

Working with Edutella

Daniel Olmedilla

L3S Research Center and Hanover University
Hanover, Germany

olmedilla@l3s.de

Contents

1	Introduction	2
2	Using Edutella classes to build a wrapper	2
2.1	Parsing a QEL Query	2
2.1.1	Class Query	3
2.1.2	QELQueryParser	3
2.1.3	DatalogQueryParser	3
2.1.4	Example	4
2.2	Adding Results	4
2.2.1	Class Result	4
2.2.2	Class ResultSet	4
2.2.3	Class QELResultFormat	4
2.2.4	Example	5
2.3	QEL-SQL Wrapper	6
2.3.1	Datalog-QEL to SQL example	6
3	Bringing SQI to Edutella	6
3.1	Offering existing Edutella providers as external providers	7
3.1.1	Installing the external provider	8
3.2	Connecting an external provider to Edutella	9
3.2.1	Installing and running an Edutella provider proxy	9
4	Edutella Mappings	11
4.1	Types of Mappings	11
4.1.1	Property Mappings	11
4.1.2	Property/Value Mappings	12
4.1.3	Double Mapping	13
4.1.4	Default Values	14
4.2	Specifying Mappings with the SQI Provider	15
4.3	Using Mapping Classes	16
4.3.1	Class PropertyMapping	16
4.3.2	Class DefaultValue	16
4.3.3	Class QueryMapping	16

1 Introduction

This document tries to bring to the light some of the most important issues of Edutella. It is intended to developers trying to connect to the Edutella network. On the one hand, a configuration out-of-the-box where only the configuration file needs to be changed is described. On the other hand, there is also an explanation about the code for those cases where ad-hoc solutions are needed.

2 Using Edutella classes to build a wrapper

The use of different ontologies, query languages and repositories among the different Learning Management Systems requires the implementation of several wrappers. Here it is described how to use the Edutella java classes to help on the implementation of wrappers receiving QEL queries.

The process that any wrapper must perform is the following:

1. Receives a QEL query as a string that uses the Elena Common Ontology
2. Understand the QEL query
3. Maps the Elena Common Ontology to the local ontology (e.g. LOM)
4. Converts the QEL to the local query language (e.g. SQL or XQuery)
5. Sends the transformed query to the repository
6. Receives the results from the repository
7. Transforms the results to a variable binding table
8. Return the results

While Edutella classes can not help the developer on the implementation of steps 3 to 6 it do can on steps 2 and 7. Edutella classes parse a QEL query into a Query object that can be use afterwards and permit to store the results directly into a ResultSet object that can generate the results table in RDF.

In this document it is described the main elements of this classes that could help developers. A UML class diagram is depicted in figure 1. If someone needs further information he should check directly the QEL specification¹, the source code or contact me.

2.1 Parsing a QEL Query

Two classes are here described that belong to packages `net.jxta.edutella.eqm` and `net.jxta.edutella.eqm.io`. Check the whole packages for more information.

¹<http://edutella.jxta.org/spec/qel.html>

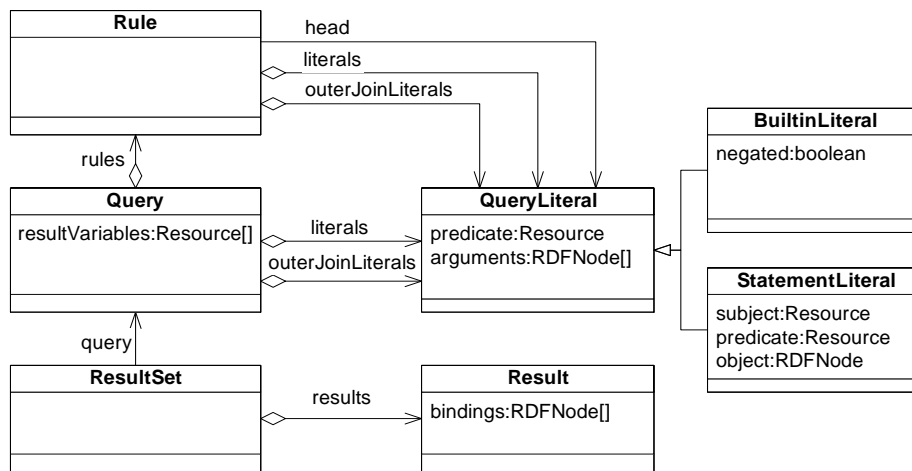


Figure 1: Edutella Query Model

2.1.1 Class Query

Represents an Edutella QEL query. Each Query object aggregates an arbitrary number of Query Literals (QueryLiteral objects), Outer Join Query Literals and Rules. Edutella Datalog rules. When executing the query all Query Literals are evaluated using the necessary rules.

