWL model with other EMF supported models, including models defined using XML Schema, Rational Rose, and annotated Java classes. All these will enable Semantic Web Application developers to develop ontologies using their favorite building tools, import them into EMF, and utilize the comprehensive development facility of Eclipse and EMF.

Components of EODM

EODM includes 4 main components: EODM Model, EODM RDF/OWL Parser, EODM RDF/OWL Reasoner and EODM Transformer

- EODM Model provides a programmatic memory-based environment to support creating & manipulating Ontology through a set of RDF/OWL API. Currently it supports 2 models: RDF/RDFS Model and OWL Model.
- RDF/OWL Parser provides an efficient parser to parse RDF/OWL files into EODM model. The EODM Parser have passed all RDF test cases in W3C.
- RDF/OWL Reasoner: EODM also provides reasoner modules to support inference about Ontology. EODM provides two types of Reasoner: RDF/RDFS Reasoner and and OWL Reasoner.
- Transformer: Based on EMF Ecore, EODM Transformer provides the ability of transforming EMF supported models to Ontology models. Currently the EODM Transformer can only support Ecore to RDF. The transform between Ecore to OWL is still under development.

What is EODM Workbench

EODM Workbench is an Eclipse-based editor for users to create, view and generate OWL ontologies. It has UML-like graphic notions to represent OWL class, restriction and property etc. in a visual way. EODM workbench is built by using EODM, Eclipse Modeling Framework (EMF), Graphic Editing Framework (GEF), which provides the foundation for the graphic view of OWL. It supports drag and drop operations and drawing functions. It also provides two hierarchical views for both OWL class/restriction and OWL object/datatype property. In EODM workbench, oneontology can have multiple views to highlight its different perspectives. Operations in different views can be synchronized automatically.

Key Features of EODM Workbench

- Eclipse-based Tool

EODM workbench is an Eclipse plugin that can integrate with any other plugins. It depends on EMF and GEF. It has its own OWL perspective, graphic OWL editor and two hierachical view. It uses the OWL model and parser in EODM.

- OWL Support

Orient, the previous version of ontology editor, can only support RDF Schema and RDF document. EODM Workbench now can support OWL-DL constructs, and generate .owl files.

- UML-like Graphic Notion

In addition to traditional tree-based ontology visualization, EODM workbench provides UML-like graphic notion. Class, DatatypeProperty and ObjectProperty in OWL share the similar notion as Class, Attribute and Association in UML. Detailed properties of OWL constructs are shown in the Property view.

- Multiple View

In EODM Workbench, one ontology can have multiple views to support visualization of huge ontology. These views are independent, and users can decide to delete something from ontology permanently, or from view only. Multiple views will be synchronized automatically.

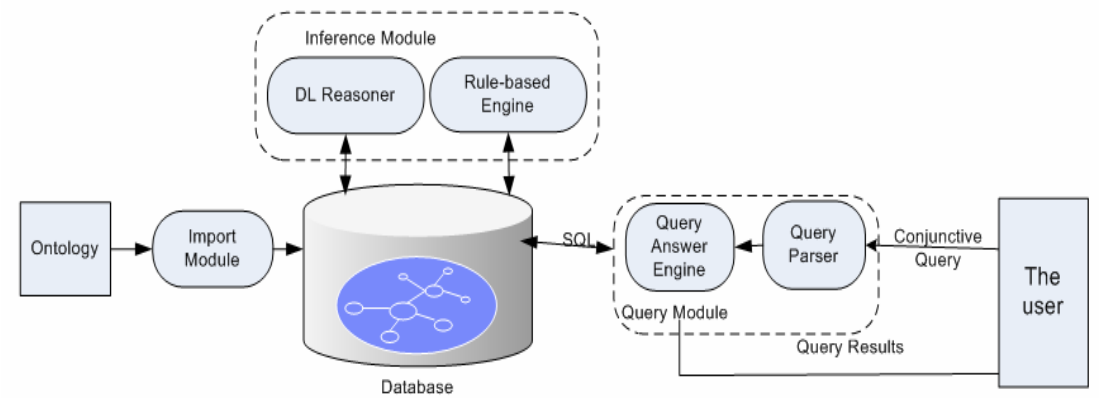
- Class/Property Hierarchy

EODM Workbench keeps the tree-based editor for class and property, and provides some shortcut to create subClassOf and subPropertyOf in the hierarchy. Users can drag class and property in the tree to the canvas in the view.

What is Minerva

Minerva is a high performance ontology repository built on relational databases. It completely follows W3C's Web Ontology Language (OWL) and SPARQL Query Language for Resource Description Framework. Minerva uses Description Logic reasoner for TBox inference and a set of logic rules translated from Description Logic Programming (DLP) for ABox inference. That is, its inference capability is over DLP. This means that inference completeness and soundness on DLP can be guaranteed. Different from other systems, Minerva designs the schema of the back-end database based on both the translated logic rules and OWL constructs to support efficient ontology inference. Our experiments on the extended Lehigh University Benchmark show Minerva has high scalability and is better than other two well-known systems.

Following figure shows the architecture of Minerva, including Import Module, Inference Module, Storage Module and Query Module.



Storage Module. It is intended to store both original and inferred assertions. Since inference and storage are considered as an inseparatable component in a complete storage and query system for ontologies, we design the schema of the back-end database to optimally support ontology inference in Minerva.

Import Module. The import module consists of an OWL parser and a translator. The parser analyzes and transforms OWL documents into an Object Oriented Model (EODM) in memory, then the translator populates all assertions into the database.

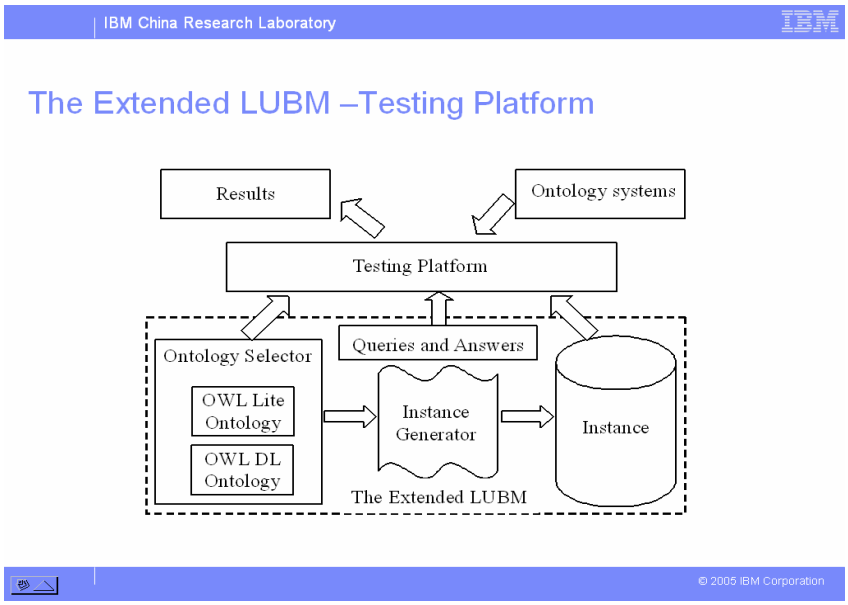
Inference Module. A DL reasoner and a rule-based engine compose the inference module. The rule-based inference covers the semantics of DLP, while the DL reasoner obtains the subsumption relationship between classes and properties which cannot be completely captured by the rules.
Query Module. The query language supported by Minerva is SPARQL, W3C's query language for RDF. There is no inference during the query answering stage because the inference has already been done at the time of data loading. Such processing improves the query response time by trading off space.

Key Features of Minerva

- Efficient management for large-scale OWL ontologies (millions of statements)
- DBMSes Supported
  - IBM DB2 (Powerful Persistent Storage)
  - Derby (http://incubator.apache.org/derby/, Embedded Storage)
  - HSQLDB (http://www.hsqldb.org/, Main Memory Storage)
- Query Language
  - W3C SPARQL Query Language
- TBox Inference Engines
  - Racer
  - Pellet
  - Structural TBox Engine (IBM CRL)
- Memory Model
  - EODM

Performance of Minerva

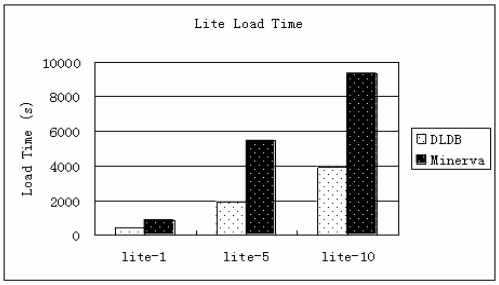
Experiments have been done to evaluate performance of Minerva on the extended Lehigh University Benchmark. The extended LUBM includes both OWL Lite and DL ontologies covering a complete set of OWL Lite and DL constructs (except TBox with cyclic definition) respectively. We also add necessary properties to construct effective instance links and improve instance generation methods to make the scalability testing more convincing.

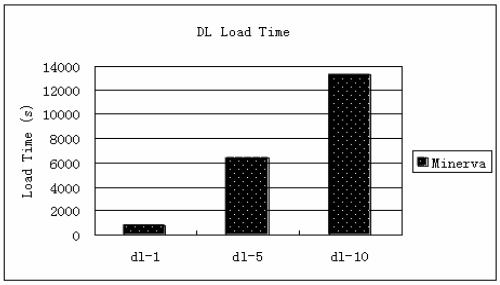


Besides Minerva, two more systems are evaluated on the extended LUBM for comparison. OWLIM is a newly developed ontology repository supporting OWL Lite inference. In fact, it is packaged as a Storage and Inference Layer (SAIL) for Sesame RDF database. DLDB-OWL (Lehigh University) is a repository for processing, storing, and querying large amounts of OWL data.

