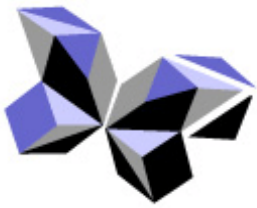

Digital Business Ecosystem

Contract n° 507953

Workpackage 24: DBE Implementation

Deliverable D24.6: Query Formulator and Semantic Discovery Tool Final Release

Project funded by the European Community under the “Information Society Technology” Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D24.6

Due dates: 06/2006

Delivery Date: 06/2006

Short Description:

This document accompanies the software deliverable for the final release of the Query Formulator and Semantic Discovery Tool. The tool is responsible for constructing queries for the Digital Business Ecosystem (DBE) distributed Knowledge Base and Service Registry. Three different distributions are available; a version contributing to the DBE Studio, where developers and users can search and browse information hosted in the DBE KB. The second version resides inside DBE Portal where users can search the semantics of services (i.e. the Service Manifests) of the distributed service registry by an Open Laszlo user interface. The last distribution is an API for developers of SME services for formulating queries into QML, the knowledge access language of the Knowledge Base. All three versions can query both data and metadata and are unattached to specific metamodels.

Author: Technical University of Crete (TUC)

Partners contributed: Technical University of Crete (TUC)

Made available to: Public

Versioning

Version	Date	Author, Organisation
1.0	30/05/2006	George Kotopoulos, Christos Alatzidis, Kostas Krommydas, Editing by Fotis G. Kazasis, Technical University of Crete
2.0	23/06/2006	Revisions based on the reviewer's comments/suggestions

Quality check:

1st Internal Reviewer : Andrew Edmonds, Intel Ireland

2nd Internal Reviewer: Victor Bayon, UCE

Table of Contents

List of Figures

1. Introduction

This document accompanies the software deliverable for the final release of the Query Formulator and Semantic Discovery Tool, a client-side tool for formulating queries against the Digital Business Ecosystem (DBE) Information and for browsing their results. DBE information is hosted in the Knowledge Base (KB) [1, 3]. The DBE KB provides a common and consistent description of the DBE world and its dynamics, as well as the external factors of the biosphere affecting it. Its content includes:

- Representations of domain specific ontologies (common conceptualization in a particular domain);
- Semantic descriptions of the Small to Medium Enterprises themselves in terms of business models, business rules, policies, strategies, views etc.;
- Semantic description of the Small to Medium Enterprise (SME) value offerings (description on how the services may be called) and the achieved solutions (service chains/compositions) to particular SME needs.
- Models for gathering usage and accounting/metering data and statistics.
- User profiles where SME's declare their preferences on the characteristics of demanded services and partners.

The DBE Knowledge Base (KB) follows the Object Management Group's (OMG) Model Driven Architecture (MDA) [8] approach for specifying and implementing knowledge structuring and organization. The MDA "*…defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform.*" [8]. In essence, this is done by separating the system design into Platform Independent Models (PIM) and Platform Specific models (PSM).

In addition the Knowledge Base follows the OMG's Meta Object Facility (MOF) [13] approach for metadata and data modelling and organization. This is done through four layers, namely M0, M1, M2 and M3 (see also *Figure 1*).

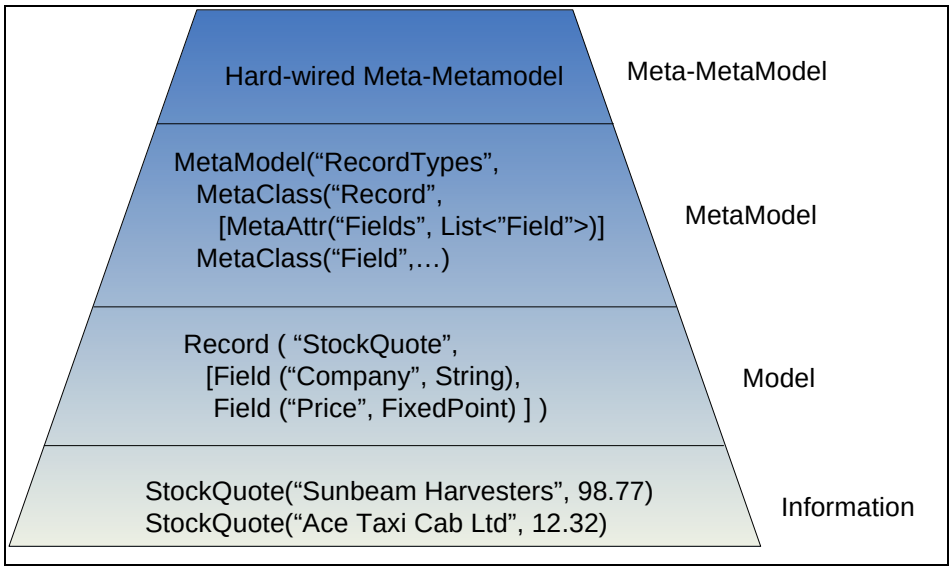

Figure 1: The four-layer metadata architecture

The level M0 of the architecture consists of the data that we wish to describe, the level M1 comprises the metadata that describe the data which are informally aggregated into models. The M2 level consists of the descriptions that define the structure and semantics of the metadata which are informally aggregated into metamodels and the M3 level consists of the description of the structure and semantics of the meta-metadata. The DBE Knowledge Base supports all the levels of the MOF architecture. Thus, each segment of information that is stored in the KB is placed as an instance of a modelling element of a higher layer within the MOF meta-data architecture. That is, MOF based languages or mechanisms should be used in the upper levels of the architecture for defining each segment of information. Different kinds of metamodels have been already developed and represented in the KB:

- Ontology Definition Model (ODM) [1], which enables the representation and storage of existing OWL domain ontologies into the KB
- Business Modelling Language (BML) [17] model and Semantic Services Language (SSL) [1] model, all which enable the representation and storage for the business models and the semantics of the services offered by SMEs into DBE.
- Service Definition Language (SDL) [18] Model for the technical description of atomic services
- User Preferences Model(UPM) [16] for expressing user profiles

Thus, the exploitable knowledge spectrum in DBE ranges from ontologies, to business models, to semantic and technical service descriptions, to user profiles, to usage data, etc. Each one of these

knowledge segments is represented using a different metamodel. In order to support efficient knowledge access over all these metamodels a query mechanism is provided through the Query Metamodel Language (QML) of the Recommender [2, 4]. QML is quite generic in order to specify, in a uniform way, knowledge access requests over all types of knowledge (both data and meta-data) that are kept in the DBE KB.

To this end, the "Query Formulator and Semantic Discovery Tool" offers a metamodel agnostic way for formulating queries on DBE information (both on data and metadata). This uniform way of constructing queries simplifies the process offering full QML compliance. There are three distributions available:

- The first is an Eclipse [10] plugin which accompanies the DBE Studio [14] offering developers and users an easy to use access point of DBE information.
- The second was implemented with the OpenLaszlo [11] platform and resides inside the DBE Portal [15].
- Finally, there is an API distribution meant to be used by SME's services that need to use QML-based query capabilities for accessing DBE information.

Chapter 2 provides a short description of the architecture of the query formulation and the semantic discovery environment. Chapter 3 supplies a user guide for the eclipse-based distribution and chapter 4 a user guide for the web-based one. The appendices provide the s/w APIs for formulating queries as expressions that follow the QML Metamodel.

2. Architecture

As already mentioned, two Graphical User Interface (GUI) types have been implemented. The first, for the DBE Studio, is implemented as an Eclipse plugin. The second, for the DBE Portal, is implemented as an Open Laszlo application. In order to support both of them the MVC (Model-View-Controller) [12] design pattern has been followed, dividing the system into three discrete parts, one managing the information (Model), one managing the graphical user interaction (View) and one for controlling the communication between the previous two (Controller), which can be extended so that to cover the needs of each specific implementation. Thus, there is a PIM part and a part referring on each individual implementation, PSM, as the MDA approach [8] suggests. As illustrated in *Figure 2*, the model part with the core controller is independent of any specific implementation, but there are two platform specific (Eclipse or OpenLaszlo) controllers and two views.

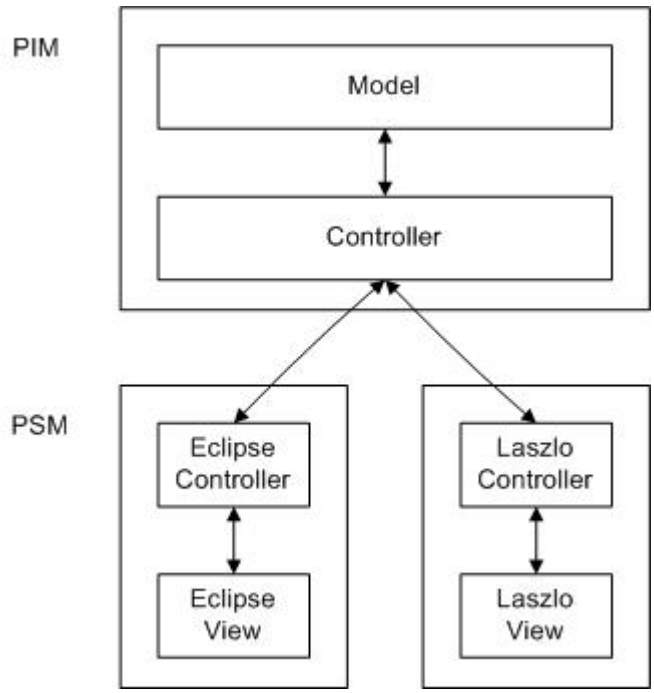

Figure 2: MVC design pattern implementation for the Query Formulator and Semantic Discovery Tool.

A more detailed architecture is presented in *Figure 3*. Users interact with the view to create queries or browse models and service manifests. The view calls the appropriate controller for the user demands to forward them to the model. The model is responsible to keep and manage MOF information (models, ontologies, data, etc), queries, query templates and query results. The MOF manager module is responsible to access and query the Knowledge Base of DBE. The queries and query templates are used by the query formulator module to formulate valid QML queries. The

MOF manager forwards the demands to the Knowledge Base through the proxy wrapper who is responsible for the actual communication with the services.

It is important to discriminate the structure of a query into two units. In the first one, the actual query constraints are kept (query), whereas the second one keeps the connection of these constraints to the actual MOF elements that are posed against (query template).