Iterator getLiterals() returns iterator over all body literals of this query

Iterator getOuterJoinLiterals() returns iterator over all outer join body literals of this query

List getResultVariables() returns the list of result variables (variables for which bindings are requested)

Iterator getRules() returns an iterator over all rules of this query

Iterator getVariables() returns an iterator over all variables of this query

2.1.2 QELQueryParser

Represents a parser that converts a QEL query (in XML format) into a Query object.

Query parse(String src) converts a string containing a QEL query (in XML format) into a Query Object

2.1.3 DatalogQueryParser

Represents a parser that converts a QEL query (in datalog format) into a Query object.

Query parse(String src) converts a string containing a QEL query (in datalog format) into a Query Object

2.1.4 Example

Here there are two examples of how to read a QEL query from a String (in the variable `queryString`) and transform it into a Query object. The first one reads a XML encoded query and the second one reads a datalog encoded query.

- ```
QELQueryParser queryParser = new QELQueryParser();
Query query = queryParser.parse(queryString);
```
- ```
DatalogQueryParser dp = new DatalogQueryParser();
String st = "@prefix qel: <http://www.edutella.org/qel#>.\n" ;
st = st + "@prefix dc: <http://purl.org/dc/elements/1.1/>.\n" ;
st = st + "?- qel:s(Resource,dc:title,Title).\" ;
Query q = dp.parse(st) ;
```

2.2 Adding Results

Three classes are here described that belong to packages `net.jxta.edutella.eqm` and `net.jxta.edutella.eqm.io`. Check the whole packages for more information.

2.2.1 Class Result

This class holds a result tuple containing bindings for all result variables

boolean hasBinding(Resource var) checks if this tuple contains a binding for var

void addBinding(Resource variable, RDFNode value) adds a variable binding to this tuple

2.2.2 Class ResultSet

Aggregates an arbitrary number of Result objects to represent a complete result set for a Query. The individual results are tuples of bindings. The variable bindings order is the the result variables order in the result variables list

Result createResult() creates a new Result object

void addResult(Result result) adds a result to this set

boolean hasResults() checks if result set contains results

Iterator getResultResults() returns an iterator over all results

List getResultVariables() returns the list of result variables (variables for which bindings are requested)

2.2.3 Class QELResultFormat

Transforms a ResultSet object to a RDF string representing the bindings table.

String format(ResultSet src) transforms a ResultSet object to a RDF string representing the bindings table.

2.2.4 Example

For each new row in the binding table (eduQuery represents the Query object containing the QEL query)

```
ResultSet resultset = eduquery.createResultSet();
Result result = resultset.createResult();

// here we check if the QueryLiteral is a statement and if
// there is a variable at the Subject property of a triple
// it should be extended for the rest of possibilities
for (Iterator i = eduquery.getLiterals(); i.hasNext();) {
    QueryLiteral el = (QueryLiteral) i.next();
    if (el instanceof StatementLiteral) {
        StatementLiteral rstmt = (StatementLiteral) el;

        Resource subject = rstmt.getSubject();
        if (eduquery.isVariable(subject)) {
            String value = "example" ;
            RDFNode v = createLiteral(value);
            result.addBinding(subject, v);
        }
    }
}

resultset.addResult(result);

-----

// and at the end we could write the results
if (rs.hasResults()) {
    for (Iterator i = rs.getResults(); i.hasNext();) {
        Result tuple = (Result) i.next();
        Resource resVar = query.getVarForLocalName("Resource");
        Resource titleVar = query.getVarForLocalName("Title");

        RDFNode resource = tuple.getBinding(resVar);
        RDFNode title = tuple.getBinding(titleVar);

        String resourceStr = resource.toString();
        String titleStr =
            TextUtilities.replaceAll(
                title.toString(),
                "%20",
                " ");

        System.out.println(titleStr + " (" + resourceStr + ")");
    }
}
```

2.3 QEL-SQL Wrapper

An example of the use of a QEL to SQL conversion can be found at the package `net.jxta.edutella.provider.qlr`. It takes a Query object, transforms it to a SQL query String (where the triples are stored in one table), sends the query to a Relational Database and converts the results to a binding table.

Note: this wrapper only performs a query language mapping. There is not schema mapping.

2.3.1 Datalog-QEL to SQL example

In order to map from datalog-QEL to a SQL query we can use the `OLRQueryFormat` class. The new SQL query assumes that the database has a table (or view) of triples representing the RDF information (with fields subject, predicate and object). The following is an example of code where we transform a simple datalog-QEL query into SQL.

```
String queryString =
    "@prefix qel: <http://www.edutella.org/qel#>." ;
queryString= queryString +
    "@prefix dc: <http://purl.org/dc/elements/1.1/>." ;
queryString= queryString +
    "?- qel:s(X, <dc:title>, Z), qel:equals(Z, 'Word')." ;
DatalogQueryParser dq = new DatalogQueryParser() ;
OLRQueryFormat of = new OLRQueryFormat() ;
Query query = dq.parse(queryString) ;
String sqlQuery = of.format (query) ;
System.out.println("SQL query: " + sqlQuery) ;
```

and the output is

```
SQL query: SELECT STMT0.SUBJECT STMT0_SUBJECT,
            STMT0.PREDICATE STMT0_PREDICATE,
            STMT0.OBJECT STMT0_OBJECT
FROM EDUTELLA_STATEMENTS STMT0
WHERE STMT0.PREDICATE = 'dc:title' AND
STMT0.OBJECT = 'Word'
```

3 Bringing SQI to Edutella

This section offers a guide to users and developers of how to connect a system to the Edutella P2P network using the Simple Query Interface² (SQI hereafter). In this document are described the possible scenarios and the different possibilities that systems can use to join the network. References to the packages needed and examples of configuration files are also given.