Experimental Results (Persistent Systems)

- **Load Time**



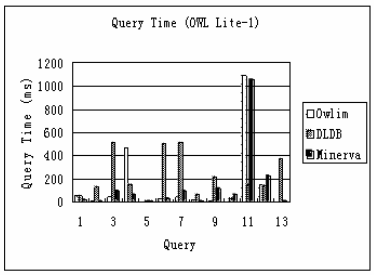


OWLIM can handle lite-1 data set since it conducts reasoning and query in memory. Minerva takes only about 2.5 hours to load OWL lite-10 data set. DLDB has the fastest speed due to doing less inference than Minerva at load time.

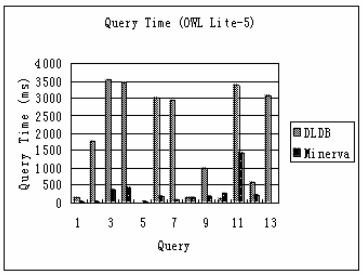
IBM China Research Laboratory

Experimental Results (Persistent Systems)

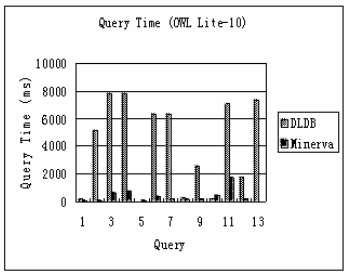
- **Average Query Response Time**



About 220, 000 triples



> 1,100, 000 triples



> 2,200, 000 triples

Experimental Results (Persistent Systems)

- Query Completeness

Query		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
OWLIM	Lite-1	1	0.62	1	1	1	1	1	0.86	1	1	1	1	0	NA	NA
Pellet	Lite-1	1	1	1	1	1	1	1	1	1	1	1	1	1	NA	NA
	DL-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
DLDB	Lite-1	1	0.82	1	1	0	0	0	0	0	0.83	0	0.2	0.51	NA	NA
	Lite-5	1	0.81	1	1	0	0	0	0	0	0.59	0	0.12	0.57	NA	NA
	Lite-10	1	0.81	1	1	0	0	0	0	0	0.87	0	0.26	0.53	NA	NA
Mineva	Lite-1	1	1	1	1	1	1	1	1	1	1	1	1	0.67	NA	NA
	Lite-5	1	1	1	1	1	1	1	1	1	1	1	1	0.61	NA	NA
	Lite-10	1	1	1	1	1	1	1	1	1	1	1	1	0.64	NA	NA
	DL-1	1	1	1	1	1	1	1	1	1	1	1	1	0.90	0.96	0
	DL-5	1	1	1	1	1	1	1	1	1	1	1	1	0.88	0.97	0
	DL-10	1	1	1	1	1	1	1	1	1	1	1	1	0.88	0.95	0

Minerva achieves better query completeness than OWLIM and DLDB. Since OWLIM cannot do complete TBox inference, it cannot correctly answer queries 2, 8 and 13. Currently, Minerva does not support Cardinality inference and thus cannot answer query 13. DLDB implements only part of OWL Lite ABox inference.

Experimental results showed high performance of Minerva in terms of query completeness, query response time, scalability and load time.

System Requirements and Installations

Installation for use within Eclipse

If you have installed a previous version of EODM in Eclipse before, please uninstall it before installing the new version in the following four steps:

1. Exit Eclipse if it is running.
2. Delete all sub-directories starting as “com.ibm.research.minerva” in the $ECLIPSE_HOME/ features/ directory
3. Delete all sub-directories starting as “org.eclipse.eodm” in the $ECLIPSE_HOME/features/ directory
4. Delete all sub-directories and JAR files starting as “org.eclipse.eodm” in the $ECLIPSE_HOME/plugins/ directory.
5. Start Eclipse with a “-clean” argument and proceed with the following instructions.

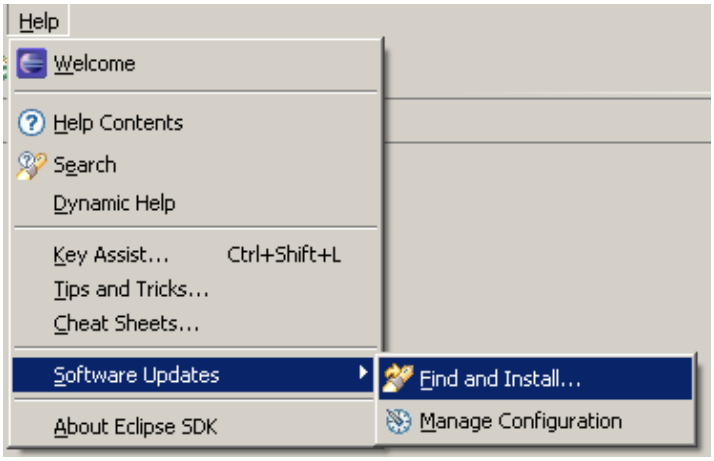
Installation Requirements

- JRE 1.4.2
- Eclipse 3.1.0
  http://fullmoon.torolab.ibm.com/downloads/drops/R-3.1-200506271435/index.php
- EMF 2.1.0 (XSD and SDO is not needed)
  http://fullmoon.torolab.ibm.com/tools/emf/scripts/downloads-viewer.php?s=2.1.0/R200507070200
- To use EODM Workbench, GEF 3.0.0 is needed. It does NOT work with GEF 3.1.0 or above.
  http://download.eclipse.org/tools/gef/downloads/drops/R-3.0-200406251257/index.php
- To use Minerva, you need a RDBMS system for storage. DB2 8.2, or Apache Derby 10.0.2.1 (http://db.apache.org/derby/derby_downloads.html), or HSQLDB 1.8.0 (http://hsqldb.org/) is supported.

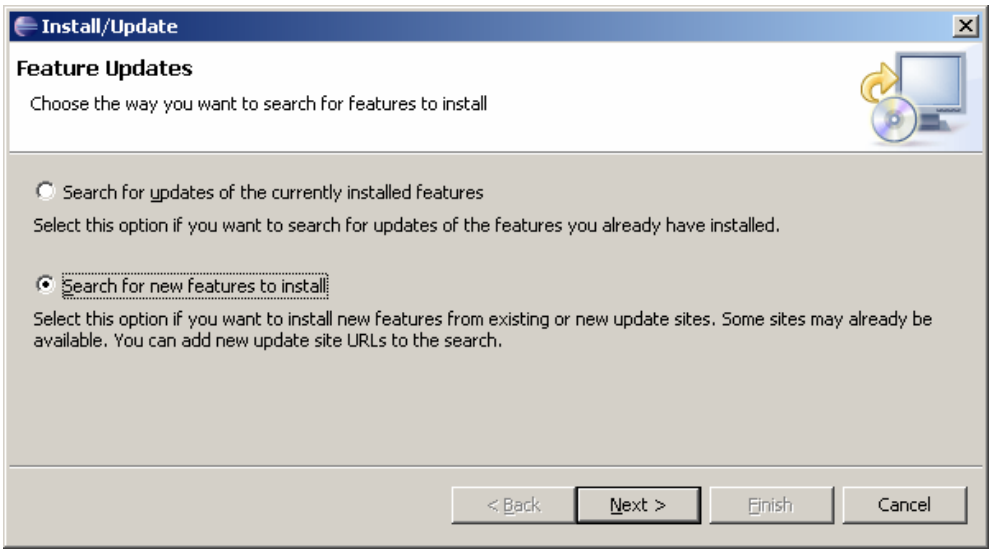
Note that JRE 1.5 does NOT work in some cases. We did not test IODT on other versions of the required components and IODT may not work on these untested versions.

Installation Instructions:

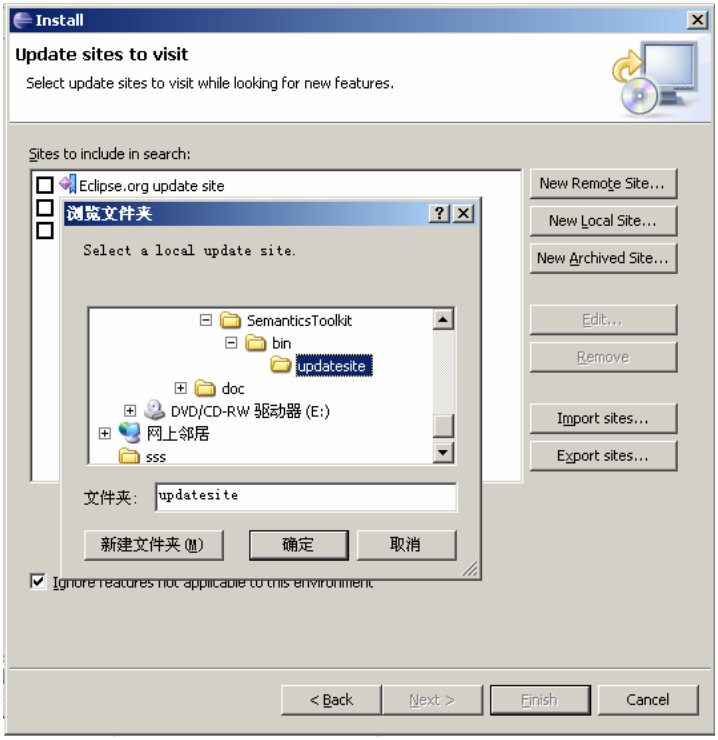
1. Download IODT-version.zip file from alphaWorks web site and unzip it to a directory. We refer to this unzipped directory as **$IODT**
2. Select Help -> Software Updates -> Find and Install …