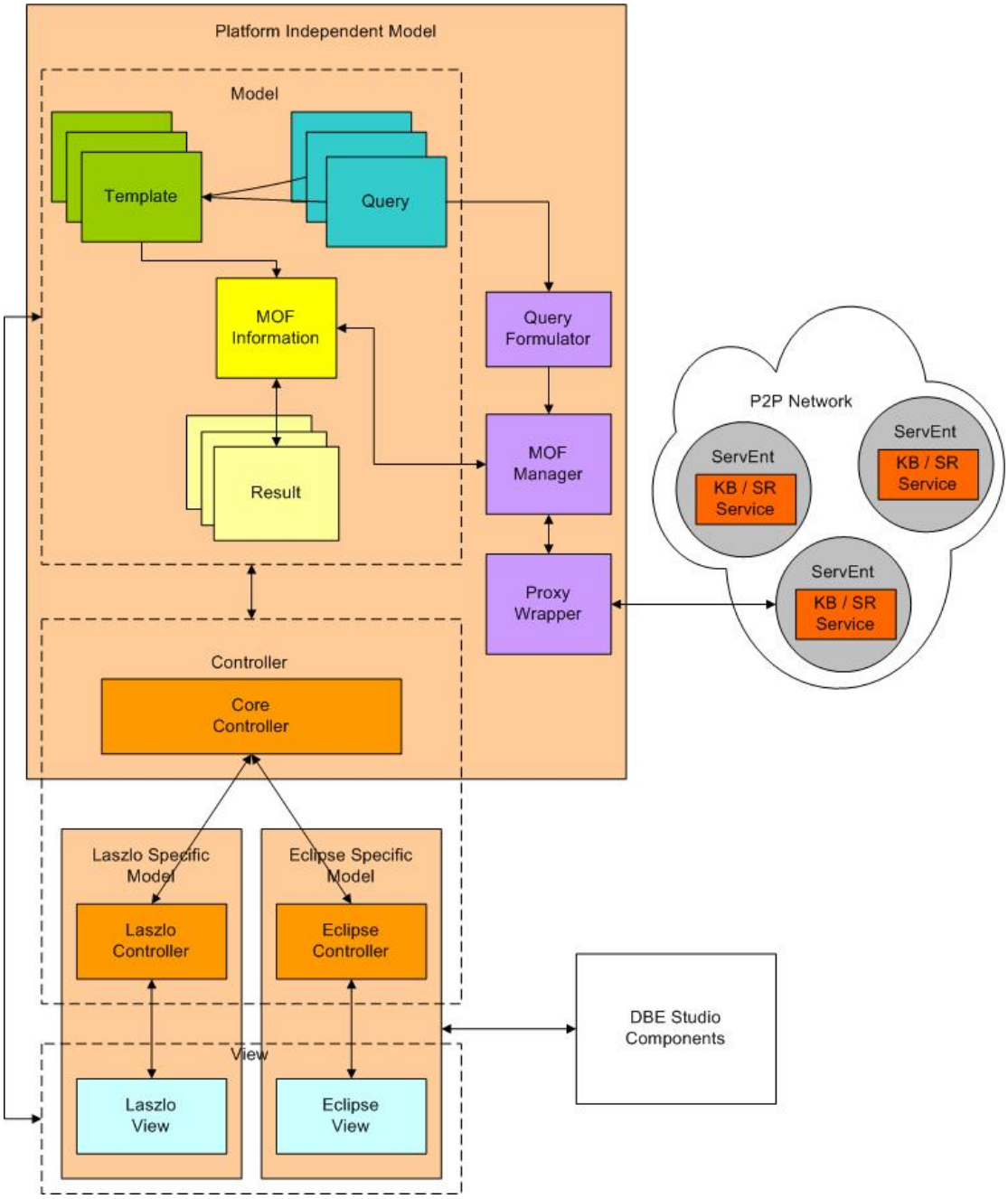

Figure 3: General System Architecture; the connection of the modules and the MVC design pattern followed are depicted.

2.1. Template

A template contains the MOF elements on which constraints are to be set. The elements come from either a metamodel (BML, SSL, etc), an ontology or a specific model. By using the template the complexity of MOF structures is concealed from the actual query. It is meant to be created once and used by many queries both from the front-end tools (DBE Studio and DBE Portal) as well as from inside SME's services. For example, for an SME service that wants to query the SR for hotels located somewhere and holding some star category, only a hard-coded template must be created by the developer and reused many times on each different search session. By using templates the user does not need to form queries into QML but in a Query-by-Example way, making the query creation process much easier.

2.2. Query

A query contains constraints referring to template elements of a certain template. The query only keeps the set of constraints (a pair of value, operation, weight) grouped into conjunctions (AND) and disjunctions (OR).

2.3. MOF Information

The MOF information module contains data structures for temporarily storing and handling all the possible kinds of MOF information (metamodels, models, ontologies and data) which can be retrieved from the KB or SR.

2.4. Result

A result is a data structure for keeping the results of queries as retrieved from the KB. Moreover, a result maintains references to the actual data (if retrieved) of the MOF information module.

2.5. Query Formulator

The query formulator is the actual mechanism for formulating queries in the QML by using the template and query structures. It is the module that can be used independently of front-end tools and enables SME services to use an easy to use query mechanism. Note that SME services may choose to directly use the QML API instead of the query formulator. This is a rather complex task as developers must be fully aware of the QML in order to effectively use it. On the other hand, the query formulator provides a transparent way for formulating the queries by only using the templates and queries structures and concealing even the existence of QML. Nevertheless, the query formulator does not export the full functionality of QML for simplicity reasons. An abstract description of the query formulator API can be found at the appendix A.

2.6. Proxy Wrapper

Communication of the tool with the DBE core services, (KB, SR) deployed on a servent, is done by a proxy interface. Proxy Wrapper assures that the connection with the service is kept alive and it generally tries to hide all proxy instabilities from the rest system.

2.7. MOF Manager

The MOF manager module, as its name implies, manages the temporarily stored metamodels, models, ontologies, and data of MOF information. Moreover, it is responsible for forwarding requests and queries to the KB through the proxy wrapper and the distribution of information retrieved from the KB to the appropriate modules.

2.8. Core Controller

The core controller module is responsible for keeping up to date the view and the model of the events that occur according to the MVC design pattern, e.g. when new results arrive or when a new element is added to the template. The core controller actually provides a set of functionality of the platform depended controllers and therefore it cannot stand by its own.

2.9. OpenLaszlo Controller

The OpenLaszlo controller is an extension of the core controller for the implementation on the OpenLaszlo platform. It is responsible to update the Laszlo view for model changes as well as to forward requests from the view to the model.

2.10. OpenLaszlo View

The OpenLaszlo view is the graphical user interface implemented on the OpenLaszlo platform. It is actually a set of XML documents which when parsed create a Macromedia Flash©® [20] application. A more detailed description for navigating this application is provided in chapter 4.

2.11. Eclipse Controller

The Eclipse controller is an extension of the core controller for the implementation on the Eclipse platform. It is responsible to update the eclipse view for model changes and to forward requests from the view to the model.

2.12. Eclipse View

Eclipse view is the GUI implemented on the Eclipse platform. It is comprised of an eclipse perspective. This comprises of a query editor and a set of related, supporting views, as described at section 3.1.

3. DBE Semantic Discovery Tool User Guide

The semantic discovery tool is provided as an Eclipse plugin. It is part of the DBE studio and offers a set of querying mechanisms to the DBE developers using the DBE Studio.

3.1. The Semantic Discovery Perspective

A perspective in the Eclipse platform provides a container for an eclipse editor and a set of related views whose positions are predefined. For the Query Formulator and Semantic Discovery Tool such a perspective has been created, the DBE Semantic Discovery Perspective, this consists of the Query Editor and the following views:

- **Query Navigator**: where the locally stored queries and templates can be seen.
- **Properties View:** where the properties of each query criterion can be seen and changed.
- **Template Properties View:** where template's properties of a certain query can be seen.
- **Results View:** displays the query results. Users can browse resulted service manifests or models resulting from a query and perform a set of operations on each of them.
- **Keyword Search View:** a search view for searching and using a set of keywords or path-based expressions with keywords.

Each view can be closed and correspondingly re-opened from the menu Window -> Show View -> and the desired view.

3.2. Starting with Semantic Discovery Tool

Both queries and templates are stored in the current eclipse workspace in the QFSDT project. *Figure 4* illustrates the semantic discovery tool perspective when it starts. Along with this plugin a set of simple example startup queries and templates are distributed and copied into the current workspace.

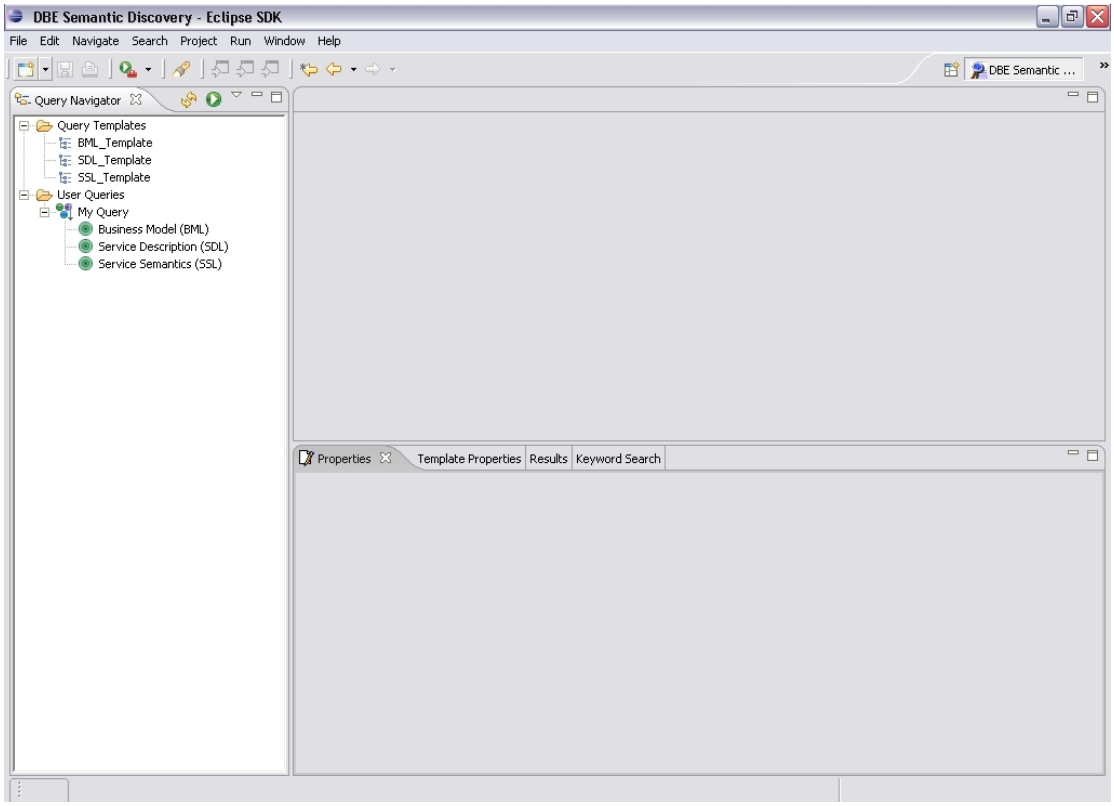

Figure 4: The DBE Semantic Discovery Tool Perspective.

3.3. Query Templates

Query templates are used as helpers to create and edit queries. Each query is connected to a template. Thus, in order to create a query we first have to create a template. Templates keep information about the types of the query terms to be used as queries. As queries can search for both data and models, the same holds for templates; a template can be created to search either models or data. To create a new template three ways can be followed:

- From main menu: File -> New -> Query Template
- "New template" at the pull-down menu of Query Navigator View
- Right click on a template and: "New template"

Following one of these steps, a wizard opens and guides the user through two steps into creating a new query template. Information, warning and error messages also guide the user in completing each step.