²<http://rubens.cs.kuleuven.ac.be/vqwiki-2.5.5/jsp/Wiki?LorInteroperability>

3.1 Offering existing Edutella providers as external providers

In order to connect an external provider to Edutella, it must have implemented the SQI. In current Edutella versions, several wrappers have been developed to connect different kind of repositories like RDF-based files, RDBs, RDF repositories (e.g. Sesame or RSSDB), etc... Any provider can use those providers to connect to its repository and offer the content via SQI. For that goal, an extra layer has been developed on top of those wrappers.

The class in charge of that extra layer is *net.jxta.edutella.sqi.proxy.ProviderProxy*. This class implement all the methods of the SQI and uses one of the Edutella wrappers to send the queries to a repository. The wrapper to be used is configured in the Target.xml configuration file. The following is an example of such a file where it is specified to use the RDQLProviderConnection which uses a RDF file as repository. The name of the file is also passed to the configuration file.

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!DOCTYPE rdf:RDF [
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
    <!ENTITY pd 'http://www.learninglab.de/~brunkhor/rdf/PeerDescription#'>
]>

<rdf:RDF xmlns:rdf="&rdf;"
        xmlns:rdfs="&rdfs;"
        xmlns:pd="&pd;">

<pd:Peer rdf:about="#Peer">
    <pd:peerName>rdqlProvider@DOC</pd:peerName>
</pd:Peer>

<!-- =====
    Here the class corresponding to the Edutella wrapper and the
    attributes of that class must be defined.
    ===== -->
<pd:Component rdf:about="#ProviderConnection">
    <pd:javaClass>net.jxta.edutella.provider.rdql.RDQLProviderConnection</pd:javaClass>
    <pd:modelFile>books.rdf</pd:modelFile>
</pd:Component>
</rdf:RDF>
```

Another example of a similar configuration file using the relational database Edutella wrapper would be:

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!DOCTYPE rdf:RDF [
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
    <!ENTITY pd 'http://www.learninglab.de/~brunkhor/rdf/PeerDescription#'>
]>

<rdf:RDF xmlns:rdf="&rdf;"
        xmlns:rdfs="&rdfs;"
```

```

        xmlns:pd="&pd;">

<pd:Peer rdf:about="#Peer">
  <pd:peerName>RDBProvider@DOC</pd:peerName>
</pd:Peer>

<!-- =====
      Here the class corresponding to the Edutella wrapper and the
      attributes of that class must be defined.
      ===== -->
<pd:Component rdf:about="#ProviderConnection">
  <pd:javaClass>net.jxta.edutella.provider.olr.OLRProviderConnection</pd:javaClass>
  <pd:jdbcDriverClass>jdbcDriverClass</pd:jdbcDriverClass>
  <pd:dbUrl>http://www.XYZ.com</pd:dbUrl>
  <pd:dbUser>UserName</pd:dbUser>
  <pd:dbPassword>Password</pd:dbPassword>
  <pd:stmtViewName>TripleViewName</pd:stmtViewName>
</pd:Component>
</rdf:RDF>

```

3.1.1 Installing the external provider

This SQI implementation uses Axis and a web server in order to provide a web service based binding. In order to better understand this, it is recommended for any developer to read the "SQI Implementor's guidelines"³. In this document, we assume that a web server (e.g. tomcat) together with axis is already installed. The files we refer to in this document can be found in the Edutella cvs. The rest of steps are as follows:

1. Copy Edutella jar files to the "/lib" folder under the axis web application. The required files (although depending on the wrapper used some other might be required) are:
 - antlr.jar
 - concurrent-1.3.0.jar
 - edutella.jar
 - icu4j.jar
 - jakarta-oro-2.0.5.jar
 - javax.servlet.jar
 - jdom.jar
 - jena.jar
 - jxta.jar
 - jxtaptls.jar
 - jxtasecurity.jar
 - lf5.jar
 - log4j.jar
 - minimalBC.jar
 - rdf-api-2001-01-19.jar
 - rdfapi.jar
 - rdffilter.jar

