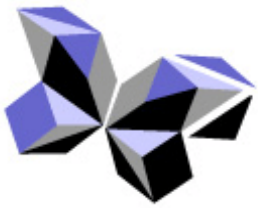

Digital Business Ecosystem

Contract n° 507953

# WP14: DBE Knowledge Base

# D14.2: 1st Prototype Implementation of the DBE Knowledge Base

Information Society Technologies

Project funded by the European Community under the "Information Society Technology" Programme

**Contract Number**: 507953

**Project Acronym**: DBE

**Title**: Digital Business Ecosystem

**Deliverable N°**: D14.2

**Due date**: 31/4/2005

**Delivery Date**: 27/04/2005

**Short Description**:

This document describes the design and the first prototype implementation of the DBE Knowledge Base (KB). The KB architecture adopts the MOF metadata framework of OMG and the implementation exploits a powerful combination of a metadata repository and a relational database system. The KB incorporates all the MOF metamodels that are currently available in DBE and provides useful functionality for managing, accessing and discovering knowledge. In this document two important DBE core services, the Knowledge Base Service (KB service) and the Semantic Registry Service (SR service), are also presented. These services exploit the basic functionality of the DBE KB infrastructure and are used to support important tasks in the Service Factory environment (KB service) and Service Execution environment (SR-service). DBE tools and components that wish to access the KB or SR service are able to lookup the available proxies and communicate with the services. The KB and SR proxies are intelligent proxies that can operate as simple communication clients or as effective development toolkits for supporting the development of DBE tools that aim to manage DBE models and data.

The knowledge access mechanisms that are mentioned in this document provide the underlying mechanism for querying metamodels, models and instances of DBE. It also forms the basis for supporting recommendations. This document accompanies the current implementation of the Knowledge Base that was demonstrated in the review of DBE of January 2005 as part of the integrated first year prototype of DBE.

**Partners owning**: TUC

**Partners contributed**: TUC

**Made available to**: All project partners and the EC

**Versioning**

| Version | Date | Author, Organisation | Description |
|---|---|---|---|
| 0.1 | 07/04/05 | NIKOS PAPPAS – TUC<br>FOTIS G. KAZASIS – TUC<br>GIORGOS ANESTIS – TUC<br>NEKTARIOS GIOLDASIS - TUC | Initial Document Creation |
| 0.2 | 22/04/05 | PROF. STAVROS CHRISTODOULAKIS - TUC | Revisions/Additions |
| 0.3 | 12/05/05 | NEKTARIOS GIOLDASIS – TUC | Revisions according to internal Review Process |

**Quality check**

**1st Internal Reviewer:** John Kennedy – INTEL

**2nd Internal Reviewer**: Miguel Vidal – SUN

# 1st Prototype Implementation of the DBE Knowledge Base

## Table of Contents:

## List of Figures:

# Executive Summary

This document describes the design and implementation of the DBE Knowledge Base (KB). This implementation was demonstrated in January 2005, during the DBE review, as part of the integrated first year prototype of DBE.

The architecture of the DBE Knowledge Base is presented and the constituent components and their interactions are described in detail. The implementation efforts lead to the provision of a basic knowledge management infrastructure that can be used to support the important core services: the Knowledge Base Service (KB-Service) and the Semantic Registry Service (SR-Service). The KB service is exploited in the Service Factory environment for storing and discovering business models while the SR service aims to support the registration and semantic discovery of DBE services in the Service Execution environment, at runtime.

The KB infrastructure is based on the OMG MOF metadata architecture [2]. The prototype provides support for handling metamodels, models and data, exploits a metadata repository and organises information using a data management system.

The KB and SR proxy (both of which are also presented in this document) were developed to enable the communication between DBE tools and components and the KB and SR services respectively. These proxies export the offered functionality and interact with the services mainly by exchanging XMI [3] documents.

The knowledge access mechanisms that are mentioned in this document provide the underlying mechanism for querying metamodels, models and instances of DBE. They also form the basis for supporting recommendations (i.e. expressing and matching preferences of users) and are presented in detail in D17-1 [15].

The first version of the DBE Knowledge Base and Semantic Registry are centralised. However, the goal during the design was to keep in mind that this infrastructure will be distributed in the future over a P2P network and the appropriate design and implementation decisions were taken that will facilitate the P2P deployment as much as possible.

Finally in this document the functionality provided by the KB and SR services is described and the interfaces of these services are presented in detail.

# 1. Introduction

The DBE Knowledge Base (KB) aims to provide a common and consistent description of the DBE world and its dynamics, as well as the external factors of the biosphere affecting it.

The content of the KB includes representations of the SME Business and Service Models, Domain Ontologies, SME Profiles and Views, the Service Usage and other kinds of information. The KB is being used in order to provide a consistent knowledge model and input for the Service Description / Business Modelling Language, the Recommendation process and the Service Composition process [16].

An important aspect of the KB management is to support the sharing of its content, and so it follows a scalable design/implementation that can support virtually any number of DBE users. From an abstract point of view, the DBE Knowledge Base is supposed to handle the DBE Knowledge. However, both the desired contents of the knowledge base and the functionality exposed may vary according to the environment that is manipulating it (Service Factory or Service Execution). For supporting these diverse goals, the DBE knowledge base offers different kinds of services for different purposes. There are services offering functionality for the Service Factory Environment (KB Service), services that offer functionality for the Execution Environment (e.g. Semantic Registry Service), and services that can be used within both environments (e.g. Recommender).

In an environment where the nature and the exploitation of knowledge are so disparate, it is important to keep the complexity of the system components as low as possible in order to allow for smooth integration and maintenance of the entire system. Thus, the fact that the DBE Knowledge Base can effectively satisfy the requirements coming from both Service Factory and Service Execution environments is of high importance. An environment with many different persistent components which are based on different technologies and provide diverse interface would cause an exponential complexity of the system making its operation, maintenance and extension extremely complex tasks.

In the following sections the KB architecture and functionality is presented as well as the main use cases supported by the first implementation prototype.

# 2. The KB Functionality

In this section we will describe, at a high level, the functional specification of the DBE Knowledge Base. The purpose of this document is to describe only the uses cases that are currently supported by the 1st implementation prototype and not the extensive use cases that are foreseen in the DBE project.