3.3.1. Query Template Creation

Step 1: Defining Name, Description and Template Type

For the first step, a name, a short description and a type for the template is given. This step can be seen at *Figure 5*. Moreover, it is possible to copy values from an existing template. There are three forms of templates:

- Metamodel/Language: Users can select one of the pre-defined DBE metamodels (i.e. BML, SSL, and SDL). The query will return models that are instances of this metamodel.
- Model: Users can select from one of the existing models. Models are retrieved from the SR that we are connected to. Moreover, the user has the ability to search models using keywords. The query will search data stored in Service Manifests and return a list of Service Manifests.
- Ontology: Users can select an ontology as an alternative of model. The ontologies shown are those stored in the KB.

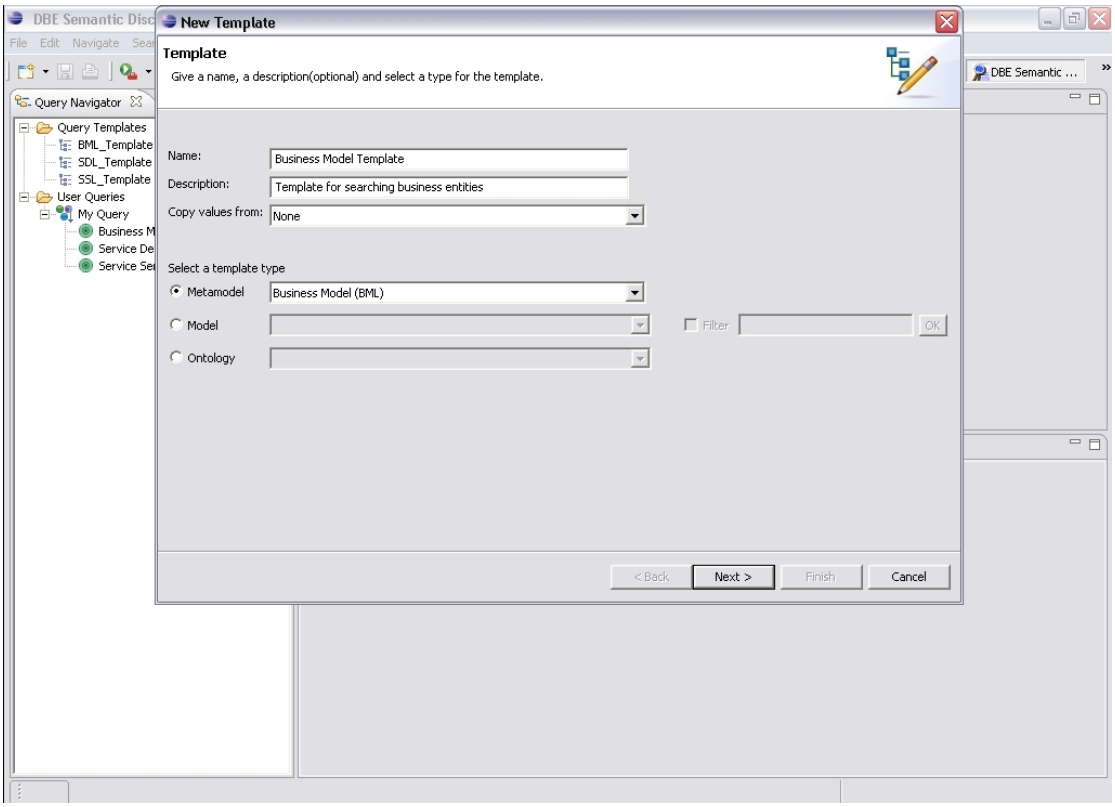

Figure 5: The first step of the new template wizard.

Step 2: Selection of template elements

In step two, a tree view is located at the left side and a table at the right side, as illustrated at *Figure 6*. The tree represents the metamodel, model or ontology selected in step one. The table shows the

selected elements. Only terminal child nodes of the tree can be selected and added, not the folders. The nodes are actually simple type (i.e. string, number, etc) elements of the model.

Users can add elements by first double-clicking them or selecting them and pressing the "Add" button. The chosen element is now added into the first empty row. The first column of the table shows the actual element and the second one a user-friendly name or label for this element. These labels must be unique. Users can also remove elements by selecting the target element and pressing the "Remove" button.

When finished adding elements, a new query template is created and stored to the local workspace once the "Finish" button is pressed. User can immediately continue creating a query based on this newly created template.

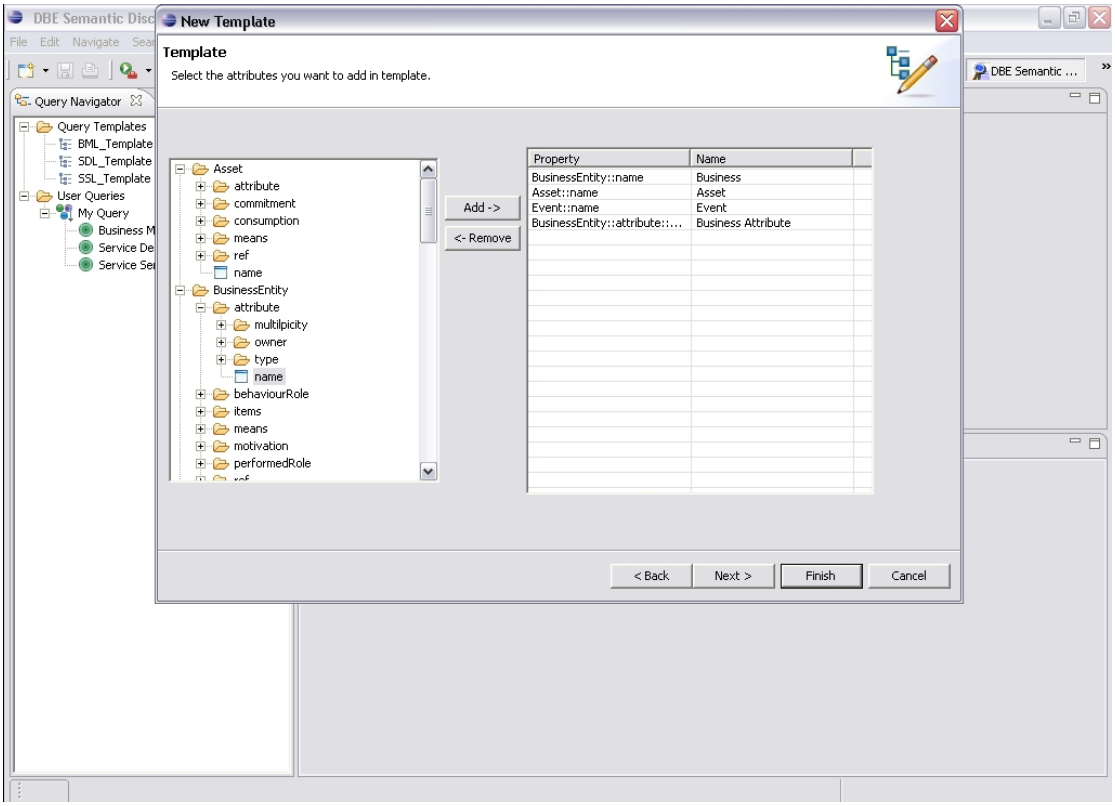

Figure 6: Second step of the New Template wizard; adding new elements.

3.3.2. Deleting a Template

To delete a template the user has to select it into the Query Navigator View and:

- Right click and "Delete Template", or
- Press keyboard "Delete"

If the user deletes a template that is referred by queries, the queries will not be able to open or be executed.

3.3.3. Editing a template

The user can view and add template elements of a created template through the template properties view as illustrated at *Figure 7*. In order for the template properties view to load information for a specific template one of the following should happen:

- Right-click on the template and from the menu “Open Template”,
- Double-click on the template,
- Press “Enter” on the keyboard when template is selected, and
- A query is open and has focus. In this case the template that is used by the query is shown to the user.

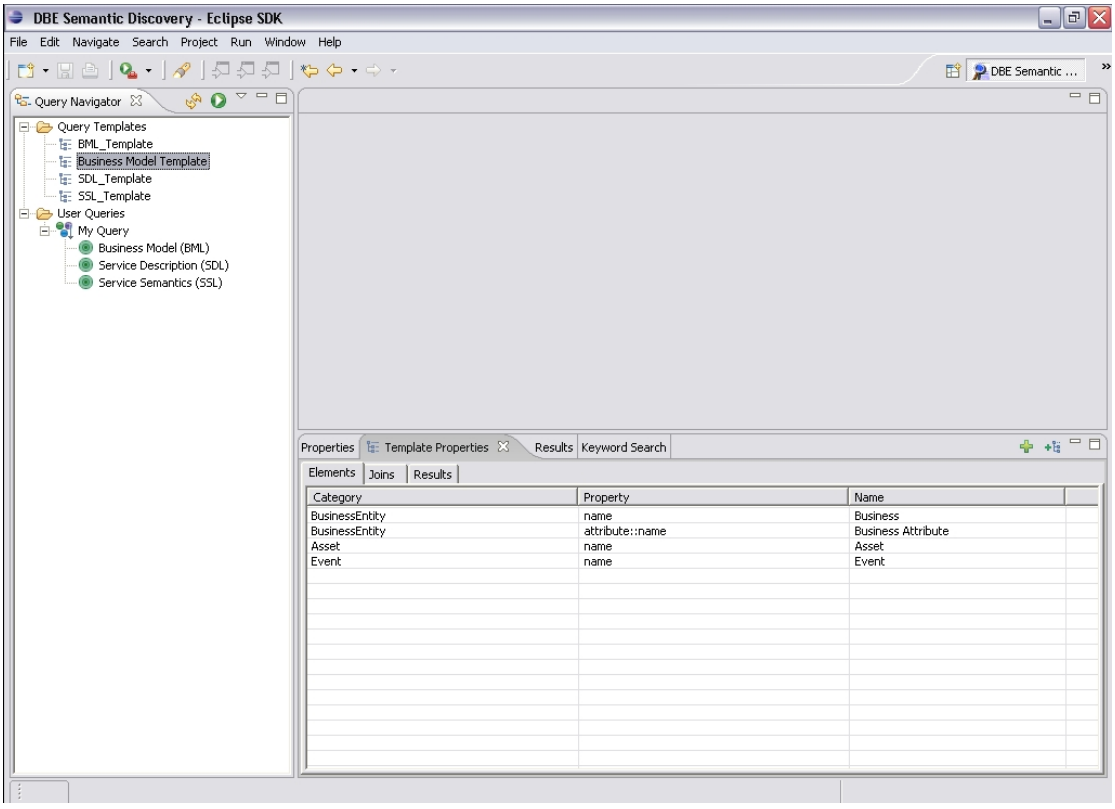

Figure 7: Template Properties View.

To edit a template the user can right-click on it and from the menu and select “Edit template”. Then, the second step of the template creation wizard is shown allowing adding, removing and editing template elements. In the Category column is shown the starting MOF element on which the constraint will be applied. The Property column refers to the path from the Category to the property of that element (i.e. “attribute::name”). The Name column refers to a user friendly name for that property.