³<http://nm.wu-wien.ac.at/e-learning/inter/sqi/ImplementingSQI.pdf>

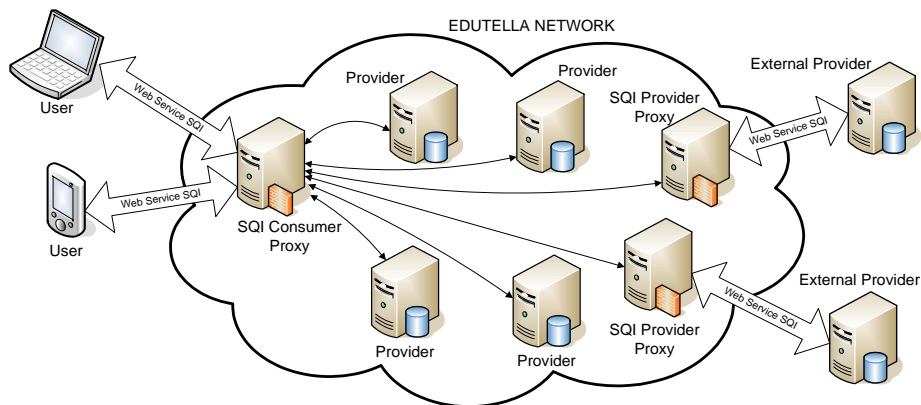


Figure 2: Edutella P2P network and SQI

- sqiservice.jar

2. Copy the Target.xml file to the base folder of your axis/tomcat classpath. The provider will try to find `"/Target.xml"` as a configuration file.
3. Edit and modify the Target.xml to reflect the values of your provider (see above).
4. Deploy the SessionManagement service and the Query service with the files rdqlQuery-Deploy.wsdd and rdqlSessionMgmtDeploy.wsdd (under the wsdd folder)
5. Start your web server or reload your axis web application.

After these steps, the content of the repository must be offered as a web service application at `http://host:8080/axis/services/RDQLSessionManagement` and `http://host:8080/axis/services/RDQLQuery`. The name of the services can be changed modifying the wsdd files.

3.2 Connecting an external provider to Edutella

In order to connect a system to Edutella using the SQI, several prerequisites must be fulfilled:

- The provider must, of course, have implemented the SQI (see section 3.1 for instructions about how to use existing Edutella wrappers)
- An Edutella provider proxy must be running in parallel to the provider. This Edutella provider proxy is in charge of:
 1. Receiving the queries submitted to the Edutella network
 2. Forward the queries to the external provider
 3. Receive the results from the external provider
 4. Forward the results to the Edutella network

This scenario is depicted in figure 2.

3.2.1 Installing and running an Edutella provider proxy

The Edutella provider proxy runs as a stand-alone application (not under a web server application like the external provider described in 3.1). Therefore, it does not require any extra-software to be installed (except java, of course). A configuration file is available in order to set up the proxy and connect it to Edutella. Although in the file is named `ProviderProxy.xml`, the name of

this configuration file can be passed as an argument to the application. This configuration file contains information about the services that this proxy will implement, properties to connect to the network, peer description, etc... We will ignore most of those settings and focusing on the beginning and the end of this file. These two parts would look like

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!DOCTYPE rdf:RDF [
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
    <!ENTITY pd 'http://www.learninglab.de/~brunkhor/rdf/PeerDescription#'>
]>

<rdf:RDF xmlns:rdf="&rdf;"
        xmlns:rdfs="&rdfs;"
        xmlns:pd="&pd;">

<pd:Peer rdf:about="#Peer">
    <pd:peerName>rdqlProviderProxy@DOC</pd:peerName>
    <pd:hasDescription rdf:resource="#Description"/>
    <pd:hasApplication rdf:resource="#Application"/>
    <pd:hasComponent rdf:resource="#Component"/>
    <pd:hasSystemService rdf:resource="#SystemService"/>
    <pd:hasSystemBuilder rdf:resource="#SystemBuilder"/>
    <pd:memberOf rdf:resource="#PeerGroup"/>
</pd:Peer>

.....

<!-- =====
    A Task is just another type of component, not for handling a
    single Event at a time, but for more persistent operation. This
    Task is using a specific RDF file as metadata source.
    ===== -->
<pd:Task rdf:about="#ProviderConnection">
    <pd:javaClass>net.jxta.edutella.sql.proxy.ProviderProxy</pd:javaClass>
    <pd:queryServiceURL>http://host:8080/axis/services/RDQLQuery</pd:queryServiceURL>
    <pd:sessionMgmtServiceURL>
        http://host:8080/axis/services/RDQLSessionManagement
    </pd:sessionMgmtServiceURL>
    <pd:user></pd:user>
    <pd:password></pd:password>
    <pd:numberResultsToRetrieve>50</pd:numberResultsToRetrieve>
</pd:Task>