Initially the main set of use cases that involve the DBE Knowledge Base are identified and then the main components that were needed are presented.

## 2.1 Use Cases of the Knowledge Base and the Semantic Registry

In order to identify the main Use Cases that the DBE Knowledge Base (KB) is going to serve, we must define the actors that are expected to exploit the DBE KB. There are three main actors that utilise the KB in the DBE environment:

The *DBE Service Consumer SME*: An SME that consumes Services provided by other SMEs

The *DBE Service Provider SME*: An SME that provides Services consumed by other SMEs. We consider Software Producer SMEs as DBE Service Providers whose value proposition is the development of software solutions.

*The DBE Regional Catalyst*: Any Institution or Organisation that has the responsibility (along with other Regional Catalysts) to import/maintain/manage regions/business domains in the DBE.

The following section presents a set of Use Cases for these actors that are supported by the centralised first prototype implementation of the knowledge base.

### 2.1.1 Use Cases for DBE Service Providers in the Service Factory Environment

| U/C No | USE CASE TITLE | Main Success Scenario |
|---|---|---|
| **1.** | **Describe SME Structure and Function** | 1. The SME uses the BML Editor in order to model itself<br>2. The BML Editor retrieves the Business Ontologies required to guide the Business Modelling Process. The SME describes its function (e.g. demographic information, structural Organisation, legal status, infrastructure, Products, revenue models, cost models etc)<br>3. The BML Editor produces a BML description |
| **2.** | **Semantically describe a Service** | 1. The SME uses the BML Editor to semantically describe its services<br>2. The BML Editor Retrieves the Service Ontologies to guide the Service Description Process |

| | | |
|---|---|---|
| | | 3. The SME creates the Semantic Service Description |
| **3.** | **Store BML Description** | 1. The BML Editor produces a BML and SSL public description<br>2. The BML Editor looks up a KB service<br>3. The BML Editor stores the public description to the KB service |
| **4.** | **Search for Business models** | 1. The SME uses a tool with a GUI (Query Tool) to formulate a query (using QML) in order to search for existing models stored in the KB.<br>2. The tool sends the query to the KB service where it is processed, and gets the returned results<br>3. The SME views the returned models with the BML Editor. The SME is able to edit, alter or add new elements to the opened model and can store it back to the KB service as a new model |
| **5.** | **Describe a service** | 1. The SME uses a tool (SDL editor) to describe the offered service functionality from a technical point of view.<br>2. The SDL editor looks up a KB service<br>3. The SDL editor stores the SDL description in the KB. |
| **6.** | **Develop Service** | 1. The SME generates code from the SDL description using the SDL2JAVA tool.<br>2. The SME adds custom code to implement their service.<br>3. The SME uses the SM creator tool to construct the Service Manifest<br>4. The SM creator looks up a KB service<br>5. The SME browses the BML , SDL and BML model instances produced by it and selects one of each<br>6. The SM creator retrieves the selected parts from the KB and assembles them in one XML document called Service Manifest<br>7. The SM creator stores the Service Manifest locally<br>8. The SME generates the deployment file that contains the service code, the Service Manifest and other relevant information. |

### 2.1.2 Use Cases for DBE Regional Catalysts in the Service Factory Environment

| U/C No | USE CASE TITLE | MAIN SUCCESS SCENARIO |
|---|---|---|
| **1.** | **Insert New Business Ontology into DBE** | 1. The Regional Catalyst uses a tool in order to create a new Business Ontology<br>2. The tool retrieves the abstract Business Ontology in order to drive the Ontology Definition Process.<br>3. The Regional Catalyst customises/extends the abstract Business Ontology to define a new Business Ontology, guided by the tool<br>4. The tool interacts with the KB in order to permanently store the new Business Ontology in it. |
| **2.** | **Update Business Ontology** | 1. The Regional Catalyst uses a Tool in order to modify an existing Business Ontology<br>2. The tool retrieves the Business Ontology that is to be modified<br>3. The Regional Catalyst modifies the Business Ontology<br>4. The tool interacts with the KB in order to make the changes in the Business Ontology permanent |
| **3.** | **Insert New Service Ontology** | 1. The Regional Catalyst uses a tool in order to create a new Service Ontology<br>2. The tool retrieves the abstract Service Ontology in order to drive the Ontology Definition Process.<br>3. The Regional Catalyst customises/extends the abstract Service Ontology to define a new Service Ontology, guided by the tool<br>4. The tool interacts with the KB in order to permanently store the new Ontology in it. |
| **4.** | **Update Service Ontology** | 1. The Regional Catalyst uses a tool in order to modify an existing Service Ontology<br>2. The tool retrieves the Service Ontology that is to be modified<br>3. The Regional Catalyst modifies the Service Ontology<br>4. The tool interacts with the KB in order to make the changes in the Service Ontology permanent |

### 2.1.3 Use Cases for SMEs in the Service Execution Environment

| U/C No | USE CASE TITLE | MAIN SUCCESS SCENARIO |
|---|---|---|
| **1.** | **Deploy Service** | 1. The SME deploys the service in DBE servent (using a deployment file). The deployment module looks up a Semantic Registry Service<br>2. The deployment module publishes the Service Manifest to the Semantic Registry and is returned a unique identification tag<br>3. The deployment module registers the service proxy in the FADA network using the service manifest identification tag as the service identifier |
| **2.** | **Discover Services** | 1. The SME uses a Tool with a GUI (Query Formulator Tool) to discover/find services<br>2. The tool retrieves the Service and Business Ontologies from the KB in order to guide the Query Formulation Process<br>3. The SME iteratively formulates its query guided by the Service Discovery Tool<br>4. The tool looks up a SR service<br>5. The tool submits the query to the Semantic Registry<br>6. The Semantic Registry returns a ranked list of qualifying Service Manifests. |
| **3.** | **Execute Service** | 1. The SME selects a service manifest from the returned results and requests its execution<br>2. The tool looks up the FADA network to find proxies of alive running services using the identification tag of the selected service manifest<br>3. The SME downloads the proxy and invokes the UI creation or other methods, as appropriate. |

# 3. The DBE Knowledge Base

## 3.1 Overview

The DBE Knowledge Base will be deployed across the SME nodes that will comprise the digital business ecosystem. It is expected that SMEs will contribute resources to the DBE network taking over equivalent roles. One of the main objectives of designing the KB infrastructure was to make possible the installation of a basic KB infrastructure on every SME node that exports core KB functionality necessary to support all the DBE activities of the SME. Thus, technology decisions for the implementation of the KB infrastructure should take into account the important platform independence requirement and also enable the effective execution of KB services under limited resources.