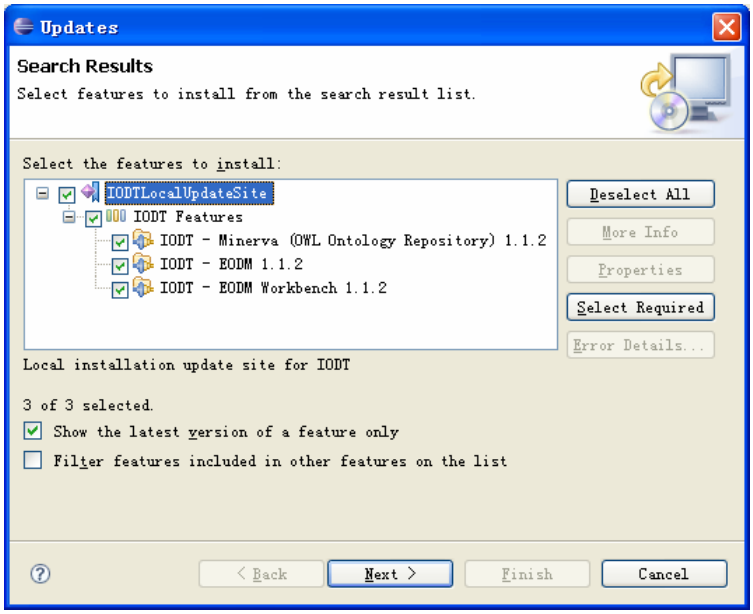
3. Select “Search for new features to install” and click “Next>”



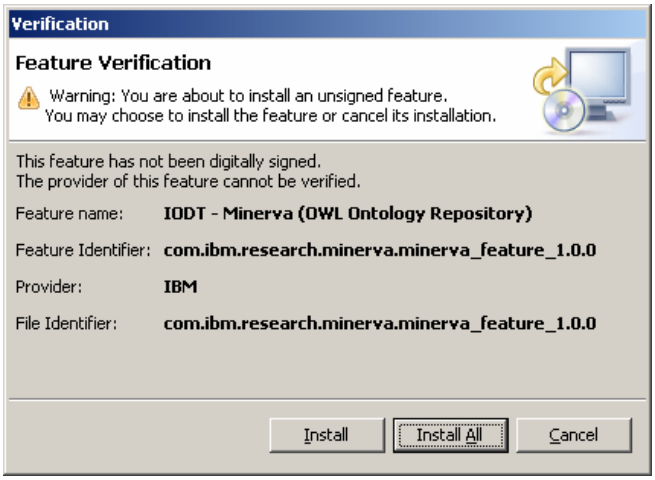
4. Select “New Local Site…”, choose the **$IODT \ bin \ updatesite** directory, confirm, input a name for the local update site, e.g.IODTUpdateSite and choose this site, click “Finish”.



5. In the new popped up dialog, you can see three features of IODT available for installation: EODM, EODM Workbench and Minerva. You can install any, some or all of them. Click “Next>”.



6. Read and accept the license of the release, and click “Next>”.
7. In the popped up dialog, features to install are listed, click “Finish”.
8. A warning dialog appears saying that the features are not signed. Please ignore the warning and click “Install All”.



9. You are recommended to restart the Eclipse workbench. Please do so. After the Eclipse is restarted, the **installation of EODM and EODM Workbench features (if you selected to install them) is successfully completed**. To complete Minerva installation, **you need perform two additional steps** to complete the installation.

10. An additional step for Minerva installation: Minerva need a RDBMS for storage and can use third-party TBox reasoning engines. You need download related JDBC drivers and JAR files for them. Please refer to the following tables for currently supported versions and download addresses of these third-party JARs and runtimes:

Software	Files and Download Links
For RDBMS	
IBM DB2 UDB Enterprise Server Edition Version 8.2, Type2 JDBC Driver	IBM DB2 UDB Tries and Betas Web Site: http://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2005-02-02+09:15:43.846099R&cat=database&fam=&s=c&S_TACT=104AHW42&S_CMP=
Apache Derby 10.1.2.1 and its JDBC Driver	http://db.apache.org/derby/releases/release-10.1.2.1.cgi derby.jar is included in db-derby-10.1.2.1-bin.zip
HSQLDB 1.8.0.2	Please download hsqldb_1_8_0_2.zip from HSQLDB web site and extract hsqldb.jar file from here: http://sourceforge.net/project/showfiles.php?group_id=23316&package_id=16653&release_id=339171
For TBox reasoners and SPARQL parser	
Log4j Version	http://logging.apache.org/log4j/docs/download.html
ANTLR 2.7.5	http://www.antlr.org/download.html
Pellet 1.2 and aterm 1.6	http://www.mindswap.org/2003/pellet/download.shtml "aterm-java-1.6.jar" is needed by Pellet inference engine and can be found in Pellet-1.2.zip.
Racer and DIG 1.1	Download Racer from: http://www.racer-systems.com/ Extract the following JARs from dig1.1-reasoners-bin.zip downloadable from sourceforge web site here: http://sourceforge.net/project/showfiles.php?group_id=107632&package_id=116214: dig1.1-reasoners.jar, dig1.1-xmlbeans.jar, jaxen.jar, xbean.jar, xmlpublic.jar

For RDBMS, you only need one of the above three. For third-party TBox reasoners, you can use Pellet, Racer or our built-in structural TBox reasoning engine.

11. After you downloaded related files from the above, please rename the JAR files according to the following table and copy them to the Minerva plugin's library directory located at **$ECLIPSE_HOME \ plugins \ com.ibm.iodt.sor_1.1.2 \ lib**.

Note that you should rename these libraries to the following names to make them be recognized by Minerva Plugins. Otherwise, the plugin may not be installed correctly.

DB2 Type2 Driver	db2java.jar
Derby Driver	derby.jar
HSQL DB Driver	hsqldb.jar
Log4j Driver	log4j.jar
ANTLR	antlr-2.7.5.jar
Pellet Driver	pellet.jar aterm-java-1.6.jar
DIG Interface	dig1.1-reasoners.jar, dig1.1-xmlbeans.jar; jaxen.jar; xbean.jar; xmlpublic.jar

Now, Minerva installation is completed. But before you can use it, please refer to the Minerva configuration section to configure Minerva properly on its use of RDBMS and TBox reasoning engines. .

Installation for use outside Eclipse

To use EODM, you need include the following JARs in CLASSPATH:

- eodm-1.1.2.jar (in the **$IODT \ bin** directory)

- emf.ecore_2.1.0.jar
- emf.common_2.1.0.jar
- emf.ecore.xmi_2.1.0.jar

The above three EMF-related JARs can be found in the following EMF package:

emf-sdo-xsd-Standalone-2.1.0.zip

which can be downloaded from
http://fullmoon.torolab.ibm.com/tools/emf/scripts/download.php?dropFile=../downloads/drops/2.1.0/R200507070200/emf-sdo-xsd-Standalone-2.1.0.zip

To use Minerva, in addition to all the above four JAR files, you need include the following additional JARs in CLASSPATH:

- minerva-1.1.2.jar (in the **$IODT \ bin** directory)
- Minerva need a RDBMS for storage and can use third-party TBox reasoning engines. You need download related JDBC drivers and JAR files for them. Please refer to the following tables for currently supported versions and download addresses of these third-party JARs and runtimes:

Software	Files and Download Links
For RDBMS	
IBM DB2 UDB Enterprise Server Edition Version 8.2, Type2 JDBC Driver	IBM DB2 UDB Tries and Betas Web Site: http://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2005-02-02+09:15:43.846099R&cat=database&fam=&s=c&S_TACT=104AHW42&S_CMP=
Apache Derby 10.1.2.1 and its JDBC Driver	http://db.apache.org/derby/releases/release-10.1.2.1.cgi derby.jar is included in db-derby-10.1.2.1-bin.zip
HSQLDB 1.8.0.2	Please download hsqldb_1_8_0_2.zip from HSQLDB web site and extract hsqldb.jar file from here: http://sourceforge.net/project/showfiles.php?group_id=23316&package_id=16653&release_id=339171
For TBox reasoners and SPARQL parser	

Log4j Version	http://logging.apache.org/log4j/docs/download.html
ANTLR 2.7.5	http://www.antlr.org/download.html
Pellet 1.2 and aterm 1.6	http://www.mindswap.org/2003/pellet/download.shtml "aterm-java-1.6.jar" is needed by Pellet inference engine and can be found in Pellet-1.2.zip.
Racer and DIG 1.1	Download Racer from: http://www.racer-systems.com/ Extract the following JARs from dig1.1-reasoners-bin.zip downloadable from sourceforge web site here: http://sourceforge.net/project/showfiles.php?group_id=107632&package_id=116214: dig1.1-reasoners.jar, dig1.1-xmlbeans.jar, jaxen.jar, xbean.jar, xmlpublic.jar

For RDBMS, you only need one of the above three. For third-party TBox reasoners, you can use Pellet, Racer or our built-in structural TBox reasoning engine.

After you downloaded related files from the above, please rename the JAR files according to the following table and include them in your CLASSPATH. **Note that you should rename these libraries to the following names to make them be recognized by Minerva. Otherwise, Minerva may not work.**

DB2 Type2 Driver	db2java.jar
Derby Driver	derby.jar
HSQL DB Driver	hsqldb.jar
Log4j Driver	log4j.jar
ANTLR	antlr-2.7.5.jar
Pellet Driver	pellet.jar aterm-java-1.6.jar
DIG Interface	dig1.1-reasoners.jar, dig1.1-xmlbeans.jar; jaxen.jar; xbean.jar; xmlpublic.jar

Now, Minerva installation is completed. But before you can use it, please refer to the Minerva Configuration section to configure Minerva properly on its use of RDBMS and TBox reasoning engines. .