</rdf:RDF>
```

The first part defines the information about this peer. The only thing that should be modified is the peerName (`<pd:peerName>rdqlProviderProxy@DOC</pd:peerName>`). The end of the file defines the information needed to connect to the external provider. The javaClass property must not be changed. The rest of the properties must be configured so queryServiceURL and sessionMgmtServiceUrl point to the external provider services. User and password are used to log into the external provider. If no user is specified, anonymous authentication is used. The last property indicates the number of results to retrieve. In the SQL, the requester gets a first set of results

and more results can be received using the `getAdditionalResults` call. This method is not implemented in Edutella (this concept is difficult to implement in a P2P network) so only the first set of results is used. The property `numberResultsToRetrieve` specifies the size of this set of results.

First of all, the configuration file must be modified accordingly (see above). Once it is done, in order to execute this proxy, several ant tasks are provided. In order to run this task ant must be installed and the command is "ant taskName". The tasks are the following:

- `usage`: gives an overview of the ant tasks contained in the makefile
- `chainsaw`: opens a graphical tool (local log4j gui) to follow the debugging of the applications
- `providerproxy`: runs an Edutella provider proxy
- `consumer`: starts edutella consumer based on `ConsumerPeer.xml`

In order to test the whole scenario, the external provider must be running. Then, the provider-proxy can be started and to check that it is really working the consumer can be started as well. In the list of providers, the name of the proxy should appear. Click on it so we indicate that we will send it a query. Click on load file and choose the file `exampleQuery.xml`. Then press on Search. At that moment, the query will be sent to the provider proxy. It will forward it to the external provider (according to its configuration file). Then, it will receive the results set from the external provider and send it back to the network. The answer should shown on the "Results" text area.

4 Edutella Mappings

Edutella is schema insensitive. It means that whenever a query is sent to the network, Edutella forwards it to all the providers in the network. If any of those providers does not understand the schema describing the query (e.g. the provider understands queries for author but not for creator) or does not have metadata for it (e.g. all its resources are for free so there is not price attribute), in that case the provider will return no results. This happens even if the concept behind "creator" and "author" is the same or if all the prices of its resources are for free so a default value "0" could have been returned as "price". In order to solve these problems we introduce *mappings*.

4.1 Types of Mappings

There might exist different kind of mappings regarding to a specific query and the local schema. In the following we describe them:

4.1.1 Property Mappings

A property mapping is useful to transform a query asking for one attribute into another one we know is equivalent in our local schema. The syntax for this mapping would be:

```
(X, property1, Y) -> (X, property2, Y)
```

For example, given the following query

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
?- qel:s(X,dc:contributor,Y).
```

and if our provider does not use that property but it uses collaborator and both have the same meaning, then it is possible to create the following mapping (it is broken in several lines for presentation purposes but it is not required)

```
(X, http://purl.org/dc/elements/1.1/contributor, Y)
->
  (X, http://provider.local.schema.org/collaborator, Y)
```

Therefore, the query would be rewritten at the provider into

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix ls: <http://provider.local.schema.org/>.
?- qel:s(X,ls:collaborator,Y).
```

In addition, this process is invisible for the requester who receives the results for the variables X and Y accordingly.

4.1.2 Property/Value Mappings

In some cases the property asked in the query is understood by the provider but the taxonomy used at the provider is different. In this case it is needed to map not only the property but also the value. The syntax of this mapping is

```
(X, property1, value1) -> (X, property2, value2)
```

A simple example of this situation would be a query asking for resources in German like the following:

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
?- qel:s(X,dc:language,'de').
```

but the provider does not use the alpha-2-codes (ISO-639-1) but the alpha-3-codes (ISO-639-2). Therefore, the previous query would not match any of the local resources. That's why the provider needs to introduce a mapping like:

```
(X, http://purl.org/dc/elements/1.1/language, 'de')
->
  (X, http://purl.org/dc/elements/1.1/language, 'ger')
```

and the query would be rewritten into

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
?- qel:s(X,dc:language,'ger').
```

NOTE:

if the query was

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
?- qel:s(X,dc:language,Y).
```

the mapping does not apply as it is not possible to know which taxonomy the requester is using.

4.1.3 Double Mapping

In some situations, it might be possible that two properties together map into only one at the provider side or that a single query property requires a mapping into two properties at the local provider. This kinds of mapping might have the following syntaxes:

- 1 to 2 mapping:

```
(X, property1, Y),(Y, property2, Z) -> (X, property3, Z)
```

or

```
(X, property1, Y),(Y, property2, value2) -> (X, property3, value3)
```

- 2 to 1 mapping:

```
(X, property1, Z) -> (X, property2, Y),(Y, property3, Z)
```

or

```
(X, property1, value1) -> (X, property2, Y),(Y, property3, value2)
```

- 2 to 2 mapping:

```
(X, property1, Y),(Y, property2, Z)
```

->

```
(X, property3, W),(W, property4, Z)
```

or

```
(X, property1, Y),(Y, property2, value1)
```

->

```
(X, property3, W), (W, property4, value2)
```

An example of a mapping 2 to 1 is the following query:

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>.
?- qel:s(X,dc:creator,Y),
   qel:s(Y,vcard:fn,Z).
```

If the provider has a creator field but not formatted according to vcard full name it can use the following mapping:

```
(X, http://purl.org/dc/elements/1.1/creator, Y),
(Y, http://www.w3.org/2001/vcard-rdf/3.0#fn, Z)
-> (X, http://purl.org/dc/elements/1.1/creator, Z)
```

and the query is rewritten into

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
?- qel:s(X,dc:creator,Z).
```

in which case the variables *X* and *Z* would receive the right values and *Y*, as it does not exist locally, would receive a null value.