Each SME node that contributes resources to the DBE network will have a local basic KB infrastructure that is called the KB peer (KBP).

KBP is based on a combination of a relational database management system and a MOF metadata repository system. All KBPs will form a P2P network and will cooperate to offer an effective way to discover, describe, share, analyse, and integrate the information of all enterprises.

The KB manages disparate data in models, describing the structure and relationships among disparate information sources.

Using models for Businesses and Services, the KB enables organisations to capture important knowledge. The models created and stored in the KB hold the details of the structure and relationships of each entity within the model. These details are available to all the DBE users (of any kind) through the core DBE tools: (Discovery Tool, BML Editor, SDL Editor, etc.)

The key features of the implementation of the DBE KB are the following:

- **Standards-Based:** The KB supports OMG standards for modelling and model interchange (standards for information organisation and interchange) (MOF 1.4[2], XMI 1.2[3]). The adoption of existing standards ensures compliance with industry directions and enables the sharing of models among various data modelling tools.
- **Extensible:** The KB extends easily to accommodate other information models and sources. There is no limit to the information sources that can be added or modelled
- **Support for Multiple Metamodels and Models:** The KB includes multiple metamodels for Ontologies, Businesses, Services and Compositions. Additional metamodels that are going to be used in the future can be easily implemented, supported and managed. The KB enables different business users to model the enterprise and the services they offer, as well as to use, integrate or modify existing models.
- **Information Views:** The KB can support different information views. In other words, different access levels to the same metadata from different perspectives.
- **Platform Independent:** Designed and implemented entirely for the Java Platform the KB is platform independent and can be deployed across SMEs regardless of existing operating systems. It is compliant with the Java Community Process

interface/platform standards (J2EE, JMI, JDBC, etc) [5], [6], [7] and the existing XML standards.

## 3.2 The MOF Metadata Architecture

According to the Object Management Group (OMG) [2], *"the central theme of the MOF approach to metadata management is extensibility. The aim is to provide a framework that supports any kind of metadata, and that allows new kinds to be added as required. In order to achieve this, the MOF has a layered metadata architecture that is based on the classical four layer metamodeling architecture popular within standardization bodies such as ISO and CDIF. The key feature of both the classical and MOF metadata architectures is the meta-metamodeling layer that ties together the metamodel and model"*. The traditional four layer metadata architecture introduces the following layers:

1. **The information layer.** It consists of the data that we wish to describe.
2. **The model layer.** It comprises the metadata that describe data in the information layer. Metadata is informally aggregated into models.
3. **The metamodel layer.** It consists of the descriptions (i.e., meta-metadata) that define the structure and semantics of metadata. Meta-metadata constructs are informally aggregated into metamodels. A metamodel is essentially an "abstract language" for describing different kinds of data.
4. **The meta-metamodel layer.** It consists of the description of the structure and semantics of meta-metadata. In other words, it is the "abstract language" for defining different kinds of metadata.

The example in Figure 1 is provided by OMG and exemplifies the classical four layer meta-modelling framework showing the organisation of metadata for some simple records along with the "RecordTypes" metamodel used to define the structure and the semantics of the metadata.

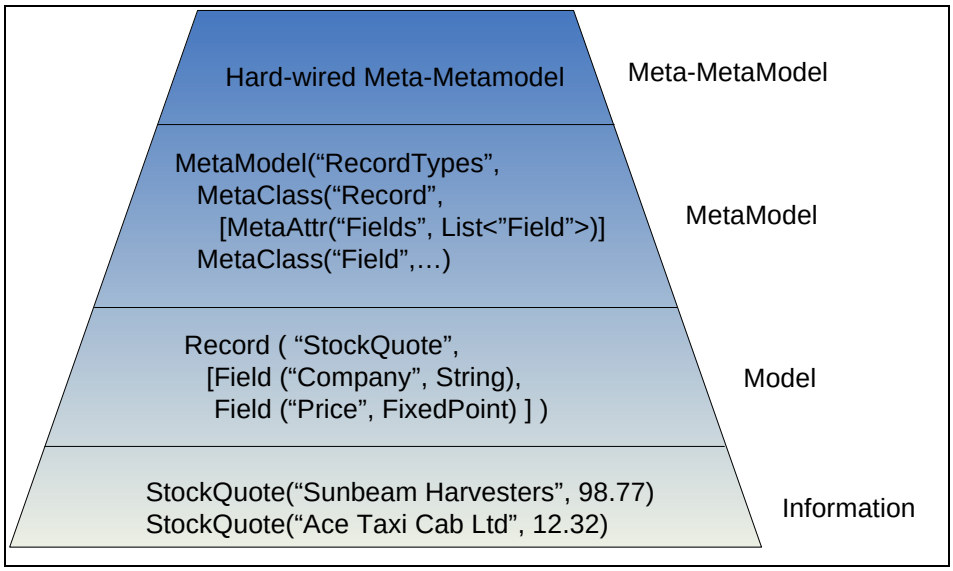


**Figure 1: The four-layer Metadata Architecture**

While the example shows only one model and one metamodel, the main aim of having four meta- layers is to support multiple models and metamodels.

The meta-modelling approach is a common way of organising models, a way that involves descriptions on levels placed on top of each other. The concepts of one level have corresponding descriptions on a next level (meta-level, level above). Stated differently, a level is a model and the level below is an instance of this model. In relation to object-oriented programming the lowest level contains the objects of a running system, while the classes reside on the next lowest level.