EODM User Guide, Tutorial and Examples

Description of Important Packages

* org.eclipse.eodm.rdfs / org.eclipse.eodm.rdfs.impl
  interface of RDFS Model / implementaion of RDFS Model
* org.eclipse.eodm.owl / org.eclipse.eodm.owl.impl
  interface of OWL Model / implementation of OWL Model
* org.eclipse.eodm.rdf.resource.parser / org.eclipse.eodm.owl.resource.parser
  EODM RDF Parser / EODM OWL Parser
* org.eclipse.eodm.rdfs.reasoner / org.eclipse.eodm.rdfs.reasoner.impl
  EODM RDF Reasoner Interface & Implementation
* org.eclipse.eodm.owl.reasoner / org.eclipse.eodm.owl.reasoner.structural /
  org.eclipse.eodm.owl.reasoner.simple
  EODM OWL Reasoner Interface and serveral Implementations
* org.eclipse.eodm.rdfs.transformer.ecore / org.eclipse.eodm.owl.transformer.ecore
  EODM Transformation with Ecore.

Creating a simple OWL model

We will start from the the basics: creating an OWL model and adding OWL classes and properties into it. The example will show how to create a simple OWL ontology by using EODM API.

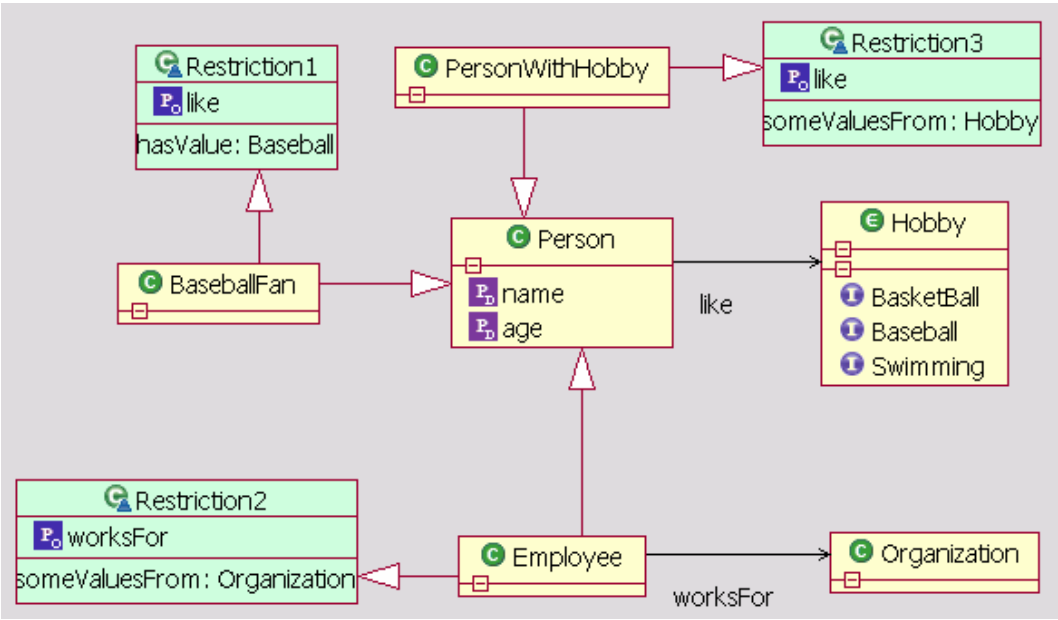


Fig 1

In this simple example, we will create a simple ontology about person. As illustrated in Figure 1, we will create an OWL class “Person” with two attributes: “name” and “age”; “Employee” is subclass of “Person”, which is defined as someone who works for some organization…

In EODM, all the OWL resources are created through OWLFactory. In the example, we will firstly create an empty ontology by calling OWLFactory.createOWLOntology(). When the ontology is created, OWL reasoures can be created by calling relevant methods of OWLFactory, as shown in Table 1.

```
// obtain the OWL factory and create the ontology
OWLFactory factory = OWLFactory.eINSTANCE;
ontology = factory.createOWLOntology();

//create a namespace and set it as namespace of the ontology
Namespace testNS = RDFSFactory.eINSTANCE.createNamespace(ontology,"test","http://test.org/test#");
ontology.setNamespace(testNS);

// create owl classes
OWLClass Person = factory.createOWLClass(ontology,"Person");
OWLClass Employee = factory.createOWLClass(ontology,"Employee");
OWLClass PersonWithHobby = factory.createOWLClass(ontology," PersonWithHobby");
OWLClass BaseballFan = factory.createOWLClass(ontology," BaseballFan");
OWLClass Org = factory.createOWLClass(ontology,"Organization");
OWLClass Hobby = factory.createOWLClass(ontology,"Hobby");

// define elements for Hobby
Individual basketball = factory.createIndividual(ontology,"BasketBall", Hobby);
Individual baseball = factory.createIndividual(ontology,"BaseBall", Hobby);
Individual swimming = factory.createIndividual(ontology,"Swimming", Hobby);

// create object properties & define their domain/range
OWLObjectProperty worksFor = factory.createOWLObjectProperty(ontology,"worksFor", Person, Org);
OWLObjectProperty like = factory.createOWLObjectProperty(ontology,"like", Person, Hobby);

// create datatype properties & define their domain/range
Namespace XSDNS = RDFSFactory.eINSTANCE.createNamespace(ontology,
                                XSD.SHORTNAME, XSD.NAMESPACE);
RDFSDatatype XSDINT = RDFSFactory.eINSTANCE.createRDFSDatatype(ontology,XSD.sxint, XSDNS);
RDFSDatatype XSDSTR = RDFSFactory.eINSTANCE.createRDFSDatatype(ontology,XSD.sxstring, XSDNS);
OWLDatatypeProperty name = factory.createOWLDatatypeProperty(ontology,"name", Person, XSDSTR);
OWLDatatypeProperty age = factory.createOWLDatatypeProperty(ontology,"age", Person, XSDINT);

// build up relationship between owl classes
HasValueRestriction Restriction1 = factory.createHasValueRestriction(ontology, like, basketball);
BaseballFan.getRDFSSubClassOf().add(Person);
BaseballFan.getRDFSSubClassOf().add(Restriction1);
SomeValueFromRestriction Restriction2 = factory.createSomeValueFromRestriction(ontology, worksFor, Org);
Employee.getRDFSSubClassOf().add(Person);
Employee.getRDFSSubClassOf().add(Restriction2);
```

```
SomeValueFromRestriction Restriction3 = factory.createSomeValueFromRestriction(ontology, like, Hobby);
PeopleWithHobby.getRDFSSubClassOf().add(Person);
PeopleWithHobby.getRDFSSubClassOf().add(Restriction3);

// create individuals for OWL class
Individual john = factory.createIndividual(ontology,"John", Person);
Individual andy = factory.createIndividual(ontology,"Andy ", BaseballFan);
```

Table 1 Creating an ontology to describe person

Navigate an OWL ontology

To navigate an OWL ontology, it is primarily through the Ontology interface and OWL resource interface. For example, to get all instances of OWLClass "Person", we can firstly call the Ontology.getContainedResource (Namespace namespace, String localName) to retrieve a reference of the OWLClass "Person" in Ontology. And then the instances of "Person" can be retrieved by calling the getInstance() method of "Person". Table 2 shows a list of codes of how to navigate an ontology.

(Note: the return results only include those declared facts, which means it will not return any results by inference)

```
// get sub class of an OWL class
OWLClass person = ontology. getContainedResource(testNS ,"Person");
List l = person.getSubClass();

// get instances of an OWL class
…
List employeeList = employee.getInstance();

// get domain of OWL property
…
List domainList = like.getRDFSDomain();

…
```

Table 2 Navigate an OWL ontology

Save OWL ontology to file

In many cases, we need to serialize ontology to file. This can be done by calling OWLXMLSaver.saveToFile(OWLOntology ontology, String fileName, String charset) method.

```
// Save an OWLOntology
try {
    OWLXMLSaver.saveToFile(ontology, "./person.owl", "UTF-8");
} catch (IOException e1) {
    e1.printStackTrace();
}
```

Table 3 Save ontology to file

Load OWL ontology from file

There are 2 major steps to load ontology into memory model. The first step is to create an *OWLDocument* instance which describes information of ontology files and then add to parser. The *OWLDocument* constructor takes the *localURI* and the *publicURI* to locate the document. User can specify whether or not to load the file from local by using the *localFile* parameter. The second step is to parse ontology by calling the OWLParser.parseOWLDocument (OWLDocument) methods. Table 3 gives a detail example of how to load OWL ontologies.

```
// New an OWL Parser
OWLParser parser = new OWLParserImpl();

// New an OWL Document and add to parser
OWLDocument food = new OWLDocumentImpl("./food.owl", null, true);
parser.addOWLDocument(food);

OWLDocument wine = new OWLDocumentImpl(null,
                    "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine", false);
parser.addOWLDocument(wine);

// Start to parse OWL documents and get OWL ontologies
OWLOntology ontofood = parser.parseOWLDocument(food);
OWLOntology ontowine = parser.parseOWLDocument(wine);
```

Table 4 Loading OWL Ontology

Reuse existing models in other modeling languages

Compared with others tools, the most important differentiation of EODM is its ability of bi-direction transformation between RDF/OWL model and Ecore Model, which gives EODM the interoperability between RDF/OWL model with other EMF supported models, including models defined using XML Schema, Rational Rose, and annotated Java classes. Leveraging EMF and EODM transformer, Semantic Web Application developers can reuse existing models that are represented in

other model languages.