NOTE:

although sometimes mappings with more than two statements on each side would be interesting, their complexity increases too much and that's why they haven't been implemented.

4.1.4 Default Values

This is not a proper mapping but as its name indicates it represents the assignment of default values to some properties. In some situations a provider might receive one query with some attributes it is able to understand but it does not have proper metadata for it. In those case, default values can be used. The syntas for default values is as follows:

```
property -> value
```

With default values, if a provider which contains free resources and therefore had no need to include metadata to describe wheather a specific resource was for free or not, could use the following mapping

```
http://ltsc.ieee.org/2002/09/lom-rights#cost -> 'No'
```

in order to automatically add the value “No” (indicating “free”) to any query of the type:

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix lom-rights: <http://ltsc.ieee.org/2002/09/lom-rights#cost>.
?- qel:s(X,dc:title,Y),
   qel:s(X,lom-rights:cost,Z).
```

In this example, only the query

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
?- qel:s(X,dc:title,Y).
```

is sent to the repository and once the repository sends the results, a new value is added for the variable Z with the specified value “No”

NOTE:

In case that there is no variable where the default value should go like in e.g.

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix lom-rights: <http://ltsc.ieee.org/2002/09/lom-rights#cost>.
?- qel:s(X,dc:title,Y),
   qel:s(X,lom-rights:cost,'No').
```

if the value is the same as the specified value, then the query would succeed and if there is not, then it wouldn't return any value

NOTE2:

In case that the variable where the default value will be assigned appears as well in one building literal like e.g.

```
@prefix qel: <http://www.edutella.org/qel#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix lom-rights: <http://ltsc.ieee.org/2002/09/lom-rights#cost>.
?- qel:s(X,dc:title,Y),
   qel:s(X,lom-rights:cost,Y),
   qel:equals(Y,'No').
```

then the variable Y in the building literal is replaced by the default value before it is sent to the repository assuring the right evaluation of the builtin literal.

4.2 Specifying Mappings with the SQI Provider

The mappings are simple to specify in the current SQIProvider by means of the configuration file. Here is an example of one configuration file (resources/config/Target.rdf in the Edutella CVS):

```
<!-- =====
      Here the class corresponding to the Edutella query mapping wrapper and the
      attributes of that class must be defined.
      ===== -->
<pd:Component rdf:about="#QueryMapping">
  <pd:javaClass>net.jxta.edutella.eqm.mapping.QueryMapping</pd:javaClass>
  <pd:normalMappings rdf:resource="#Mappings"/>
  <pd:defaultValueMappings rdf:resource="#DefaultValues"/>
</pd:Component>
```

This part simply define the class used for the mapping and specify the pointer to the property mappings and default values. At this time, QueryMapping is the only class that performs mappings so this part must remain unchanged.

```
<rdf:Seq rdf:about="#Mappings">
  <rdf:li>(X, http://purl.org/dc/elements/1.1/identifier,Y)
    -> (X, http://www.l3s.de/~olmedilla/uri, Y)</rdf:li>
  <rdf:li>(X, http://purl.org/dc/elements/1.1/contributor, Y),
    (Y, http://www.w3.org/2001/vcard-rdf/3.0#fn, Z)
    -> (X, http://purl.org/dc/elements/1.1/creator, Z)</rdf:li>
  <rdf:li>(X, http://www.open-q.de/supplier_name, Y),
    (Y, http://www.w3.org/2001/vcard-rdf/3.0#orgname, Z)
    -> (X, http://www.open-q.de/supplier_name, Z)</rdf:li>
  <rdf:li>(X, http://www.open-q.de/organizer_contact, Y),
    (Y, http://www.w3.org/2001/vcard-rdf/3.0#fn, Z)
    -> (X, http://www.open-q.de/organizer_contact, Z)</rdf:li>
  <rdf:li>(X, http://www.hcd-online.com/nsv1/add_information,Y)
    -> (X, http://www.l3s.de/~olmedilla/uri, Y)</rdf:li>
</rdf:Seq>
```

This part of the configuration file shows the property mappings. It is a RDF sequence and each element of the sequence (each value between the tags <rdf:li> and </rdf:li>) specifies a new property mapping. The syntax corresponds to the one described in sections 4.1.1,4.1.2 and 4.1.3.