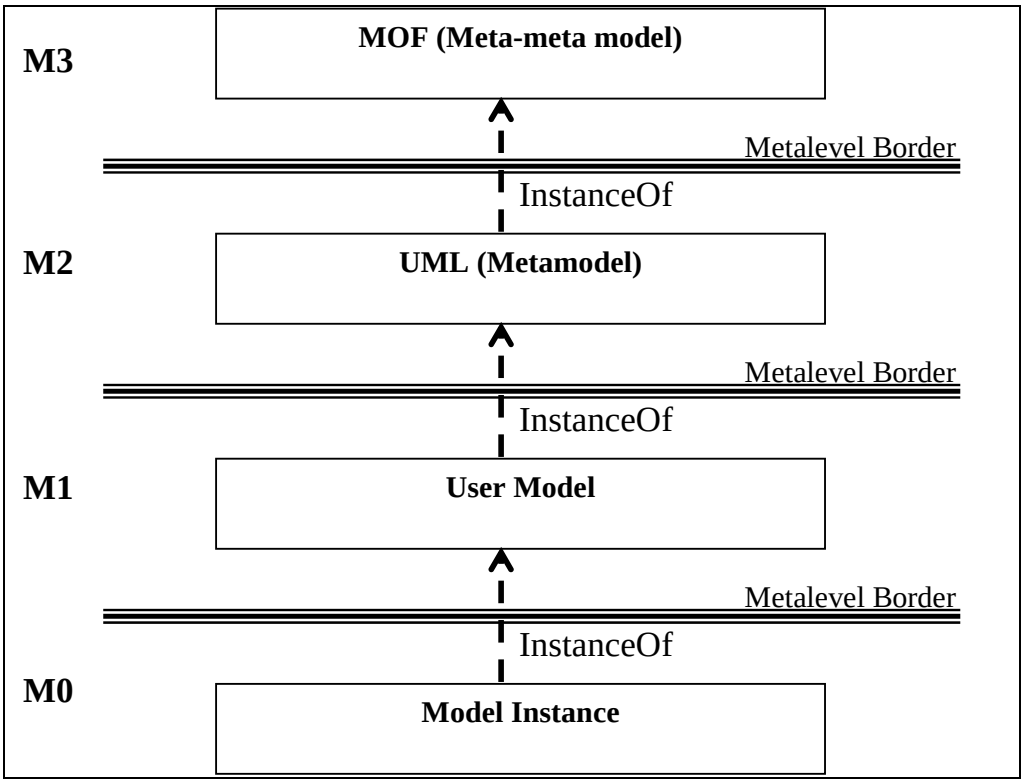


**Figure 2: The MOF 4-layer metamodel architecture**

The advocated architecture is based on strict meta-modelling which means that all elements on one level are instantiated from the level directly above. It seems natural to view the instantiation logic as operating with 3 levels, e.g. you have a description of what a class is, you have a class (e.g. class Person) and you have an object. The definition of the class concept must of course be done with the help of yet another level. Both MOF and UML offer support for object-oriented concepts, and the core parts of MOF and UML are structurally equivalent. Since MOF is used to define itself, the level above MOF (M4) can be seen as MOF once more; one can imagine an infinite number of MOF levels. MOF is defined by self referencing, "ending up with" class Class which is an instance of itself.

## 3.3 The KB peer Infrastructure

The KB infrastructure follows the OMG's MOF metadata approach. The KB is able to handle all the types of information within DBE (metadata and data) and extract useful knowledge. The KB manages MOF metamodels, models, and instances providing the full functionality of a MOF repository, and uses XMI documents for metadata and data interchange.

The KB also exports a variety of additional interfaces to enable the storing and management of all the information used inside the ecosystem (e.g. usage data).

From a technical point of view, the KB Architecture is based on a combination of a MOF-compliant repository and a data management system. The KB embeds the Netbeans Metadata repository (Netbeans-MDR) [8] and is exploiting as a persistency layer a light java relational database (Apache Derby Java Database) [10].

The KB infrastructure is able to fully support the currently available DBE metamodels Ontology Definition Metamodel (ODM) [16], Business Modelling Language (BML) [12], Semantic Service Language (SSL) [16], Service Description Language (SDL)[13] and Service Composition Metamodel (SCM) [14] and provides basic functionality for processing, storing and retrieving models following the above metamodels.

The KB is compliant with the JMI 1.0 specification [6] that is the result of a Java Community Process (JCP) effort to develop a standard Java API for metadata access and management. The advantages of compliance with JMI 1.0 are:

- the provision of a standard metadata management API for the Java 2 platform,
- the definition of a formal mapping from any OMG standard metamodel to Java interfaces,
- the support of advanced metadata services (such as reflection and dynamic programming) and
- the interoperability between tools that are based on MOF metamodels and are deployed in the DBE environment.

# 4. The KB Infrastructure and Common Components

## 4.1 The KB peer

The KB provides a common persistence and knowledge management layer in the digital ecosystem. The KB offers a set of APIs and JMI interfaces to the DBE components, modules and tools that need access to the DBE Knowledge (M1 Model information and M0 data). The components that comprise the basic KB peer infrastructure are the KB manager, the Model Manager, the Recommender, the Usage Manager, the Knowledge Access Module, the MDR manager and the DB manager.

The KB infrastructure consists of a set of common components that cooperate with each other to support the basic KB functionality. These components are shown in