Table 5 shows a simple example of how to transform an Ecore model into an OWL ontology.

```
// new a Ecore to OWL transformer
ECore2OWLTransformer ecore2owl = new ECore2OWLTransformer();

try {
        // convert Ecore model to OWL
        ecore2owl. convertEcore2OWL ("./person.ecore","./person.owl");
}
catch (Exception e) {
        e.printStackTrace();
}
```

Table 5 Transform Ecore Model to OWL Ontology

Inference with EODM Reasoner

The inference ability of OWL is one important distinct feature from other modeling language, such as UML. EODM also provides a reasoner to support inference. Currently EODM reasoner only supports taxonomy subsumption reasoning for OWL reasoning. In other words, EODM reasoner only supports reasoning about relationships between OWL classes and OWL properties.

```
// creating a reasoner
OWLTaxonomyReasoner reasoner = StructuralReasonerFactory.instance()
                        .createOWLTaxonmyReasoner();

// initialize reasoner
reasoner.initialize(ontology);
...
// get Descendant Classes
List childClass = reasoner.getDescendantClasses(c1);

// get Ancestor Property
List plist = reasoner.getAncestorProperties(p1);

// test whether two OWL class are subsumed
boolean isSubClass = reasoner.isSubClassOf(c1, c2);
```

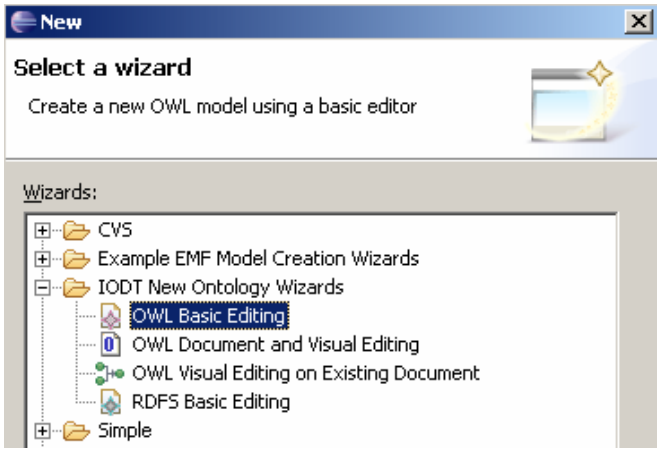
Table 6 Inference with EODM Reasoner

Example Code

The **$IODT \ examples** directory contains some Java code examples for using the EODM API. You can know what the example is about from the name of the Java files.

EODM Workbench Tutorial

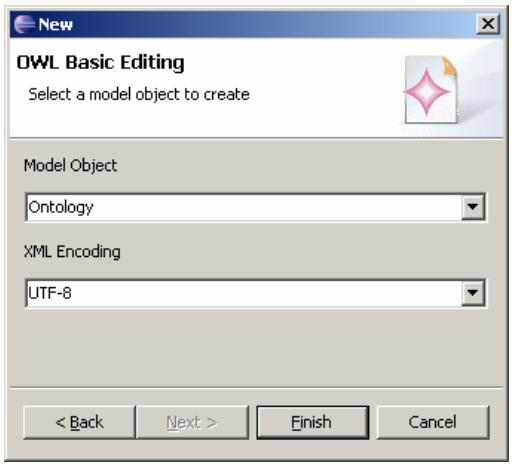
From Eclipse menu, select File -> New -> Other, in the popped up dialog, there is a "**IODT New Ontology Wizards**" category under which there are four editing tools. Two of them are for OWL visual editing and two for RDF/OWL basic editing.



The following two sub-sections will introduce you to the basic editing and visual editing respectively.

RDFS/OWL Basic Editing

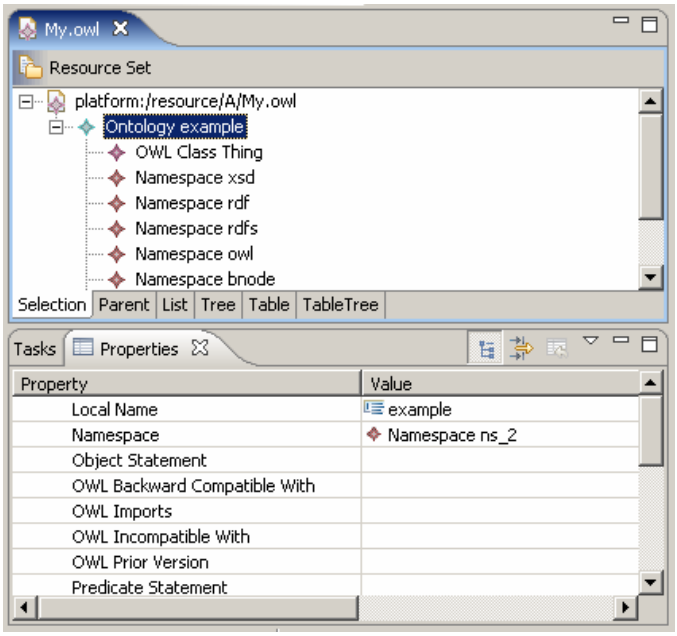
To create a new RDFS/OWL model and do some basic editing, select File->New->Other, select "OWL Basic Editing" or "RDFS Basic Editing" under the "IODT New Ontology Wizard" category. We select "OWL Basic Editing" as an example to show how basic editing can be used. In the following-up dialog, input the file name of the new OWL model and the folder it should be put. Click "Next>", in the next dialog, select "Ontology" as the model object, as shown below:



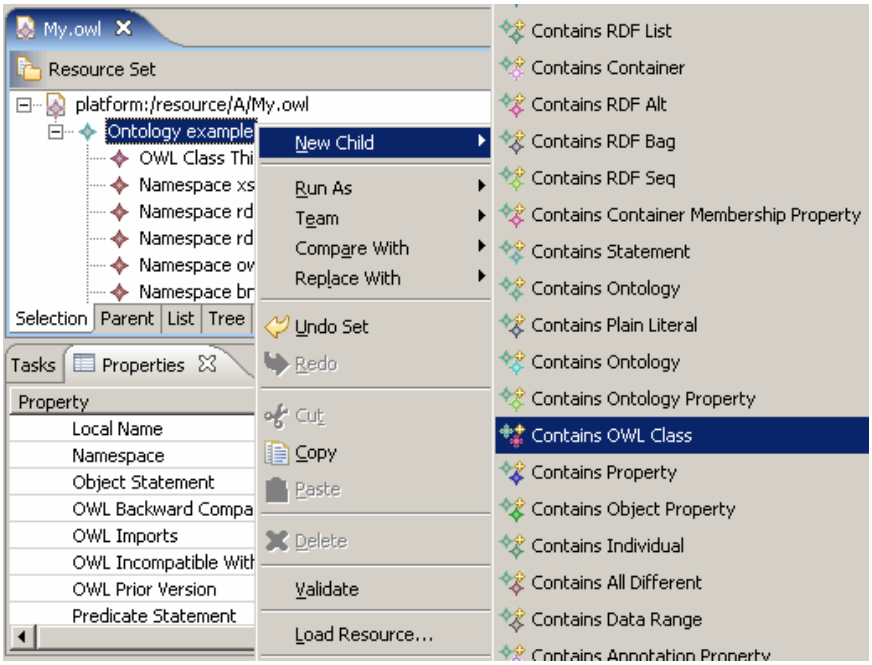
Please select "Ontology" as the model object; it will be used as the container for other model

objects. Otherwise, you may not be able to create some model objects.

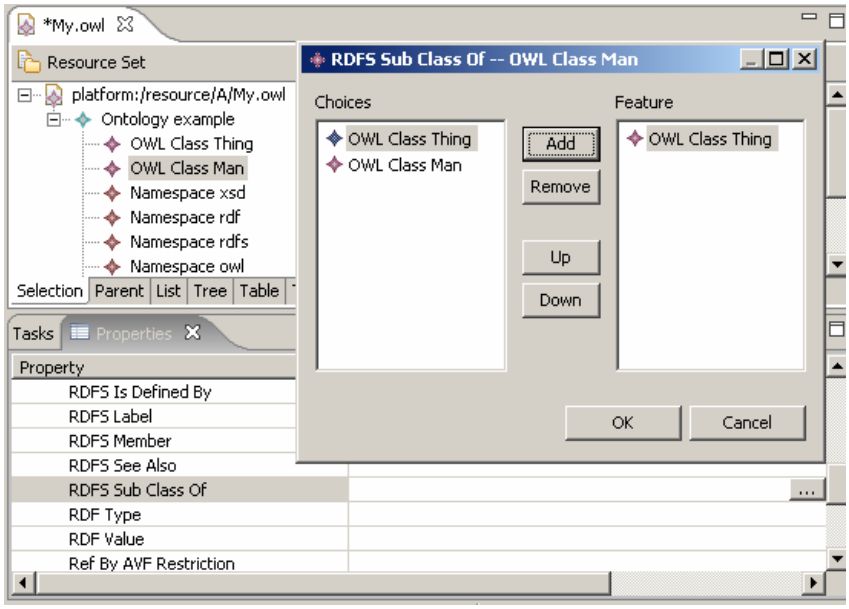
Click “Finish”. A basic editor appears in the workspace. Please also open the standard “Properties” view in Eclipse if the view is not shown yet. Click the “Ontology OWLInputStream” node in the tree, the Properties view then list all properties of the ontology resource. Click the value of the “Local Name” and type in any local name you want for the ontology resource. In this example, we give it an “example” local name, as shown below:



Right-click the “Ontology example” node, select “New Child” -> “Contains OWL Class” to create a new OWL Class. Edit the “Local Name” and other properties of the newly created class. In this example, we give it a local name “Man”.