```
<rdf:Seq rdf:about="#DefaultValues">
  <rdf:li>http://purl.org/dc/elements/1.1/language -> 'de'</rdf:li>
  <rdf:li>http://purl.org/dc/elements/1.1/creator -> 'Jan Brase'</rdf:li>
  <rdf:li>http://ltsc.ieee.org/2002/09/lom-rights#cost -> 'No'</rdf:li>
  <rdf:li>http://ltsc.ieee.org/2002/09/lom-rights#copyright_and_other_restrictions
    -> 'No'</rdf:li>
  <rdf:li>http://ltsc.ieee.org/2002/09/lom-edu#learning_resource_type
    -> 'Unknown'</rdf:li>
  <rdf:li>http://www.open-q.de/supplier_name -> 'null'</rdf:li>
  <rdf:li>http://www.open-q.de/organizer_contact -> 'null'</rdf:li>
  <rdf:li>http://www.open-q.de/goal -> 'null'</rdf:li>
```

```

<rdf:li>http://www.open-q.de/priceamount -> '0'</rdf:li>
<rdf:li>http://www.open-q.de/pricecurrency -> 'EUR'</rdf:li>
<rdf:li>http://www.hcd-online.com/nsv1/learning_resource_category
-> 'LM'</rdf:li>
</rdf:Seq>

```

This last part of the configuration file is also a RDF sequence and specifies the default values. The syntax corresponds to the one described in section 4.1.4

4.3 Using Mapping Classes

This section tries to describe the use of the mapping classes and its aim is to help developers to implement their own solutions with them. In order to fully understand this section it is recommended to read first the information described above about class Query (section 2.1.1) and class ResultSet (section 2.2.2).

4.3.1 Class PropertyMapping

The class PropertyMapping represents, as its name indicates, a property mapping. It receives a string in the constructor specifying the mapping according to the syntax described in sections 4.1.1, 4.1.2 and 4.1.3. Here is one example:

```

PropertyMapping mp =
  new PropertyMapping (
    "(X, http://purl.org/dc/elements/1.1/title,Y)
    -> (X, http://purl.org/dc/elements/1.1/description, Y)"
  ) ;

```

4.3.2 Class DefaultValue

The class DefaultValue represents default values assigned to properties. It receives a string in the constructor which specifies the default value according to the syntax described in section 4.1.4. Here is one example:

```

DefaultValue dv =
  new DefaultValue ("http://purl.org/dc/elements/1.1/language -> 'de'" ) ;

```

4.3.3 Class QueryMapping

The class QueryMapping is the class in charge of transforming a query into a new query with the properties mapped and then transform the results taking into account the mappings and default values.

This class is created without any argument in the constructor

```

QueryMapping qm = new QueryMapping() ;

```

and once it is created, new property mappings or default values can be added. For example, given the mapping and default value defined above we have

```
qm.addMapping(mp) ;
qm.addMapping(dv) ;
```

Once the class QueryMapping is completely initialized with all the property mappings and default values, then it is ready to do its work.

The mapping consists of two steps. The former one transforms the query into a new one with the attributes mapped according to the property mappings and the latter transforms the results mainly adding the default values.

The former is performed by means of the method qm.mapQuery() which receives one Query as an argument and returns a mapped Query. The latter is performed with the method queryMapping.mapResultSet() which receives one ResultSet as an argument and returns a transformed ResultSet.

An example of the whole code would be:

```
// as you know the query must be parsed from e.g. datalog to the Query class
DatalogQueryParser dp = new DatalogQueryParser() ;
DatalogQueryFormat df = new DatalogQueryFormat() ;
df.setNsPrefix("dc", "http://purl.org/dc/elements/1.1/") ;
df.setNsPrefix("vcard", "http://www.w3.org/2001/vcard-rdf/3.0#") ;

String queryString = "@prefix qel: <http://www.edutella.org/qel#>." ;
queryString += "@prefix dc: <http://purl.org/dc/elements/1.1/>." ;
queryString += "@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>." ;
queryString += "?- qel:s(X,dc:title,Title)," ;
queryString += "qel:like(Title,'prueba')," ;
queryString += "qel:s(X,dc:author,Author)," ;
queryString += "qel:s(X,dc:subject,'Typel')," ;
queryString += "qel:s(X,dc:contributor,'DOC')," ;
queryString += "qel:s(X,dc:creator,Creator)," ;
queryString += "qel:like(Creator,'creator')," ;
queryString += "qel:s(X,dc:language,Language)," ;
queryString += "qel:s(X,dc:date,'dateValue')," ;
queryString += "qel:s(Creator,vcard:fn,Name)." ;

// here is where the parsing takes place
Query query = dp.parse(queryString) ;

System.out.println ("Query: " + df.format(query)) ;

// a class QueryMapping is created
QueryMapping qm = new QueryMapping() ;