**Figure 3** and a brief description can be found in the next section.

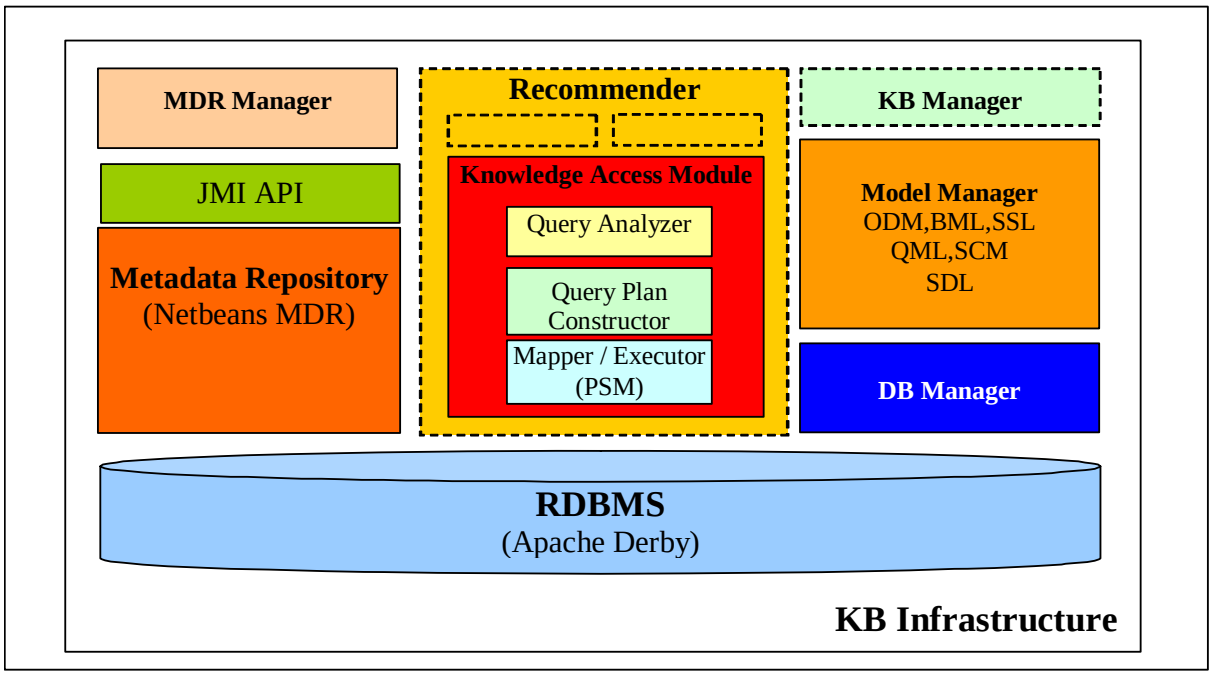


**Figure 3: The KB Peer Architecture**

## 4.2 The KB Components

The KB peer common components are:

- The KB manager takes over the management of the KBP (Knowledge Base Peer), which is typically located in an SME, and is responsible for the P2P cooperation across the KBPs.
- The Model Manager integrates and exploits the data management system in order to increase the performance and the efficiency for the manipulation of models and data. Efficient data management is based on the use of the indexing, optimisation

and query mechanisms of the underlying relational database management system. The Model Manager is responsible for processing and storing metadata into the data management system. The KB handles models that are described by XMI 1.2 documents and are compliant with the already imported MOF-compliant metamodels. The XMI documents are processed and are stored into the metadata repository. The Model Manager using the JMI interfaces of the metadata repository processes the metadata of the newly inserted DBE models, extracts information and organises this information into the database system. Currently the Model manager is capable of processing models that comply to the following metamodels: the ODM (Ontology Definition Metamodel), SSL (Semantic Service Metamodel), SDL (Service Description Metamodel), SCM (Service Composition - BPEL), QML (Query Metamodel)

- The Recommender Module implements the components related to the evaluation of both candidate services and candidate business partners, in the process of creating composite services and establishing partnerships, respectively. In particular the Recommender Service acts as an autonomous process that manages SME preferences (either business preferences or service preferences) and matches theses preferences with available business descriptions and service descriptions. The Recommender Service is supported by the Recommender Module that is responsible for matching preferences with business descriptions and service descriptions. The major assumption behind the design and implementation of the Recommendation mechanisms is the existence of powerful business and service ontologies that capture the semantics of business models and service descriptions. These ontologies are used to define the corresponding preferences for businesses and services. The recommender exploits the relational database system to store preferences and all recommendation mechanisms operate on top of the relational database to implement the necessary matching functions between preferences and business/service descriptions. In the current implementation basic recommendation mechanisms are being tested based on primitive preference structures since the SME preferences structures are not yet defined. The Recommender Module also includes the Knowledge Access Module and other modules that will be developed in the future.
- The Knowledge Access Module [15] takes over the processing of queries submitted to the KB. A powerful Query Metamodel Language (QML) has been developed, capable of expressing metamodel-driven queries on models and data. Instances of the Query Metamodel (query models) represent a query request in the KB. The Query Metamodel is based on OCL 2.0 and is appropriately adapted to MOF 1.4. Using this metamodel one can formulate metamodel independent queries utilising the JMI reflective functionality. These queries will be streamed to the server (encoded as XMI 1.2 documents), and are stored in the metadata repository. The Knowledge Access Module is composed of the Query Analyser, the Query Plan Constructor and the Query Mapper/Executor. The Query Analyzer interprets and analyses the query model using the JMI interfaces of the Metadata Repository. The Query Plan Constructor evaluates the query and constructs a valid query syntax tree. The Mapper/Executor module accepts as input the query syntax tree and, based on platform specific mapping rules, executes the query on the data management system.
- The MDR manager provides functionality for managing the Metadata Repository [8] and finally,

- The DB manager administers the DBMS and exports database specific functionality that can be used by other modules in the KB.

## 4.3 Organising Information

As described previously, the metadata repository offers the functionality of processing XMI documents, and exporting, in XMI documents, pre-selected content that is stored in the repository. The Model Manager bridges the repository and the data management system and is responsible for appropriately populating either the database, when a storage request is issued, or the metadata repository, when a result of a query needs to be exported.

The following picture illustrates a sample of the relational model of the SSL that has been imported into the database and is used by the Model manager to organise the information that is extracted from the processing of the stored semantic service description models. Appropriate database schemas have been developed to efficiently handle BML, ODM, SDL and SCM models. The organisation of the information in the database enables the exploitation of the efficient querying mechanisms provided by the database in favour of the information discovery and recommendation tasks of the KB.