3.4. Queries

3.4.1. Query Creation

A query contains constraints on characteristics defined by templates. A query always refers to a specific template. If no template exists or fulfils the user needs, a new template must be created. A query must be part of a query group. Groups help users organize their queries. To create a query either:

- Select “File -> New -> Query” from the main menu,
- Select “New Query” from the pull-down of the query navigator view,
- Right-click on a query or on a query folder and select “New query”,
- Right-click on a template and select “New query”

A “New Query” wizard opens with one step to be completed by the user and is illustrated in *Figure 8*. Information, warning and error messages guide and help users to create a query. The wizard asks the user to enter a name, a group and a template. When “Finish” is pressed, a file is created and opened for editing.

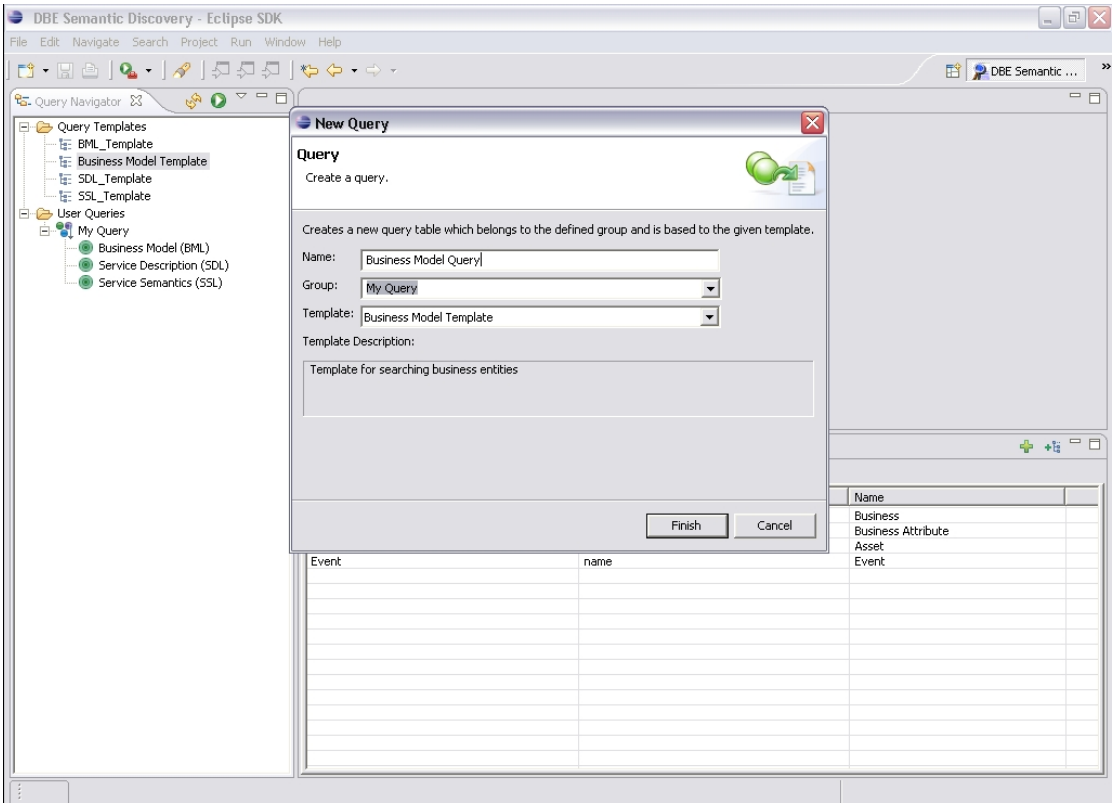

Figure 8: “New Query” wizard.

3.4.2. Query Editing

Queries are represented as tables (illustrated by *Figure 9*). Each column of the table refers to a template element (the elements' label is shown) and, thus, to a metamodel, model, or ontology element. Each value added on a column's cell is a criterion on the column's element which will be matched during the execution of the query. The criterion has a value, an operation (i.e. equals, larger than, like, etc), and a weight ranging from 1 to 100 representing how important this criterion is for the overall query. If a weight is small then results may not match it at all. By selecting a cell in the table, the properties view appears in the bottom of the screen and displays the data of the current cell allowing the user to edit them from here. To edit the cell inside the table the user can either double click on the cell, or press "F2" or "Enter".

The process for formulating the query follows the well-accepted Query-By-Example (QBE) [19] paradigm, i.e. elements lying in different columns of the same row of the table are connected together using a logical AND (conjunctive form), whereas elements lying in different rows are connected together using a logical OR (disjunctive form).

Note that the operation can be added inside the cell before the value. If no operation is added, equals is chosen.

To save a query the user should go from eclipse's main menu or use "Ctrl+S".

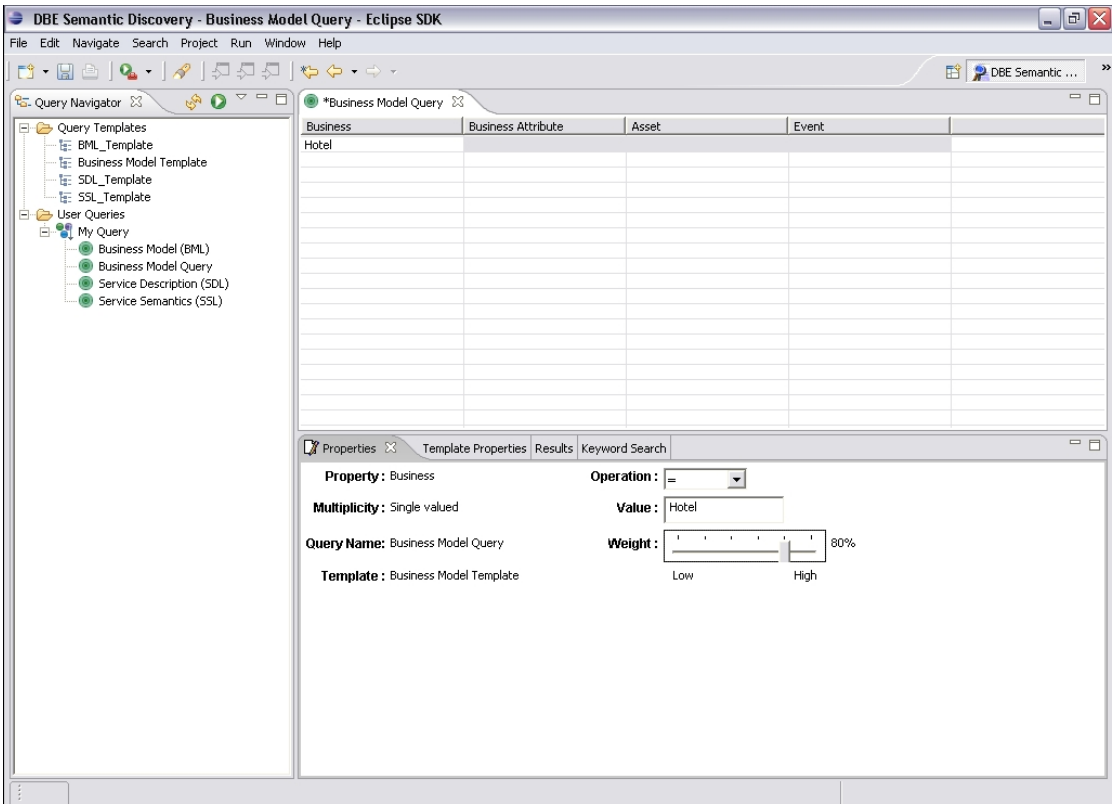

Figure 9: Query Editor and Properties View.

3.4.3. Query Deletion

To delete the user has to follow the same process as described at Deleting a Template (see subsection 3.3.2).

3.4.4. Executing a Query

After a query is created and edited it can be executed. During the execution query formulator formulates a valid QML query which is then sent by the proxy wrapper to the KB or SR, based on the query type. Afterwards, MOF manager collects the results in an asynchronous way. The results are shown at the Results View. To execute a query the user can either:

- Press “Execute editor query” at the toolbox of query navigator view,
- Press the “F5” key inside the query editor,
- Right-click on the query inside query navigator view and select “Execute this query” , or
- Right-click inside the query editor and select “Execute query”.

3.5. *Browsing Query Results*

When the query is executed results are fetched and shown in the results view as illustrated at *Figure 10*. The results are in three columns:

- The model or service manifest name. Different icons are shown for each one.
- The relevance rank to the query criteria. Results are always ordered by rank.
- A short description of the model or service manifest.

The result table is refreshed periodically as new results are calculated at the repository. This is performed as the information is not stored centrally but distributed between peers and as time elapses new results may be delivered. From the view toolbar, the user can stop fetching results, refresh at his/her own will, and clear the result table.

User can browse the returned models or the service manifests returned to further explore their contents and decide if they are the most appropriate results. If the result is a model the user can browse all the information that comprises it or see which service manifests have adopted this model and then browse the service manifests. By exploring a service manifest, the user can see:

- The Service Semantics Model (SSL part of the Service Manifest),

- Business Model (BML part),
- Service Description (SDL part), and
- Business Description (The data of the Service Manifest),

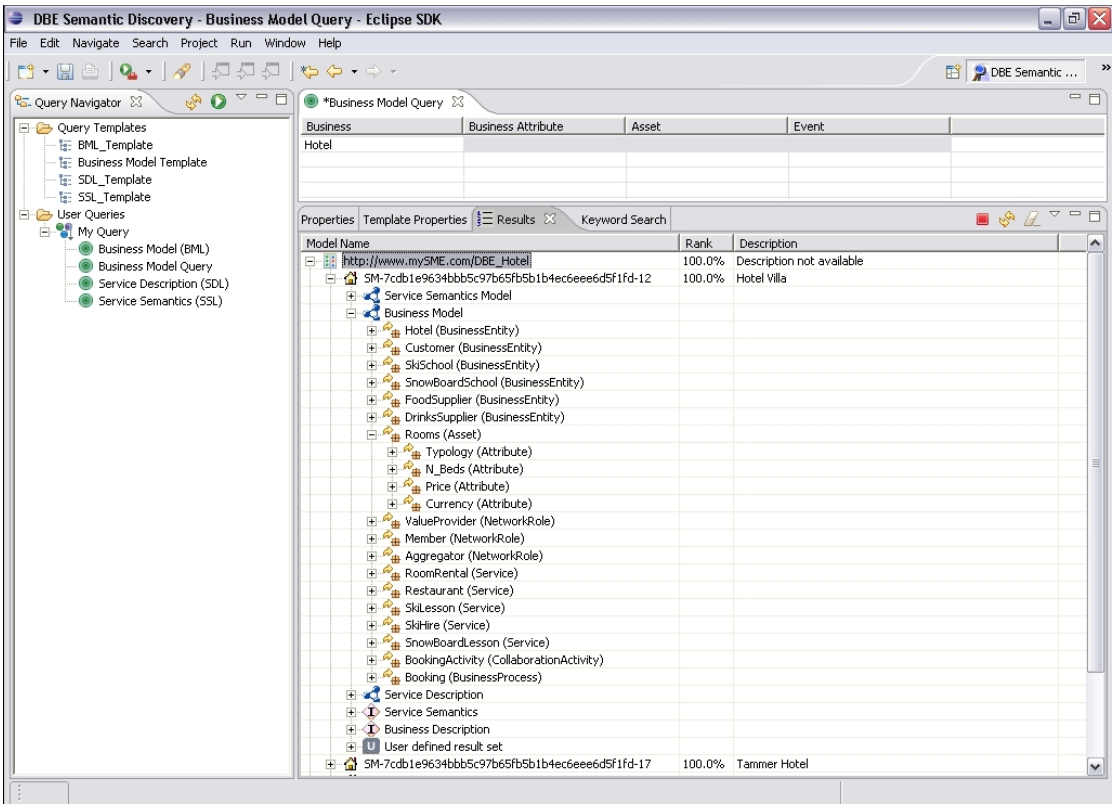

Figure 10: Browsing results.