// different mappings are created
PropertyMapping mp1 =
    new PropertyMapping (
        "(X, http://purl.org/dc/elements/1.1/title,Y
        -> (X, http://purl.org/dc/elements/1.1/description, Y)" ) ;
PropertyMapping mp2 =
    new PropertyMapping (
        "(X, http://purl.org/dc/elements/1.1/author, Y
        -> (X, http://purl.org/dc/elements/1.1/author2, 'Daniel'))" ) ;
PropertyMapping mp3 =
```

```

    new PropertyMapping (
        "(X, http://purl.org/dc/elements/1.1/contributor, Y)
        -> (X, http://purl.org/dc/elements/1.1/contributor2, Y)" ) ;
PropertyMapping mp4 =
    new PropertyMapping (
        "(X, http://purl.org/dc/elements/1.1/creator, Y),
        (Y, http://www.w3.org/2001/vcard-rdf/3.0#fn, Z)
        -> (X, http://purl.org/dc/elements/1.1/creator2, Z)" ) ;

// default values are also created
DefaultValue d1 =
    new DefaultValue ("http://purl.org/dc/elements/1.1/language -> de" ) ;
DefaultValue d2 =
    new DefaultValue ("http://purl.org/dc/elements/1.1/creator2 -> null" ) ;

// now they are adding to the QueryMapping class
qm.addMapping(mp1) ;
qm.addMapping(mp2) ;
qm.addMapping(mp3) ;
qm.addMapping(mp4) ;
qm.addMapping(d1) ;
qm.addMapping(d2) ;

// here, given a specific query, the mapping is used
Query mappedQuery = qm.mapQuery(query) ;
System.out.println ("Query mapped into: " + df.format(mappedQuery)) ;

// this statement executes the mapped query
// (you should replace with the method you use for that)
ResultSet rs = provider.executeQuery(mappedQuery) ;

QELResultFormat qf = new QELResultFormat() ;
String results = qf.format(rs) ;

System.out.println ("resultSet received: " + results) ;

// the resultSet must also be mapped (inverse mapping)
// and for it we use the same query mapping
// NOTE: the query mapping will always do the inverse mapping of
// the last query sent to the provider
ResultSet mappedResultSet ;
mappedResultSet = queryMapping.mapResultSet(rs) ;

results = qf.format(rs) ;
System.out.println ("final resultSet: " + results) ;

    In the example above, the original query is

@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>.
?- qel:s(<#Creator>,vcard:fn,<#Name>),
   qel:s(<#X>,dc:author,<#Author>),
   qel:s(<#X>,dc:contributor,'DOC'),
   qel:s(<#X>,dc:creator,<#Creator>),
   qel:s(<#X>,dc:date,'dateValue'),

```

```
qel:s(<#X>,dc:language,<#Language>),
qel:s(<#X>,dc:subject,'Type1'),
qel:s(<#X>,dc:title,<#Title>),
qel:like(<#Creator>,'creator'),
qel:like(<#Title>,'prueba').
```

and according to the mappings the mapped one is

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.
?- qel:s(<#X>,dc:author2,'Daniel'),
   qel:s(<#X>,dc:contributor,'DOC'),
   qel:s(<#X>,dc:date,'dateValue'),
   qel:s(<#X>,dc:description,<#Title>),
   qel:s(<#X>,dc:subject,'Type1'),
   qel:like(<#Title>,'prueba').
```

assuming that we get one result as follows

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:edu="http://www.edutella.org/qel#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <edu:ResultSet>
    <edu:result>
      <rdf:Seq>
        <rdf:li rdf:resource="prueba"/>
        <rdf:li rdf:resource="http://www.xyz.com/a1.html"/>
        <rdf:type rdf:parseType="Resource">
          </rdf:type>
        </rdf:Seq>
      </edu:result>
    <rdf:type rdf:parseType="Resource">
    </rdf:type>
    <edu:resultVariables>
      <rdf:Seq>
        <rdf:li rdf:resource="#Title"/>
        <rdf:li rdf:resource="#X"/>
      </rdf:Seq>
    </edu:resultVariables>
  </edu:ResultSet>
</rdf:RDF>
```

the mapped version of that result would be:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:edu="http://www.edutella.org/qel#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description>
    <rdf:type rdf:parseType="Resource">
    </rdf:type>
    <edu:resultVariables>
      <rdf:Seq>
```

```
    <rdf:li rdf:resource="#Title"/>
    <rdf:li rdf:resource="#X"/>
    <rdf:li rdf:resource="#Name"/>
    <rdf:li rdf:resource="#Language"/>
  </rdf:Seq>
</edu:resultVariables>
<edu:result rdf:parseType="Resource">
  <rdf:_2 rdf:resource="http://www.xyz.com/a1.html"/>
  <rdf:_1 rdf:resource="prueba"/>
  <rdf:type rdf:parseType="Resource">
    </rdf:type>
    <rdf:_4>de</rdf:_4>
    <rdf:_3>null</rdf:_3>
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
  </edu:result>
  <rdf:type rdf:resource="http://www.edutella.org/qel#ResultSet"/>
</rdf:Description>
</rdf:RDF>
```