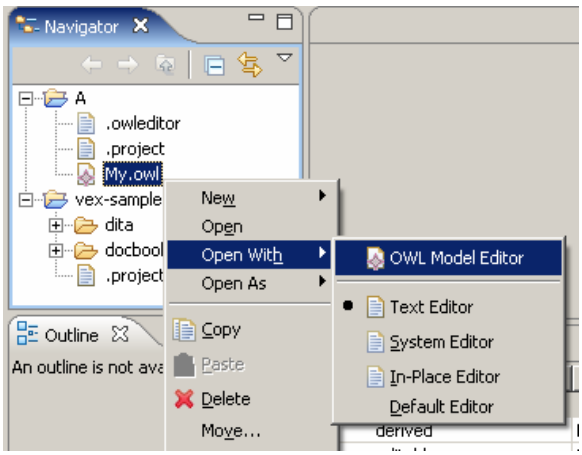


In the Properties view of this “Man” class, click the value of the “RDFS Sub Class Of” property, click the “…” button on the right hand, a dialog appears that let you choose the super-classes of the “Man” class. Select “Thing” and click “Add” to add it to “Man”’s super-class. As the following image shows:



Hence, the basic editing of RDFS/OWL is based on a tree of model objects and the editing of properties in the standard “Properties” view. Based on the above example of creating a class and adding a sub-class relationship, you can try use the editor to create more model objects and editing other properties. You will soon get familiar with this simple UI.

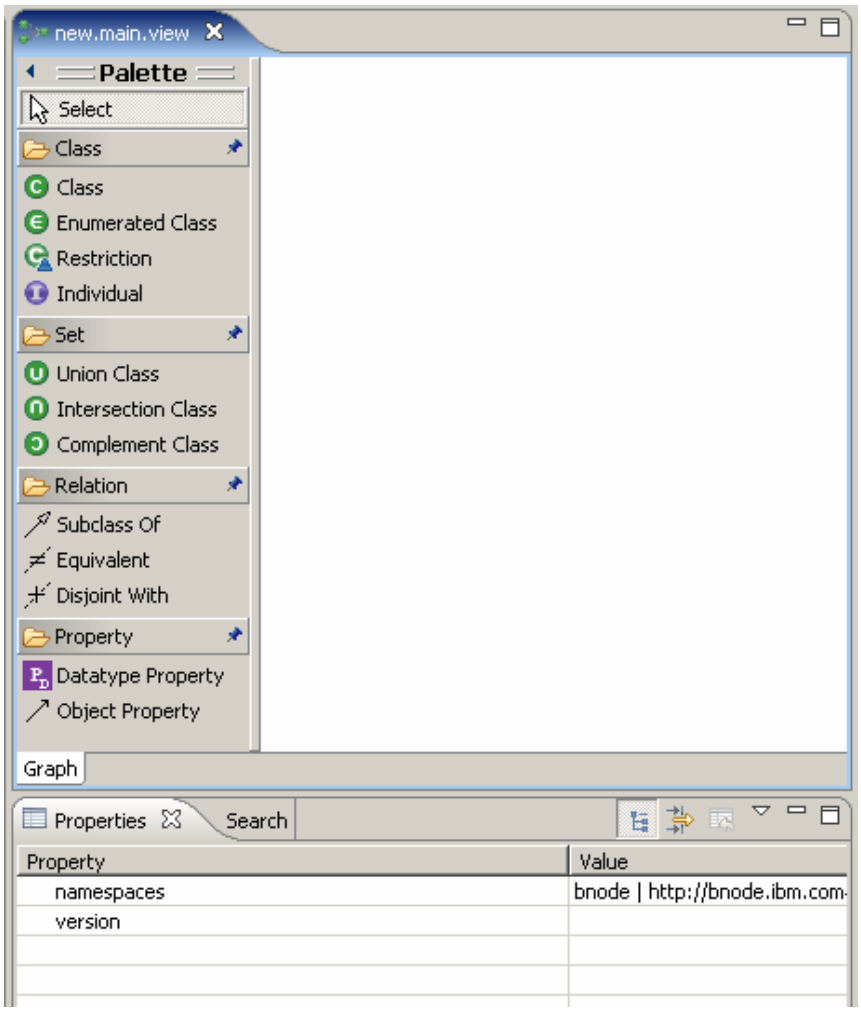
After editing, you can close the editor and save the changes made in the file you created. The ontology will be saved in RDF/XML format. By right-clicking a RDF(S) or OWL model in RDF/XML format file in Eclipse, and selecting “Open With”-> “OWL Model Editor” (or “RDFS Model Editor”), you will be able to open the same simple editor for editing the RDF(S) or OWL ontology, as shown in the following:



If you select to open with “Text Editor”, you will be able to view the file directly in the textual RDF/XML format and edit it just like any textual document.

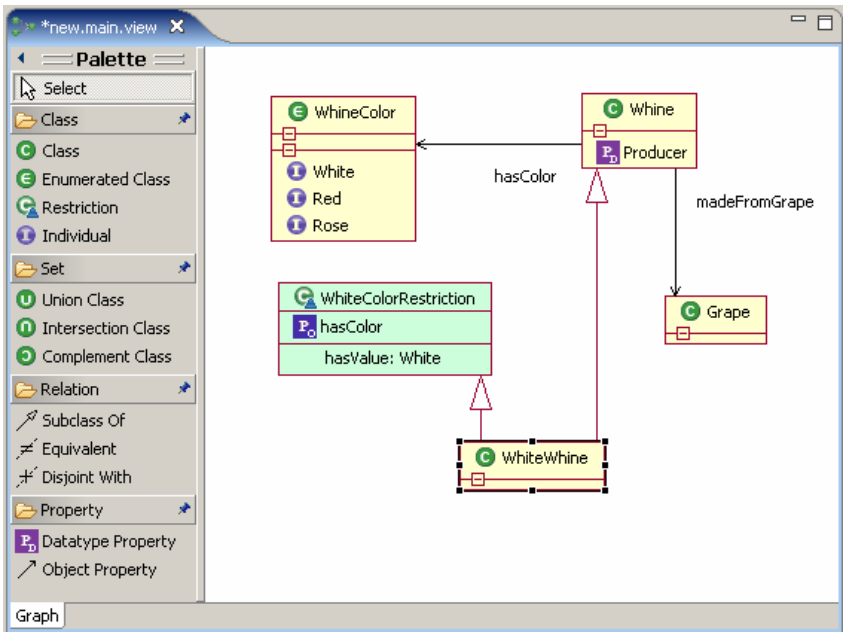
OWL Visual Editing

From Eclipse menu, select File -> New -> Other, in the popped up dialog, there is a "**IODT New Ontology Wizards**" category, select "**OWL Document and Visual Editing**", click "Next>", choose a file name and directory to save the new OWL document, click "Next". You can configure a set of namespaces for the OWL document. Click "Finish". A visual editor is opened in the workspace, with a widget palette at the left-hand of the editor. Please open the standard "Properties" view of Eclipse if it is not opened. The workspace should look like:



Initially, the Properties view shows the properties of the current ontology. During the editing, you can always click on the white canvas to select the ontology and its properties.

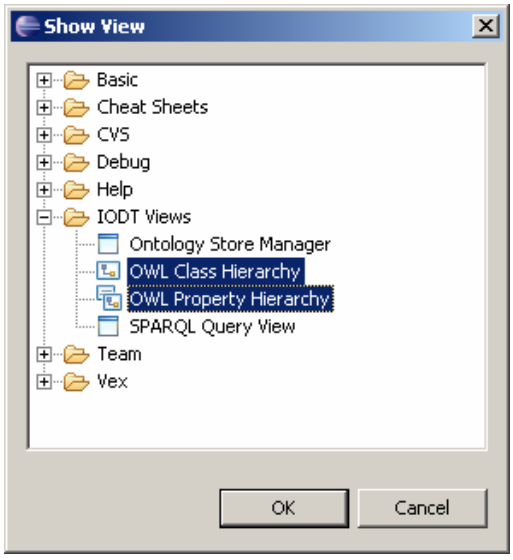
Clicking on the icons at the left-hand palette to create OWL model elements. For relations and properties, after click the icon on the palette, you need click on a class on the canvas, move the pointer to another class and click again. This will create a relation or property between the two classes. The following figure shows a diagram after some time of editing:



In this documentation, we highlight some of the functions of this visual editor. Users are encouraged to try use this UI to get familiar with its usage.

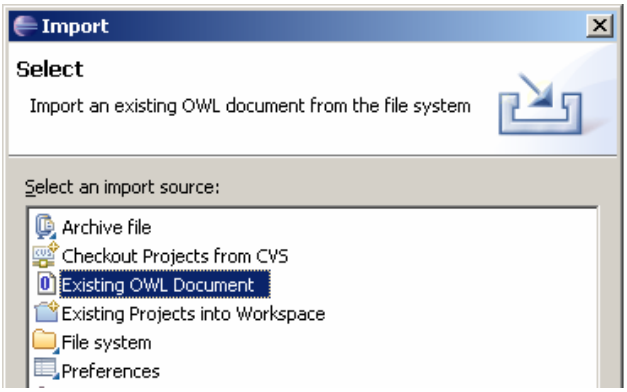
Class and Property Hierarchies View

From Eclipse menus, select "Window->Show View->Other", in the "IODT Views" catalog, there are "OWL Class Hierarchy" and "OWL Property Hierarchy" views respectively.