3.5.1. Operations on results

Results are needed in order to develop new DBE services or execute existing DBE services. Both these functionalities are provided through the Results View. Depending on the type of the result (BML, SSL, SDL model or Service Manifest) different operations may apply. Other DBE Studio plugins register the applicable operations which are shown in a drop-down menu. The default operations registered by the tool itself are:

- **"Execute Service":** is only applicable to service manifests (shown in *Figure 11*). It opens a browser if an OpenLaszlo interface is provided or the Java Swing interface.
- **"Search by Model"**: Another operation provided by the tool is to search by using a specific model. This allows the user find a model and then create a template and a query for instances following it.

Another operation provided by DBE Studio plugin "BML Editor" is the "Open BML Model", which opens a BML model into the BML Editor Perspective (shown in *Figure 12*).

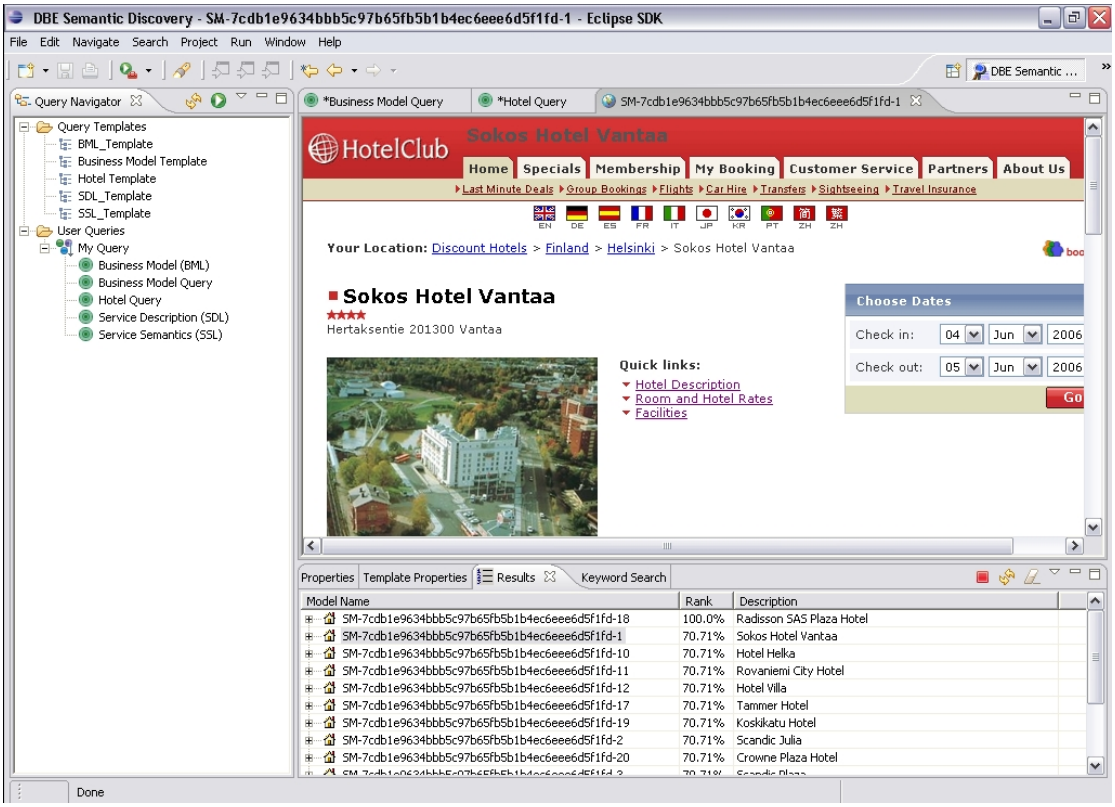

Figure 11: Service Execution.

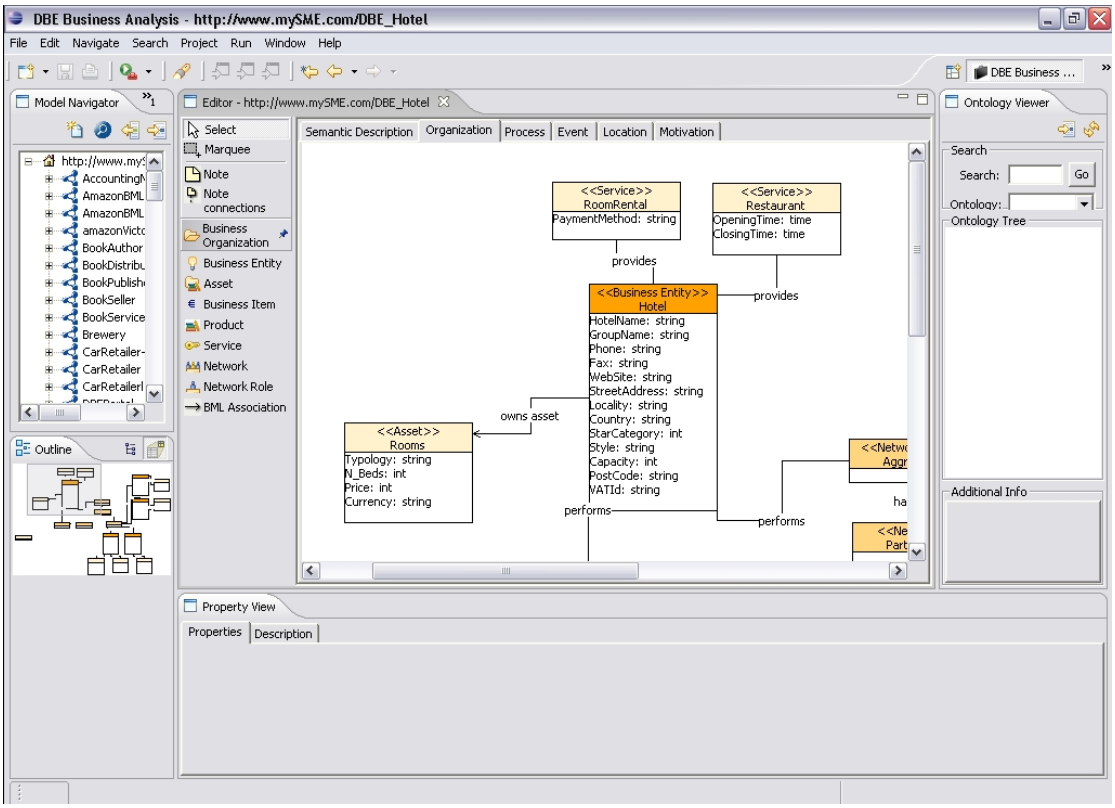

Figure 12: Open BML Model inside the DBE Business Analysis Perspective

3.6. Keyword based Search

Instead of creating queries and templates a simpler approach is to specify just a set of keywords. Using the first approach, results are more relevant to the needs of the user because in this way a user can say that he/she wants “Athens” to be the city name where a hotel is located. This is a feature not available to keyword-based searches. The keyword search is available from the keyword search view, illustrated in *Figure 13*. The results are shown inside the same view and not at the results view. The results can be browsed the same way as at results view.

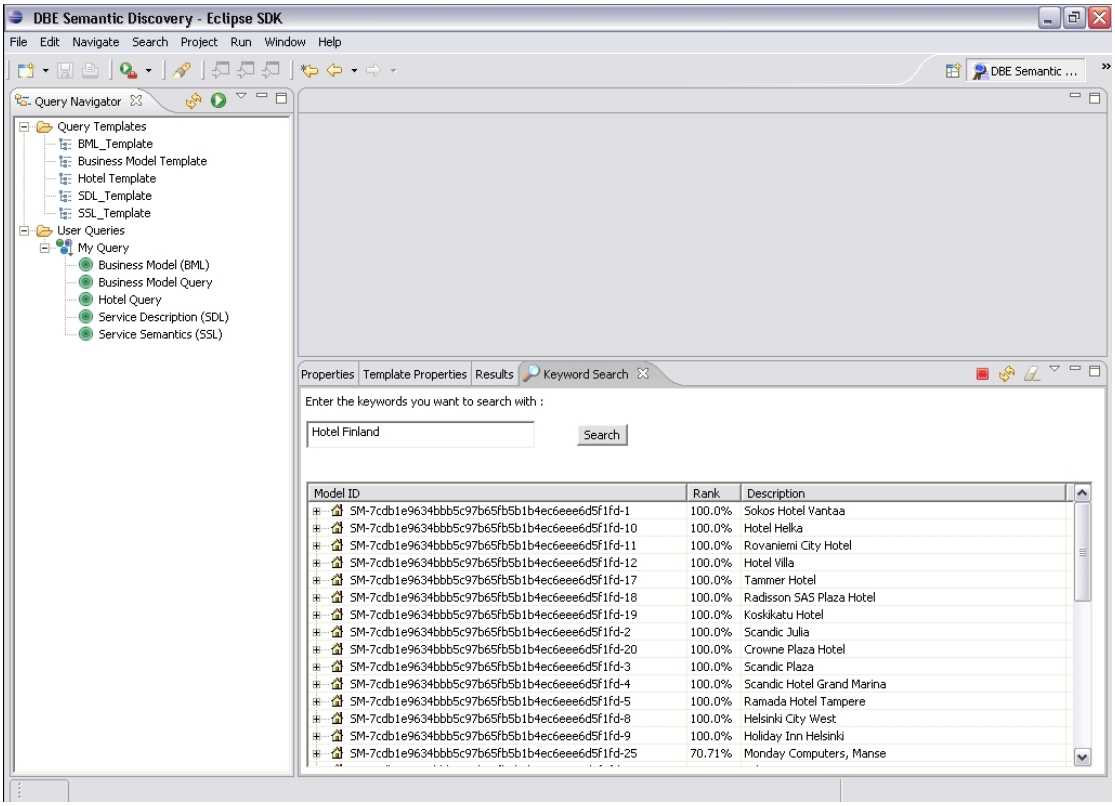

Figure 13: Keyword Search View.

A more sophisticated approach, rather than using single keywords as “Athens” and “Hotel” is to use path expressions, i.e. to specify the path that a term should match. Note that “path expressions” does not refer to “XPath expressions”. For example the expression “Hotel/Location=Athens” would require the value “Athens” to be found under the model or ontology path “Hotel/Location”. In this way the search engine is guided to where exactly the keyword “Athens” should appear. Other operators are also supported; e.g. another example using different operations and numeric values could be “Hotel/StarCategory<5”. Note that “Hotel” does not refer explicitly to a BussinessEntity or any other element of BML metamodel but on any type of element (i.e. even an ontology element) that is named “Hotel”.