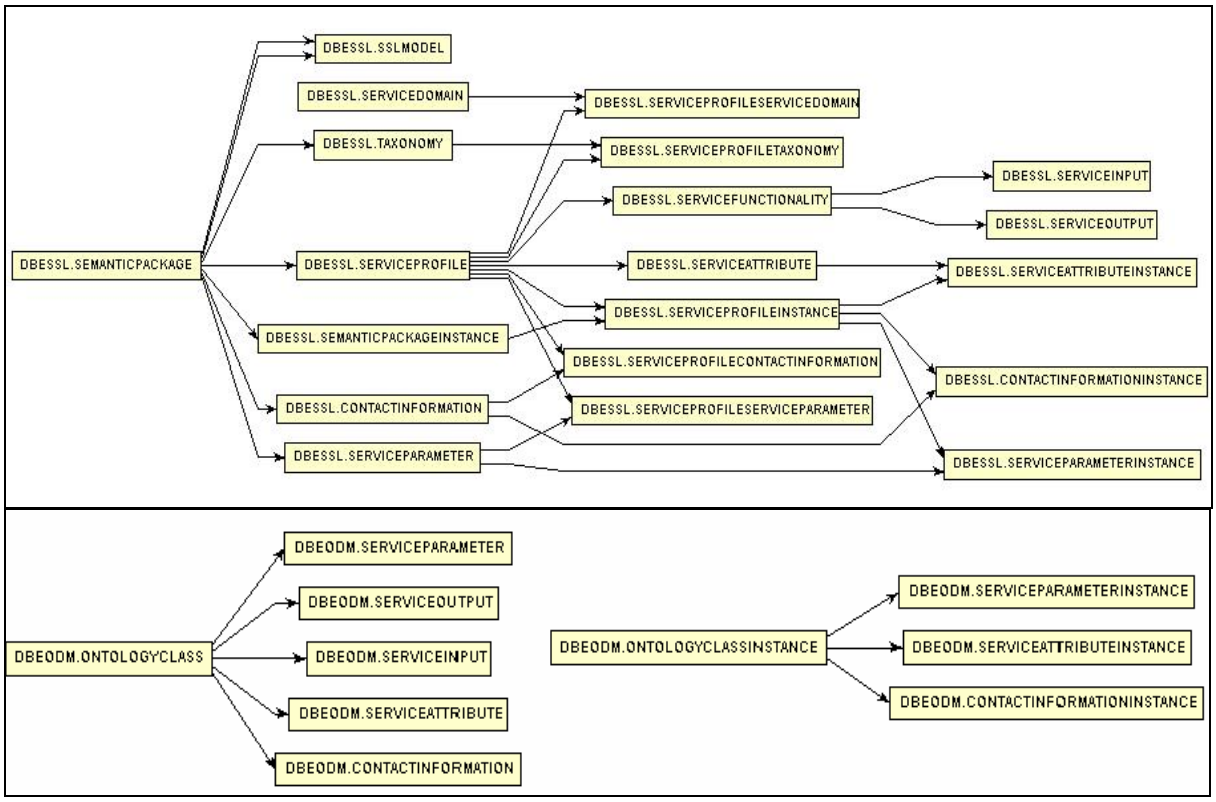


**Figure 4: The RDB Schema for managing SSL models and data**

## 4.4 Handling Model Instances

In MDA development the source of a great deal of confusion is the location of information within the OMG four-level MOF meta-data architecture. As illustrated in Figure 1, “data” has traditionally been hosted in the bottom “M0” layer of the four-level hierarchy. However, in

the latest (UML 2.0) [11] interpretation of this infrastructure, the bottom level is now inhabited by "real world" objects while models of these objects (so called instance specifications) occupy the M1 level alongside classes. This solves several problems, leading to a situation where the M1 level of the infrastructure contains both type and instance information (see Figure 5). This fact is not only present in the UML 2.0 but also in other domains. For example in the ontology community the vast majority of ontologies are constructed according to the simple "two level" framework. Concepts in the domain of discourse are viewed as types or "universals", while specific "knowledge" is viewed as a set of instances or "individuals".

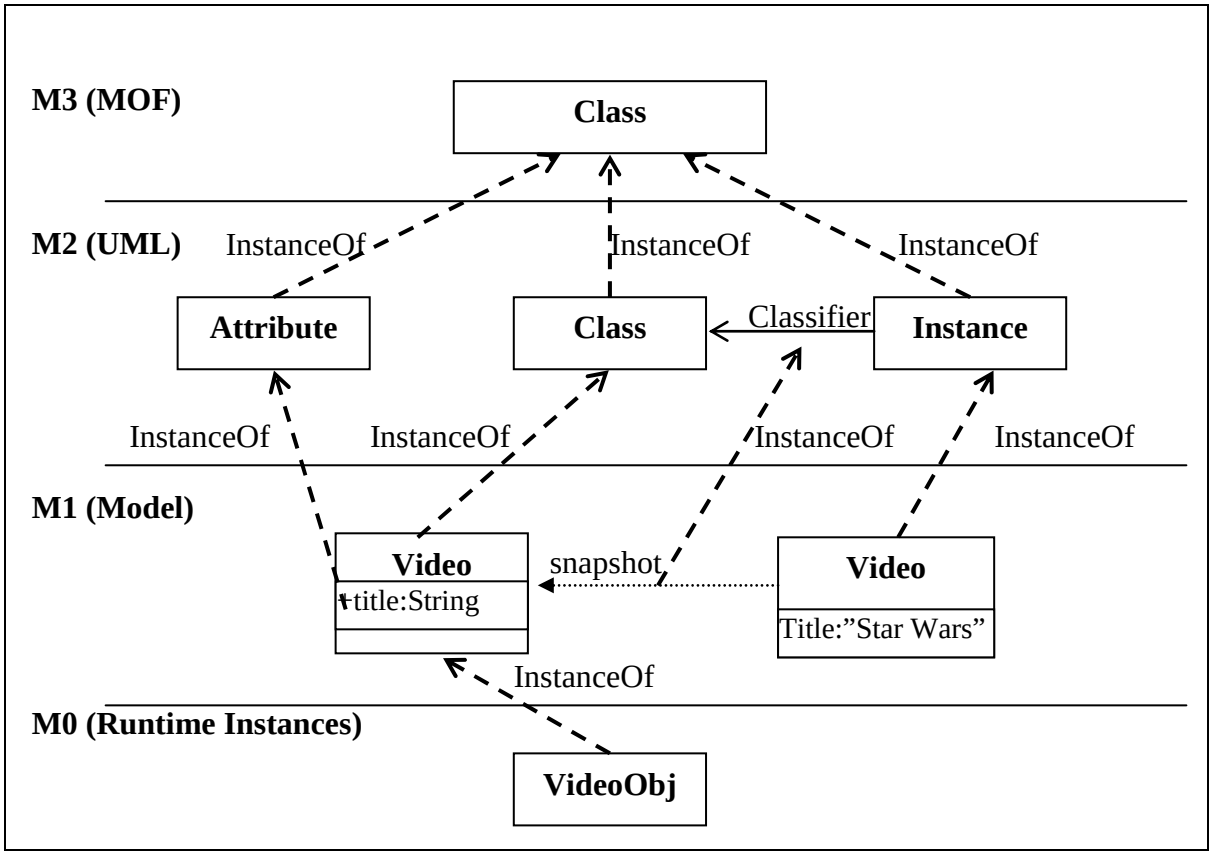


**Figure 5: The MOF Meta-Data Architecture**

Since a great number of BML models are expected to be stored in the DBE KB, a feasible and uniform approach is needed to handle the data of these models. One technically complex approach is to support XML [4] documents (for interchanging data) that conform to multiple different XML schemata.