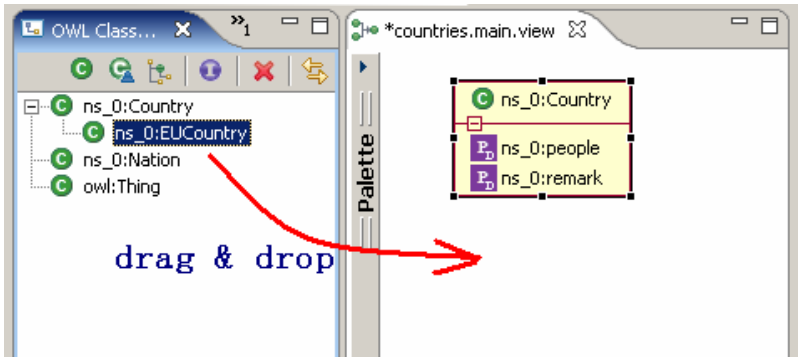


Import an Existing OWL document for visual editing

From Eclipse menus, select "File -> Import". In the import dialog, select "Existing OWL Document" and click "Next" to import an existing OWL document in the file system.



After the document is imported, a blank canvas is shown. You can open "OWL Class Hierarchy" view and drag classes from the view to the canvas to visualize the ontology.



Create Datatype Property

Select "Datatype Property" icon on the left palette and click on a existing class on the canvas, this will put the new datatype property in the clicked class. The semantics is that the property will have this clicked class as its domain. You can then select the data type by clicking on it and edit its range property in the Properties view.

Create an Enumerated Class

Select "Enumerated Class" icon on the left palette and click on the canvas will create an initial enumerated class in the ontology. You can editing its various properties in the properties view or click its name to change its local name, etc. To enumerate its individuals, you need click the "Individual" icon, move the pointer to the enumerated class and click again. This will bring up a dialog to give a local name to the individual. The individual is then one of the instance of the enumerated class.

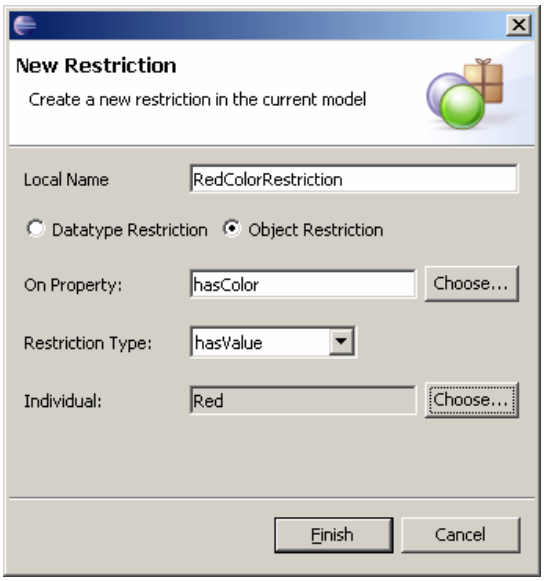
Note that, currently, the tool can not create individuals for other types of classes.

Create Union, Intersection and Complement Class

You need click the respective icon on the left palette and click on the canvas to create the class. After that, you need edit the “union of”, “intersection of”, and “complement of” property in the Properties view. This will bring a dialog to allow you to choose existing classes in the ontology.

Create Restriction Class

Click on the “Restriction” icon on the left palette and click on the canvas, this will bring up the following dialog to allow you to specify in detail the restriction:

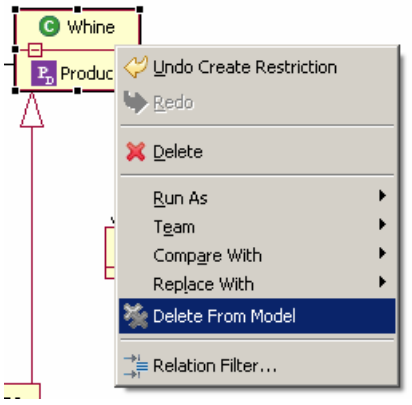


Delete a Model Element from View

Select a model element in the canvas and press the “Delete” key will delete it from the view but it is still in the model.

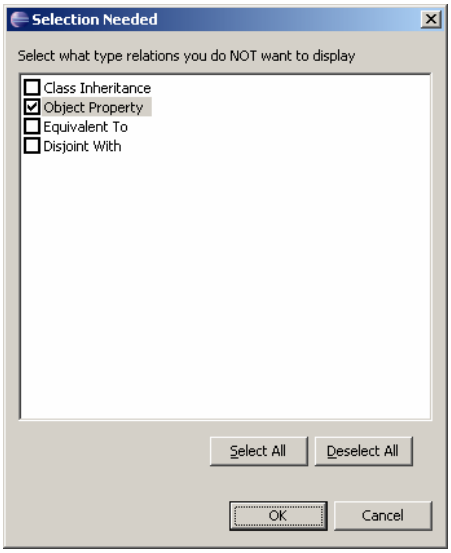
Delete a Model Element from Model

Select a model element in the canvas, right click it and select “Delete From Model”.

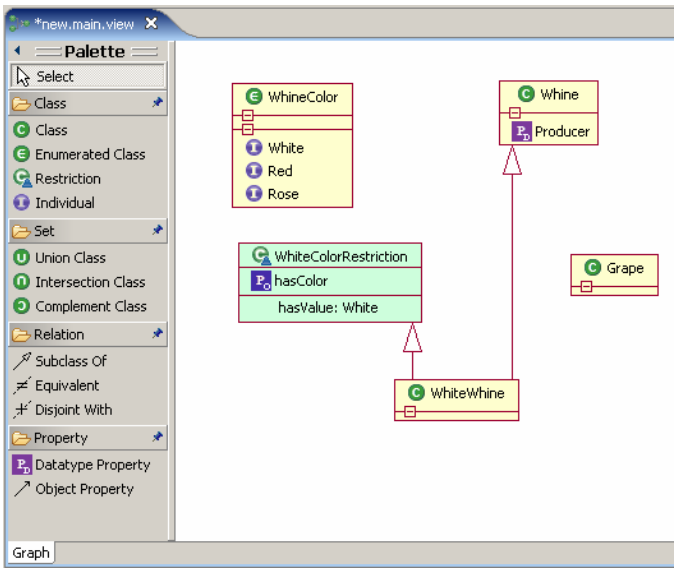


Relation Filter

You can choose what types of relations should be shown on the canvas. Right-click on the white space of the canvas and choose “Relation Filter…”. This will bring up a dialog to let you choose types of relations to hide.

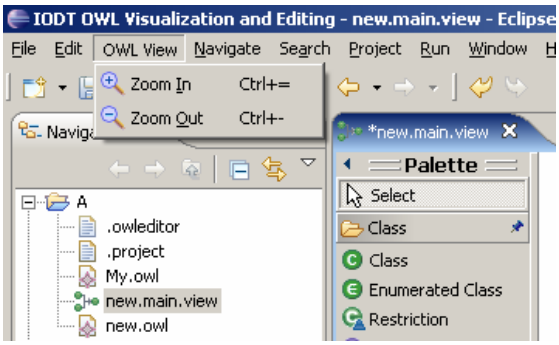


If we choose “Object Property”, for example, the example diagram’s object properties (madeFromGrape and hasColor) among classes will not be shown. The result diagram then becomes the following (where only sub-classes relations are shown):



Zoom In/Out

Please select menu “OWL View” -> “Zoom In” or “Zoom Out” or use short-cut keys for zooming.



Limitations of OWL Visual Editing

What EODM workbench CANNOT Do

- NO SET operators support, such as intersectionOf, unionOf and complementOf
- NO instance editing capabilities, except for oneOf
- NO imports support for multiple ontologies
- NO drag support for arcs in the canvas

Minerva Configuration, User Guide and Examples

Minerva Configuration

You can configure Minerva to meet your application requirements. Following parameters are currently configurable.
The configuration parameters are defined as follows (case sensitive).

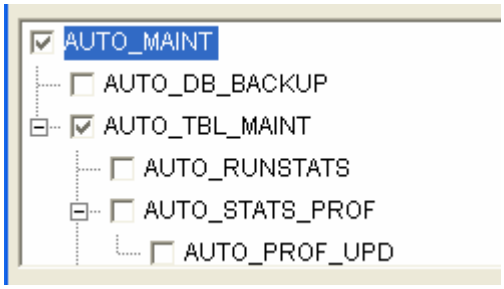
Items	Descriptions
Database = jdbc:db2:Minerva	The database used as the backend store
User = db2admin	The user name for accessing database
Password = xxxxxx	The password for the user
Driver = COM.ibm.db2.jdbc.app.DB2Driver	DB2 JDBC driver
TboxEngine = Pellet (Values can be Default or Racer)	Currently, Minerva supports three Description Logic inference engines, The structural TBox engine, Racer and Pellet. The user can specify preferred engine. The default engine is the structural TBox engine.
RacerIP = 172.16.87.56	If Racer is used, please provide IP address of Racer HTTP server.
Port = 8080	The port of Racer HTTP server
DuplicateRemover = false	When inserting duplicate triples into the repository, there are two methods to remove them. The first way is that Minerva removes duplicates. The second is that DBMS removes duplicates using primary key / unique index constraints. If DuplicateRemover is set to "true", Minerva will be responsible for duplicate removing. Otherwise, DBMS will do. If the latter, the speed of loading data is faster but database exceptions will be thrown out. The user can ignore these exceptions which do not cause any errors.