3.7. Preferences

Selecting from the main menu "Window -> Preferences ->DBE Studio->Semantic Discovery Tool" the user can see the preferences of the tool, as illustrated at *Figure 14*. The user can optionally override the servent URL he/she wants to connect to, instead of using the globally defined servent URL ("Window -> Preferences ->DBE Studio"). A KB and/or a SR service must be deployed on that servent. The user can select if he/she wants to connect to a KB or a SR by selecting:

- Search for models (Knowledge Base)
- Search for services (Service Registry)

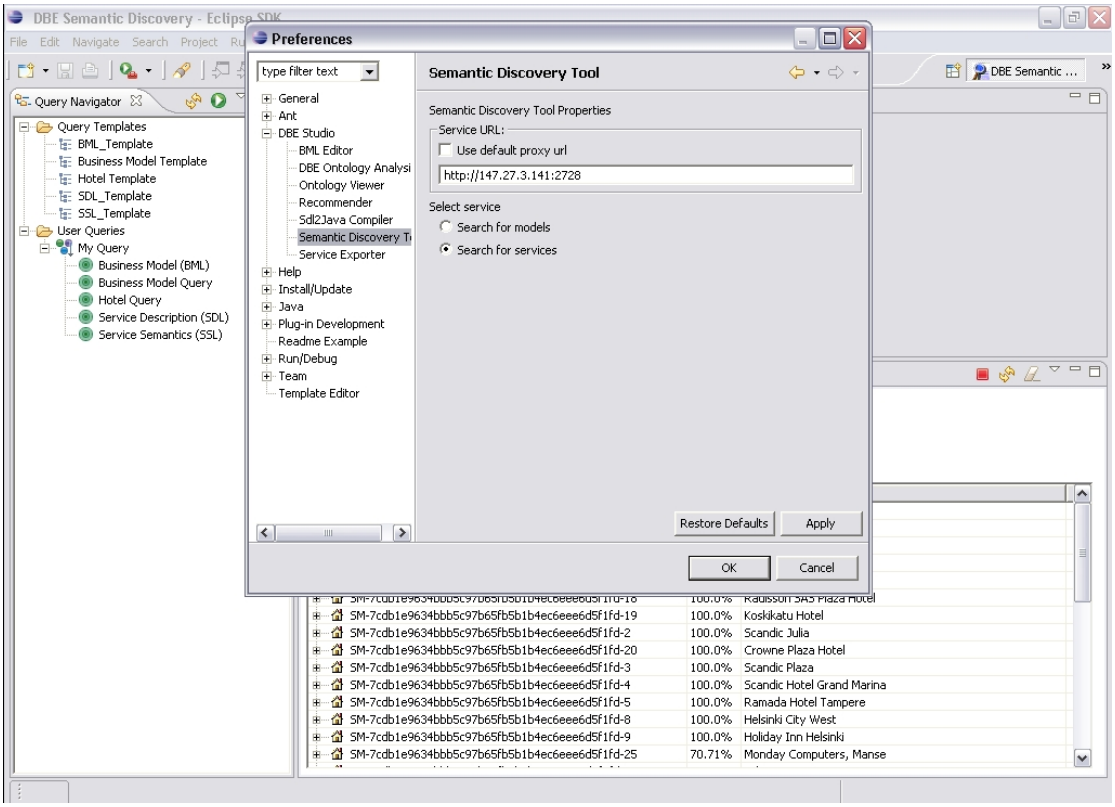

Figure 14: DBE Semantic Discovery Tool Preferences.

4. DBE Service Discovery Portal User Guide

The DBE Service Discovery Portal is used by end-users in order to find and execute a DBE service. The user can search with a simple keyword-based search or an advanced search based on a model. The portal implementation is developed using the OpenLaszlo technology [11].

The main page is illustrated in *Figure 15* and enables users to search using keywords. Path-based expressions for specifying keywords (described in section 3.6) are also supported. The results appear as illustrated in *Figure 16* and are ordered by their relevance to the query. They are fetched asynchronously from the Service Registry. Users can refine their search criteria from the results page.

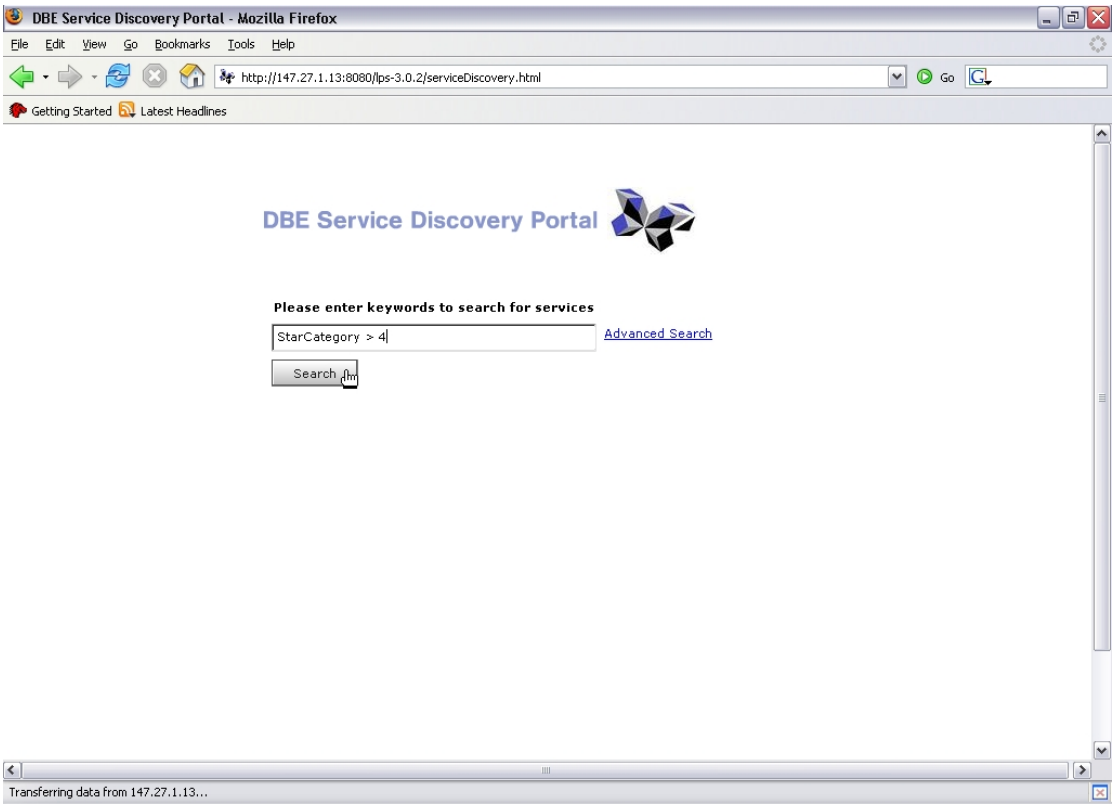

Figure 15: The main page of the Semantic Discovery Tool Portal, keyword search is available from here.

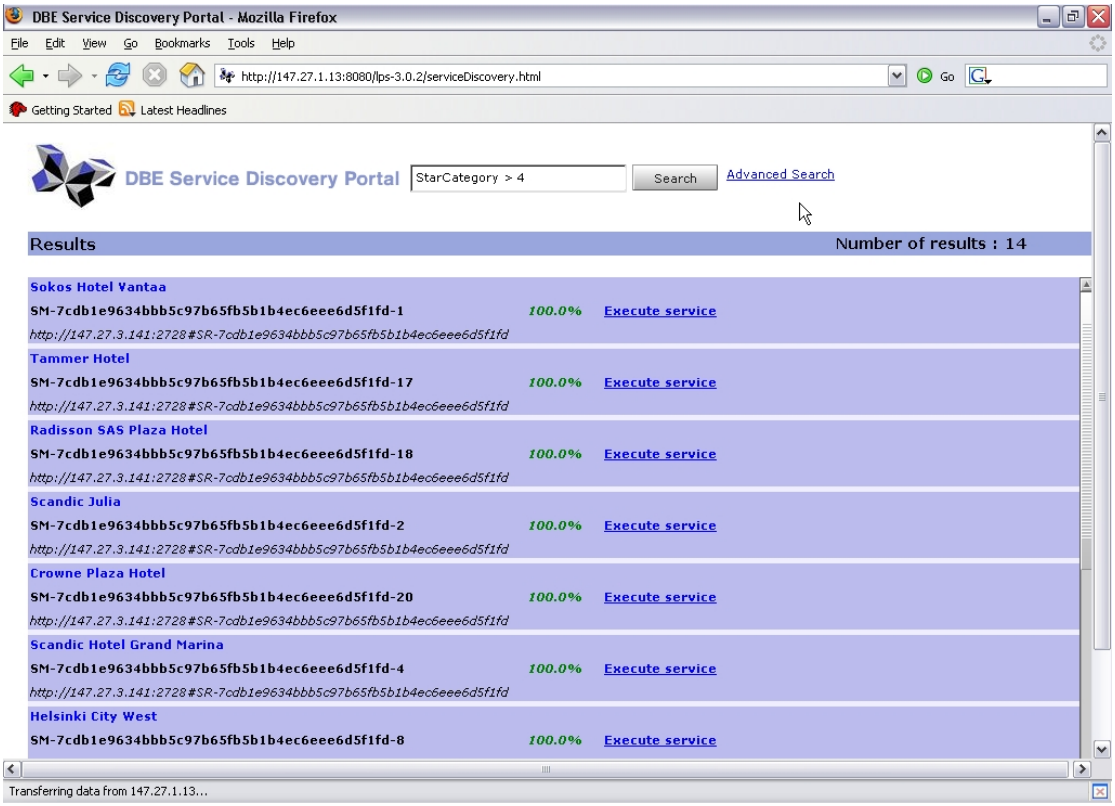

Figure 16: Results page.

4.1. Advanced Search

If the user needs to specify some semantics on the query terms, he can use the advance search. The advance search page contains on the left a model selection part, on the middle a model explorer part and on the right the criteria part, as illustrated in *Figure 17*. The user using keywords can search for models and when he selects one of them from the list the model is loaded into the middle part as a tree. The user can browse the model and select the desired leaves. When a leaf is selected it is added into the right part, the criteria part. Users can add a value, an operation and a weight regarding the importance of the criterion in the whole query. When the process of adding criteria is finished, the user can press the “Search” button to execute the query.

Internally a template and a query are created in order to formulate the query into QML and send it to the SR for execution. Thus, the advance search page is a different way to create a template with fewer capabilities than the eclipse version of the tool.