A more uniform approach for the KB and SR services was followed, using the XMI1.2 specification [3] for describing both models and their instances. XMI has been developed in response to an OMG requirement for a stream-based Model Interchange Format. However, XMI could be also used for other kinds of structured data interchange. Thus, in the DBE environment, where the transmission of both models (schemata of data and constraints) and data (instance values) is required, the use of XMI as a uniform exchanging mechanism is valuable.

The ability of the Service Registry to accept both models and data encoded as XMI elements inside the Service Manifest XML container is of high importance. Moreover, this approach reduces the complexity of the system and makes its maintenance easier since the existing

persistency infrastructure (KB infrastructure) is exploited. Moreover any repository or tool that can encode and decode XMI streams can exchange structured information with other repositories or tools with XMI capability. This capability could be based on a mapping between particular MOF models and an internal model. One of the advantages of this approach is that the DBE tools can use the KB proxy functionality to generate specific JMI interfaces to be used for the creation of model metadata and model instance data.

Based on the above, an instantiation MOF model (M2 level) has been developed that acts as a model for describing instance specifications at M1 level (following the approach of UML 2.0) and is used in the current implementation to support the descriptions of instances of the SSL models. These model instances are being streamed to the KB service as XMI1.2 documents in order to be processed in the KB and finally organised in the database.

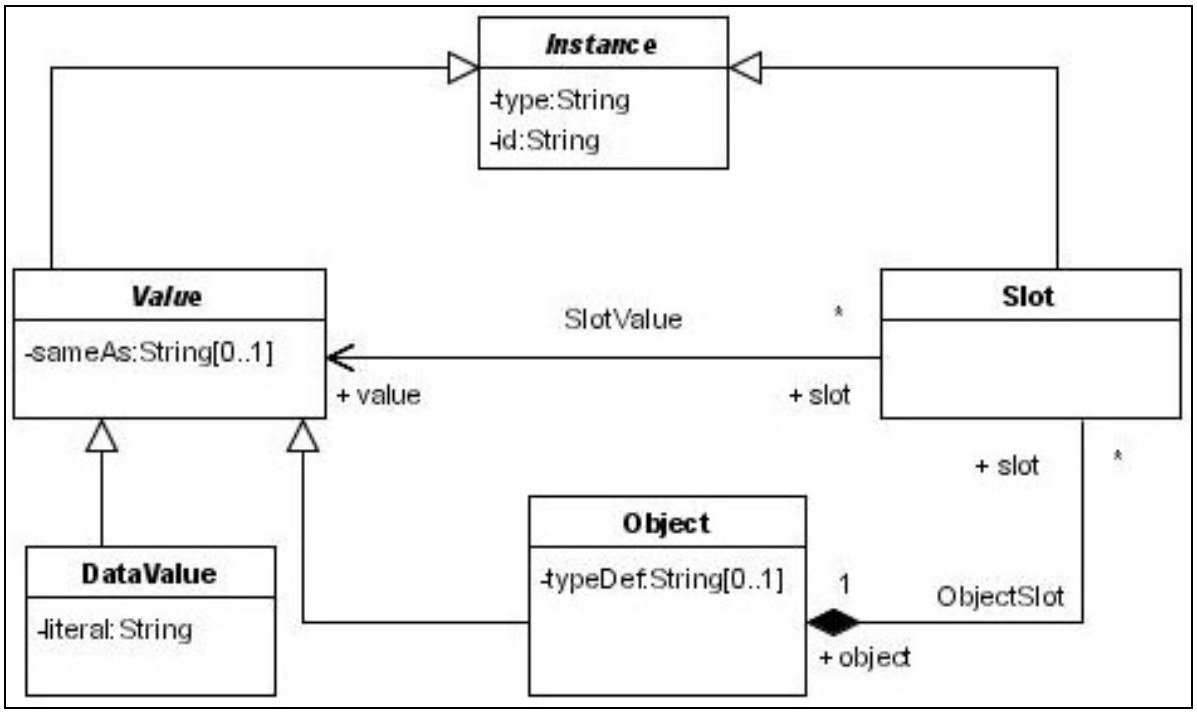


**Figure 6: The Instantiation Metamodel**

A similar approach will be followed also for model data of other metamodels in the case where M0 data exist for models. The current approach and implementation also permits the packaging of both M1 models and the corresponding M0 data as an integrated XMI document enabling the direct validation of the data against the M1 models.

# 5. The KB and SR Services

On top of the KB basic infrastructure the DBE core services related to DBE Knowledge Base (the KB Service, the Semantic Registry Service and the Recommender) can be deployed with the support of some additional middleware components that may be needed (Usage Manager, SM toolkit, SM manager etc.).

It is possible during the installation phase to configure the system so as to enable only the features that are necessary to operate the Semantic Registry Service or the KB Service respectively. These DBE core services export the functionality that can be supported by the underlying infrastructure components (e.g. KB, Model, Usage, DB, MDR managers, Knowledge Access Module, etc.).

As far as model information is concerned the KB and SR services act as JMI-enabled metadata services. The KB is the central metadata store for the DBE system. The KB and SR services are capable of providing a complete implementation of the JMI specification.

An implementation of the JMI specification might, for example, consist of a server process that realises the JMI interfaces corresponding to some MOF-compliant metamodel, the JMI reflective interfaces, and the XMI reader and writer interfaces. Precisely how a JMI-enabled metadata service is implemented is not prescribed by the JMI specification itself.