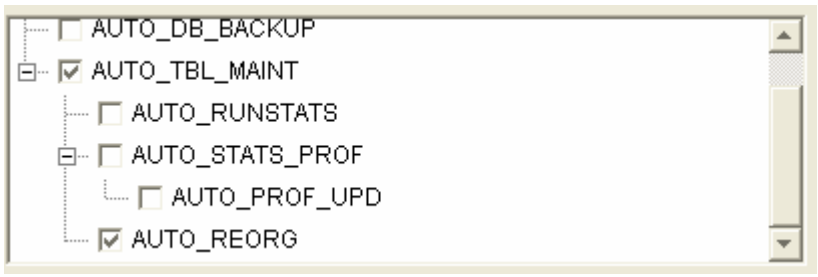
In the **$IODT \ examples \ configuration** directory, there are three sample configuration files corresponding to Derby, DB2 and HSQL DB, respectively.

If you are using Minerva in Eclipse, please go to the Minerva plugin directory located at **$ECLIPSE_HOME \ plugins \ com.ibm.iodt.sor_1.1.2 \ configuration** and specify the configuration parameters in file "minerva.cfg". In this directory, there are also three sample configuration files corresponding to Derby, DB2 and HSQL DB, respectively. **Restart your Eclipse after configuration.**

DB2 is powerful but needs to be well configured for optimal performance. Following configuration parameters have been tested in experiments for storing large volume of OWL ontologies..

1. LOGFILSIZ = 10000
2. LOGPRIMARY = 20
3. Maintenance
   a) AUTO_REORG: CHECK
   b) AUTO_TBL_MAINT: CHECK
   c) AUTO_MAINT: CHECK





4. Performance Parameters
   - BUFFPAGE: 2500
   - SORTHEAP: 25600
   - SHEAPTHRES_SHR: 256000

Sample Code

Following describes how to build and manipulate an OWL ontology Repository step by step. Please refer to the "MinervaQueryDemo.java" and "MinervaStorageDemo.jave" files in the **$IODT \ examples** directory.

1. Specify the directory which includes the configure file.

```
Config.setConfigFile(new FileInputStream ("D:/eclipse/workspace/Minerva/minerva.cfg"));
```

2. Install an ontology repository which can store multiple ontologies. Just need to install a repository once for a database

```
Config.install();
```

3. Construct an ontology store managers which manage all ontologies in the built knowledge base.

```
OntologyStoreManager minerva = new OntologyStoreManager();
```

4. New, select, delete an ontology store

```
//deleting an ontology store menas deleting all statements in the store
```

```
minerva.deleteOntologyStore("os2");
//need to new an ontology store before adding ontology into Minerva
minerva.newOntologyStore("os2");
//need to select an ontology store to manipulate
minerva.selectOntologyStore("os2");
```

5. Add OWL ontologies to the selected ontology store.

```
minerva.addOWLDocument("test/univ-bench-dl.owl", false);
//need to specify file name and whether the file includes only ABox statements
//public boolean addOWLDocument(String fileName, boolean bPureABox)
```

6. Put an EODM model into the selected ontology store.

```
minerva.addEODMModel(ontofood,false);
//need to specify model name and whether the model includes only ABox statements
//public boolean addEODMModel(OWLOntology onto, boolean bPureAbox)
```

7. Explicitly notify Minerva to do inference. Adding owl files into Minerva is only to insert statements into the ontology store and does not conduct inference. Users need to explicitly notify Minerva to do inference.

```
minerva.commit();
```

8. Minerva can do update with transaction support in two working modes. The first is to add or delete an assertion every time. If minerva failed to update, no "dirty" assertions inferred from the input will be inserted into repository. If minerva succeeded to update, all inferred triples will inserted. In this working mode, commit is completed automatically. Users do not need to commit manually. For example,

```
minerva.addTypeOfAssertion("http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#StMic
helleMuscadet", "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Muscadet");
minerva.deleteObjectRoleAssertion("http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#
StMichelleMuscadet","http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#hasMaker4","
http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Marietta");
```

The second working mode is group mode. Users can add or delete a group of assertions every time. If failed to update, no an assertion from the input group can be inserted into repository. If succeeded, all assertions will be inserted. This means that atomic operation in group mode is on a group of assertions, instead of an assertion. For example,

```
/*add a group of assertions into ontology store*/
//1. set working mode to group mode. the step is a must
minerva.setGroupCommitment();
//2. add or delete assertions
minerva.addTypeOfAssertion("http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#StMic
helleMuscadet", "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Muscadet");
minerva.addObjectRoleAssertion("http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#St
MichelleMuscadet","http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#hasMaker4","htt
p://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Marietta");
minerva.addDatatypeRoleAssertion("http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#
StMichelleMuscadet","http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#hasMaker6","
Marietta","en",null);
```

```
//3. commit a group of assertions, true means "commit", flase means "rollback"
    minerva.commitAssertions(true);
```

9. Retrieve statements using SPARQL query and traverse them

```
//public ResultHolder getQueryResult(String query)
ResultHolder result = minerva.getQueryResult("select * where (?x rdf:type ?y) ")
String[] name=result.getVariablename();
    for (i =0; i<name.length;i++)
        System.out.print(name[i]+"        ");
    System.out.println();
    while (result.next()){
        for (i =0; i<name.length;i++)
            System.out.print(result.getInt(i+1)+"      ");
        System.out.println();}
```

10. Issue a query and get triple list rather than variable-value pairs

```
List tripleList=minerva.getTripleList(q3);
for(Iterator i=tripleList.iterator();i.hasNext();){
    String[] res = (String[])i.next();
    for(int j=0;j<3;j++)
        System.out.print(res[j] + " ");
    System.out.println(); }
```

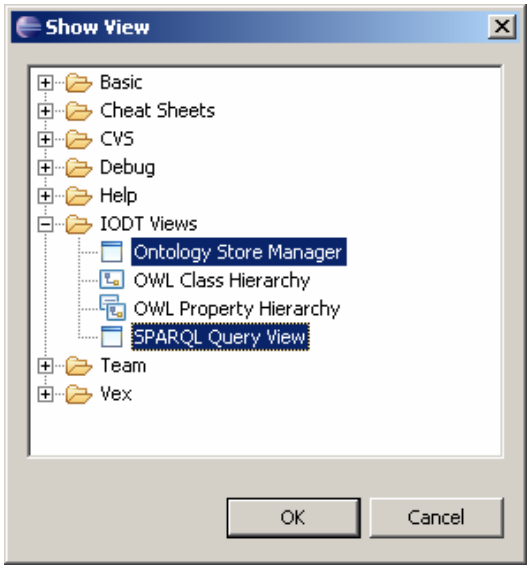
11. Close minerva and all connections

```
minerva.closeConnection() ;
```

Minerva Eclipse Plugin User Guide

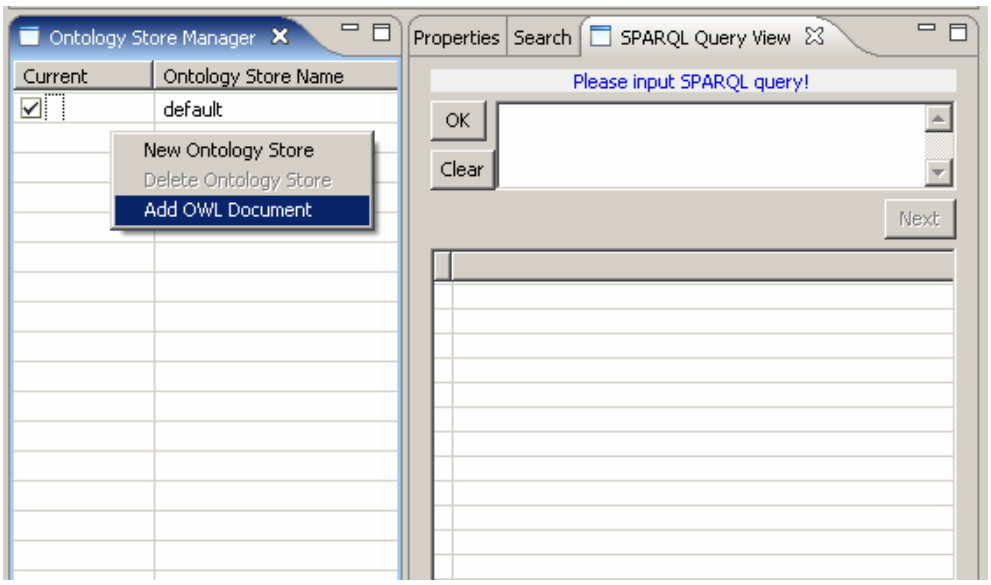
Users can use Minerva in their applications by calling provided APIs. For convenience, an Eclipse Plugin is provided. The following shows how to load and query ontologies using the Eclipse plugin.

1. From Eclipse menu, select “Window -> Show View ->Other” to open a dialog and select the “Ontology Store Manager” and “SPARQL Query views” in the “IODT Views” category and click “OK”:



If you get errors when opening the two views, you may not have put the required JAR files into the Minerva plugin’s lib directory or have edited the configuration file , please see the installation section on how to do that.

Drag and put the two views in your preferred position in Eclipse.



2. .In the Ontology Store Manager view, you can right click an ontology store to create a new ontology store, delete an ontology store, and add OWL documents into the selected store. It may take a long time to add large OWL documents into the repository, depending on the size, complexity of the ontology and the backend RDBMS's performance.

3. After add OWL documents into the ontology store, you can input SPARQL query in the SPARQL Query View to search the selected ontology store. Please click next button to browse all results.