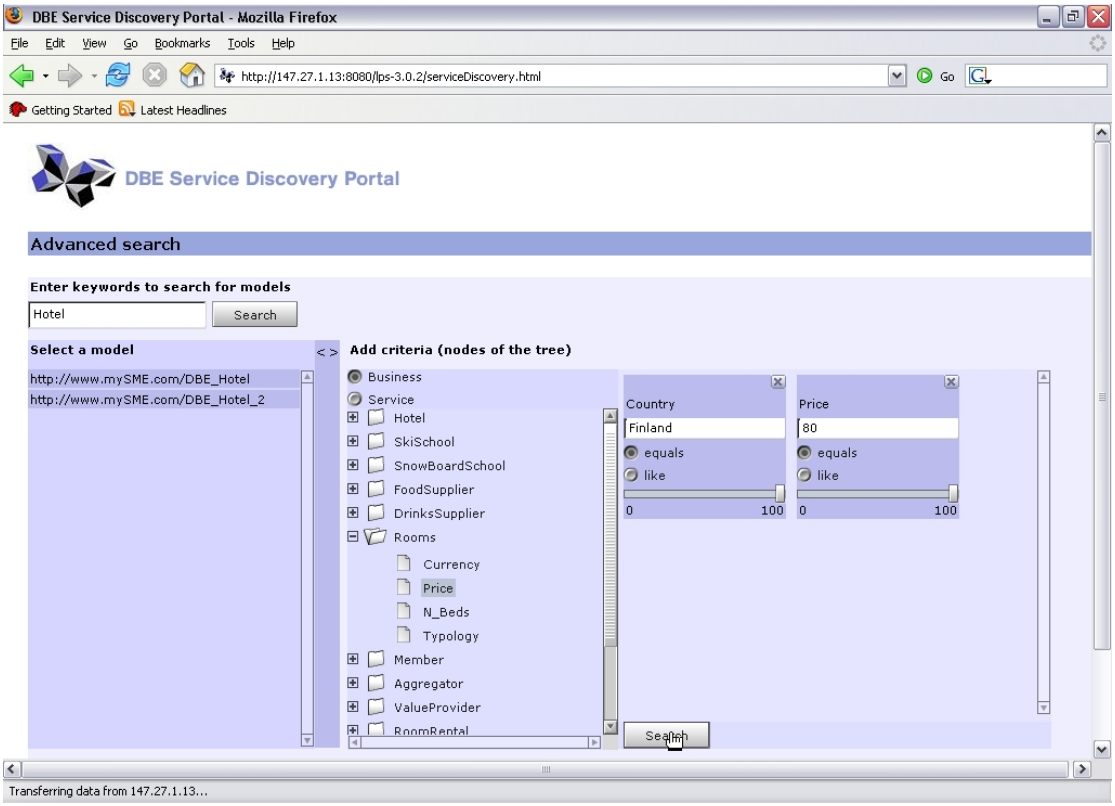

Figure 17: Defining criteria values.

4.2. Operations on Results

The user can click a result item to see an indicative part of the service manifest information. As illustrated at *Figure 18*, a new window opens with the appropriate details.

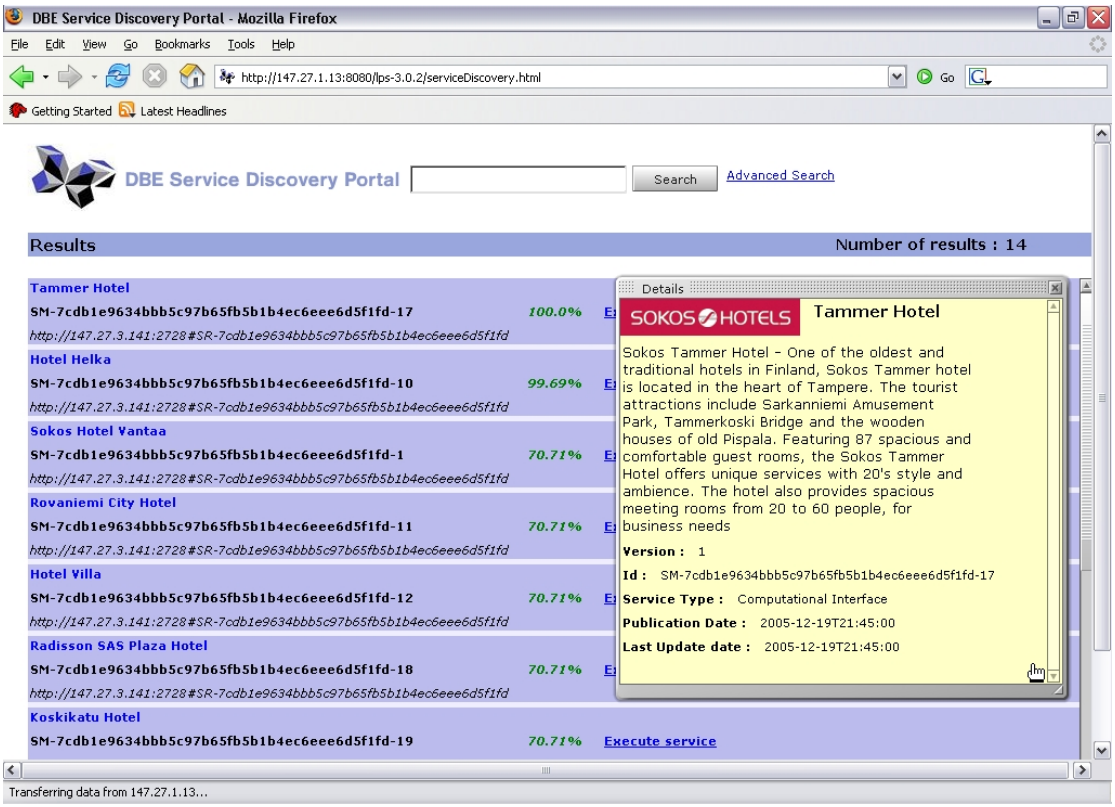

Figure 18: Details window for a results item.

Moreover, a user may need to execute the service described in the resulted service manifest by selecting the “Execute Service” link. Then the actual service will be contacted and a web interface (if available) will be loaded in another web browser, as illustrated in *Figure 19*.

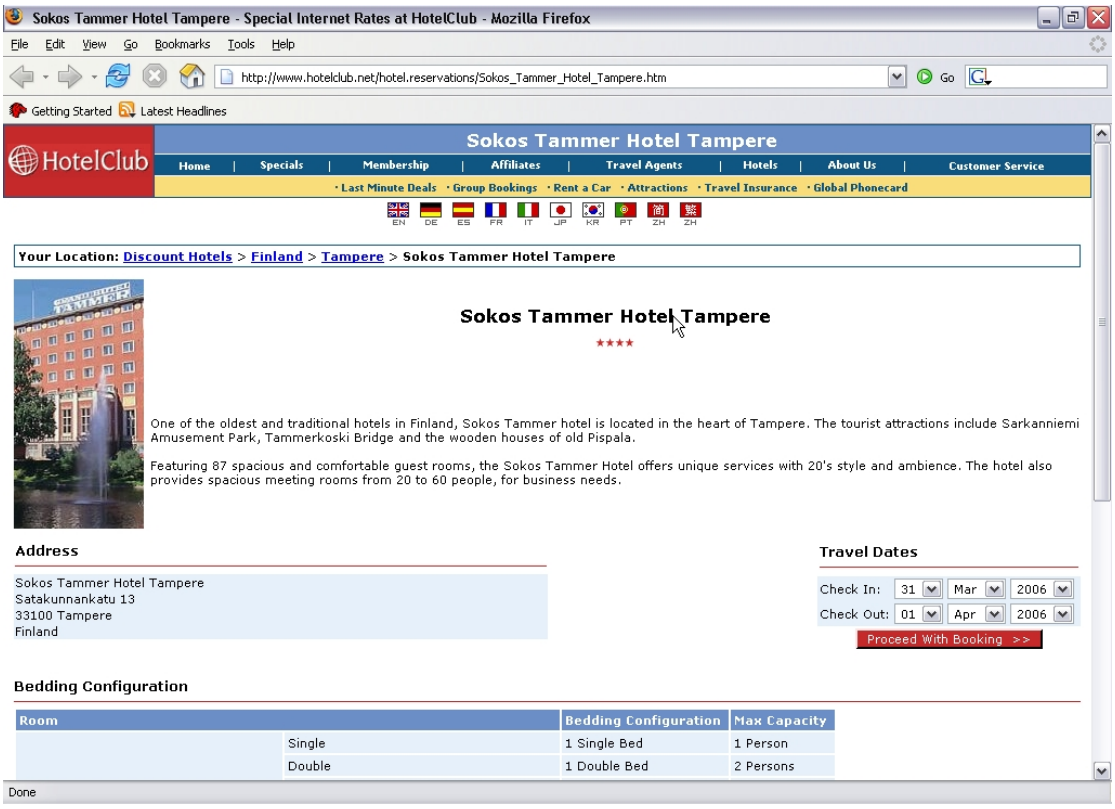

Figure 19: Executing a service.

5. Glossary

Term	Description
API	Application Programming Interface: Is a technology that facilitates exchanging messages or data between two or more different software applications
BML	Business Modeling Language
CIM	Computational Independent Model: The most abstract layer in the MDA architecture. Models of this layer describe a modelled system from the business point of view (i.e. requirements, abstract functions, etc.) without any assumption on the implementation (software or not) details.
KB	Knowledge Base: the part of the DBE system where DBE knowledge is stored and managed. Such knowledge refers to ontologies, business and service descriptions, etc.
KB Service	Knowledge Base Service: A Service on top of the DBE Knowledge Base that provides functionality for storing and retrieving models.
MDA	Model Driven Architecture: An approach (proposed by OMG) to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology.
MDR	Meta-Data Repository: MDR implements the OMG's MOF standard based metadata repository based on the JMI specification
MOF	Meta Object Facility: A generalised facility for specifying abstract information about very concrete object systems.
MOF Repository	A repository for storing, managing and retrieving meta-data (models) and meta-meta-data (metamodels) that have been described with MOF.
OCL	Object Constraint Language: OMG's standard for expressing constraints and well-formedness rules on object models. The latest release is also considered suitable for querying object models.
ODM	Ontology Definition Metamodel: A MOF model (metamodel) developed in the DBE for ontology representation according to the corresponding OMG RFP
OMG	Object Management Group: International standardisation body
P2P	Peer-To-Peer
PIM	Platform Independent Model of a modelled system
PSM	Platform Specific Model of a modelled system
QML	Query Metamodel Language: It is a Knowledge Access Language developed in DBE in order to provide uniform access to various kinds of DBE knowledge.
Recommender	A DBE (autonomous) Core Service that will provide users (SMEs) with personalised knowledge by exploiting their profiles

Term	Description
SDL	Service Description Language: A MOF model (metamodel) that provides technical description of the programmatic interface of a service
Semantic Registry	The component of the DBE Knowledge Base that hosts the published services (in the form of Service Manifest Documents).
Service Manifest (SM)	The Service Manifest is a two-fold formal description of a specific DBE Service, and contains both the models and data (comprising both business and technological information) of a single specific real word service owned by a specific SME.
SME	Small and Medium Enterprise: Independent enterprise with less than 250 dependent.
SR	Semantic Registry: It is the component of the Knowledge Base that hosts the service descriptions published in the DBE environment and available for discovery and consumption.
SSL	Semantic Service Language: A MOF-based language for semantically describing SME services in DBE.
UML	Unified Modeling Language: A method for specifying, visualizing, and documenting the artefacts of an object-oriented system under development; as well as for business modelling.
XMI	XML Metadata Interchange: An SMIF standard specification based on XML.
XML	eXtensible Mark-up Language. XML is a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere.

6. References[1]

1. TUC, DBE Deliverable, D14.1 – DBE Knowledge Representation Models, May 2005
2. TUC, DBE Deliverable, D17.1 – Recommender, March 2005
3. TUC, DBE Deliverable, D14.3 1st P2P Distributed Implementation of the DBE KB and SR, December 2005
4. TUC, DBE Deliverable, D14.4 Second Release of Recommender, December 2005
5. Object Management Group (OMG), "Meta Object Facility(MOF) Specification," (2002), version 1.4, http://www.omg.org/
6. Boldsoft, International Business Machines Corporation, IONA and Adaptive Ltd. "OCL 2.0 OMG Final Adopted Specification", (OMG Document ptc/03-10-14), October 2003
7. OMG XML Metadata Interchange (XMI) Specification v1.2 http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf, 2002
8. MDA Guide Version 1.0.1: http://www.omg.org/docs/omg/03-06-01.pdf, 2003
9. Metadata Repository (MDR) Project. http://mdr.netbeans.org/
10. IBM Corporation, "Eclipse Platform Technical Overview", http://www.eclipse.org 2003
11. OpenLaszlo http://www.openlaszlo.org
12. Trygve Reenskaug "MODELS - VIEWS - CONTROLLERS" (MVC), http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf, 1979
13. Object Management Group – OMG, "MetaObject (MOF) Facility Specification", http://www.omg.org/mof, 2002
14. INTEL, DBE Deliverable, D26.3 DBE Studio Integration, April 2006
15. INTEL, DBE Deliverable, D26.6 DBE Portal Specification, February 2006
16. FZI, DBE Deliverable, D7.2 - Initial Description of Profiling mechanism design and rationale with respect to one or two use cases: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID=361&expandFolder=361&folderID=328, October 2005
17. ISUFI, DBE Deliverable: D15.1 - BML First Release
18. SOLUTA.net, DBE Deliverable, D16.1 – Service Description Models and Language Definition
19. M. M. Zloof. "Query by example: A database language", 1977 http://www.research.ibm.com/journal/sj/164/ibmsj1604C.pdf
20. Macromedia Flash, http://www.macromedia.com/software/flash/flashpro

[1] DBE deliverables are available at the DBE official site http://www.digital-ecosystem.org

7. Appendix A - The Query Formulator API

Interface IQueryFormulator

Title: QML Formulation API

Description: An API for formulating QML queries and Expressions (**org.dbe.kb.qi**)

Field Summary	
static int	**AND**
static int	**OR**

Method Summary	
void	**clearQuery**() All the objects created so far by the formulator are clear from the MDR Repository.
org.dbe.kb.metamodel.qml.contextdeclarations.InvariantContextDecl	**formulateConstraint**(java.util.Collection path, java.lang.String operation, java.lang.String value) This method formulates a constrained OclExpression from a Vector of Mof Classes, a string representation of an operation, e.g. "=", and a String value.
org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression	**formulateExpression**(java.util.Collection path, java.lang.String operation, java.lang.String value) This method formulates a constrained OclExpression from a Vector of Mof Classes, a string representation of an operation, e.g. "=", and a String value.
org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression	**formulateExpressions**(java.util.Collection expresions, int type) Formulates a conjunctive or disjunctive OCL expression
org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression	**formulateFuzzyExpression**(java.util.Collection exprs, double[] weights, int type) Formulates a fuzzy conjunctive or disjunctive OCL expression
org.dbe.kb.metamodel.qml.contextdeclarations.QueryContextDecl	**getQuery**(java.lang.String name, org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression body, java.lang.String result) Constructs QML expressions for fuzzy query

Class **QueryFormulator**
Title: QML Formulation API
Description: An API for formulating QML queries and Expressions (**org.dbe.kb.qi**)

Method Summary	
void	**clearQuery**() Keeps track of all Objects created by the formulator and clears every time needed.
void	**copyPackage**(java.lang.String fromExtend, java.lang.String toExtend, javax.jmi.reflect.RefPackage fromPackage) Copies one RefPackage from the MDR extend to the toExtend.
org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression	**formulateExpressions**(java.util.Collection expresions, int type) Formulates a conjunctive or disjunctive OCL expression
org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression	**formulateFuzzyExpression**(java.util.Collection expresions, double[] weights, int type) Formulates a fuzzy conjunctive or disjunctive OCL expression
org.dbe.kb.metamodel.qml.contextdeclarations.QueryContextDecl	**getQuery**(java.lang.String name, org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression exp, java.lang.String result) Constructs QML expressions for fuzzy query

Class **ModelQueryFormulator**

Title: QML Formulation API

Description: An API for formulating QML queries and Expressions (**org.dbe.kb.qi**)
This class creates model queries.

Constructor Summary

`ModelQueryFormulator(org.dbe.kb.metamodel.qml.QmlPackage qml)` Creates a new Query instance to be used later on.	

Method Summary

`org.dbe.kb.metamodel.qml.contextdeclarations.InvariantContextDecl`	`formulateConstraint(java.util.Collection path, java.lang.String operation, java.lang.String value)` Formulates a QML constraint
`org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression`	`formulateExpression(java.util.Collection path, java.lang.String operation, java.lang.String value)` Formulates a QML constraint

Class **InstanceQueryFormulator**

Title: QML Formulation API

Description: An API for formulating QML queries and Expressions (**org.dbe.kb.qi**)
This class creates instance queries.

Constructor Summary

`InstanceQueryFormulator(org.dbe.kb.metamodel.qml.QmlPackage qml)` Creates a new Query instance to be used later on.

Method Summary

`org.dbe.kb.metamodel.qml.contextdeclarations.InvariantContextDecl`	`formulateConstraint(java.util.Collection path, java.lang.String operation, java.lang.String value)` Formulates a QML constraint
`org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression`	`formulateExpression(java.util.Collection path, java.lang.String operation, java.lang.String value)` This method formulates a constrained OclExpression from a Vector of Mof Classes, a string representation of an operation, e.g. "=", and a String value.
`org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression`	`formulateExpression(java.lang.String[] path, java.lang.String operation, java.lang.String value)` This method formulates a constrained OclExpression from a Vector of Mof Classes, a string representation of an operation, e.g. "=", and a String value.
`org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression`	`refineInstanceQuery(org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression hard, org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression soft)`

Class **AdvancedQueryFormulator**

Title: Advanced QML Formulation API

Description: An Advanced API for formulating QML queries and Expressions (**org.dbe.kb.qi.adv**)

Field Summary

`static int`	`INSTANCE_QUERY`
`static int`	`MODEL_QUERY`

Constructor Summary

AdvancedQueryFormulator(org.dbe.kb.metamodel.qml.QmlPackage qmlPackage, int type) Creates a new Advanced Query Formulator

Method Summary

org.dbe.kb.metamodel.qml.contextdeclarations.QueryContextDecl	**getQuery**(QueryExpr[] expressions) Creates and returns a QueryCoonextDecl class.
Template	**getTemplate**() Gets the template of the formulator.
void	**setTemplate**(Template template) Sets a template to the formulator

Class **QueryExpr**

Title: Advanced QML Formulation API

Description: An Advanced API for formulating QML queries and Expressions (**org.dbe.kb.qi.adv**) The objects of this class are actual query expressions

Constructor Summary

QueryExpr()
QueryExpr(java.lang.String operation, java.lang.String id, java.lang.String value, double weight) Creates a new Query Expression for a specific operation, template element id, value and weight

Method Summary

java.lang.String	**getOperation**()
java.lang.String	**getTemplateElementId**()
java.lang.String	**getValue**()
double	**getWeight**()
void	**setOperation**(java.lang.String operation)
void	**setTemplateElementId**(java.lang.String templateElementId)
void	**setValue**(java.lang.String value)

void	**setWeight**(double weight)

Class **Template**

Title: Advanced QML Formulation API

Description: An Advanced API for formulating QML queries and Expressions (**org.dbe.kb.qi.adv**) This class denotes a reusable query component.

Constructor Summary

Template()

Method Summary

void	**addTemplateElement**(TemplateElement te) Adds a template element to the template
java.lang.String	**getDescription**() Gets the template's description
TemplateElement	**getTemplateElement**(int index) Gets the template Element at the specified index
java.util.Vector	**getTemplateElements**() Gets a collection of the template elements
void	**setDescription**(java.lang.String description) Sets the template's description

Class **TemplateElement**

org.dbe.kb.qi.adv

Constructor Summary

TemplateElement()
TemplateElement(java.lang.String id, java.lang.String path, java.lang.String type)

Method Summary

javax.jmi.model.MofClass	**getContext**()
java.lang.String	**getDelimeter**()

java.lang.String	**getId**()
java.lang.String	**getPath**()
java.lang.String	**getType**()
void	**setContext**(javax.jmi.model.MofClass context)
void	**setDelimeter**(java.lang.String delimeter)
void	**setId**(java.lang.String id)
void	**setPath**(java.lang.String path)
void	**setType**(java.lang.String type)

8. Appendix B – Register Discovery Tool API

This is an API meant to be used by other eclipse plugins in order to register operations on the query results. This API is already used by many DBE Studio plugins as “BML Editor”.

org.dbe.studio.tools.discovery

Class RegisterDiscoveryTool

public class **RegisterDiscoveryTool**
extends java.lang.Object

The RegisterDiscoveryTool class is a tool for other modules to communicate with the Formulator. Other modules can register their proxy, in order Formulator not to search on a FADA server for another one. They can also register an implementation of DiscoveryListener. The listeners are invoked when user double clicks a result. Also the result is kept after user has pressed finish and can be read asynchronously by the module whenever desired.

Field Summary

static int	**BML_MODELS_TYPE** Used for only for BML models
static int	**MODELS_TYPE** Used for all Models is equal to BML_MODELS_TYPE | SSL_MODELS_TYPE | SDL_MODELS_TYPE
static int	**SDL_MODELS_TYPE** Used for only for SDL models
static int	**SERVICES_TYPE** Used for Service Manifests
static int	**SSL_MODELS_TYPE** Used for only for SSL models

Constructor Summary

RegisterDiscoveryTool()

Method Summary

static void	**deregisterListener**(java.lang.Class listenerClass) deregisters a listener of type Class
static void	**deregisterListener**(DiscoveryListener listener) deregisters a listener
static java.util.Vector	**getListeners**(int type)
static java.lang.Object	**getResult**() Gets the model id of the selected result.
static void	**registerListener**(DiscoveryListener listener)

	Deprecated. *Please use registerListener(DiscoveryListener, String, ImageDescriptor,int)*
static void	**registerListener**(DiscoveryListener listener, java.lang.String message, org.eclipse.jface.resource.ImageDescriptor image) **Deprecated.** *Please use registerListener(DiscoveryListener, String, ImageDescriptor,int)*
static void	**registerListener**(DiscoveryListener listener, java.lang.String message, org.eclipse.jface.resource.ImageDescriptor image, int type) Register's a listener.
static void	**setResult**(java.lang.Object mid) Sets the model id of the selected result.

org.dbe.studio.tools.discovery

Interface DiscoveryListener

---

public interface **DiscoveryListener**

The Discovery Listener. This listener is invoked when a user selects a result. You can register this Listener at the RegisterDiscoveryTool class.

See Also:

org.dbe.dtool.RegisterDiscoveryTool

Method Summary

void	**getResults**(java.lang.Object modelId) Overload this method to get the modelID of the model selected by the user.