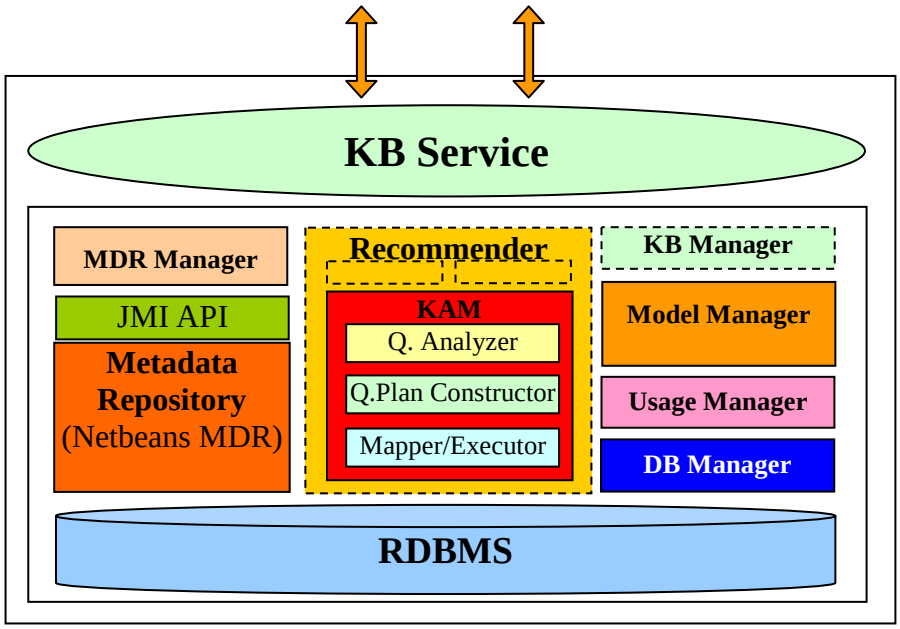


**Figure 7: A DBE Knowledge Base Service**

In **Figure 7** the deployment of a KB service in the DBE environment is shown. The KB service is supported by the basic KB infrastructure and is able to accept knowledge access requests from multiple DBE components and tools. The KB Service exports functionality for storing, retrieving and searching models that comply with the supported MOF metamodels. Such functionality is vital in the Service Factory environment where the SMEs will need to model themselves by discovering and altering existing models or creating and sharing new models that represent specialised business knowledge features. The KB service accepts, through network connections, only XMI 1.2 documents that represent DBE models. These documents are streamed to the MDR, processed by the model manager and, finally, stored in the database.

The Usage Manager is an additional component that is deployed with the KB service and handles the service usage data of DBE services. It also exports APIs (see Appendix 10.7) to be used by the service usage interceptors for storing and organising data into the database. This data may be used in the future to automatically create new SME profiles, or adapt existing SME profiles to improve system recommendations

Since the FADA proxy framework has been adopted in the DBE, the proxies that communicate with the KB and SR service should export the JMI interfaces in cooperation with the service as will be described in the following section.

**Figure 8** presents the deployment of a Semantic Registry Service. The SR service is used in the Service Execution environment. SMEs that wish to advertise their services publish, to the SR service, information about the enterprise and the services offered. Appropriate business models, semantic service models, data instances, service interface description models and other information is bundled together in an XML document to construct the Manifest of an SME service offering.

The Service Manifest (SM) is published in the SR service where it is processed, and the information extracted is properly organised in the database enabling effective semantic discovery tasks. The SR service contains two additional components the SM toolkit and the SM manager. The SM toolkit provides functionality for handling the SM XML documents (see Appendix 10.9). This component exports APIs for creating, updating and extracting particular information elements from the SM documents. The SM toolkit is used by the SM manager to disassemble an incoming SM document into its constituent parts. As mentioned above these parts (among optional others) consist of the BML, SSL, SDL models as well as the BML and SSL instances, and are represented as autonomous XMI documents within the original service manifest. These XMI documents are stored in the metadata repository and processed by the Model manager and organised in the database as has been described in the KB infrastructure section.

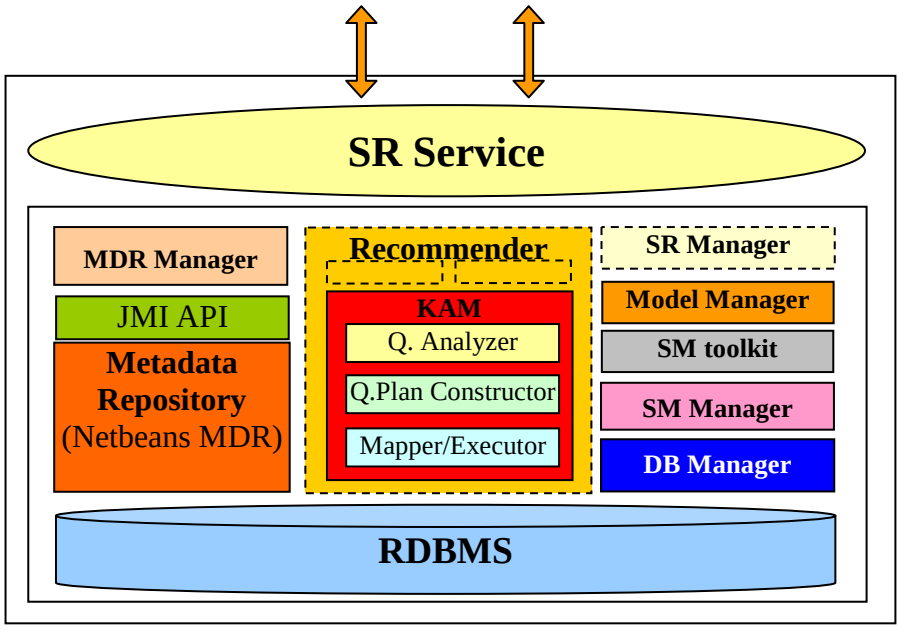


**Figure 8: A DBE Semantic Registry Service**

The SR service exploits the Recommender's Knowledge Access Module to respond to advanced knowledge discovery requests that are expressed as QML queries.

# 6. The KB and SR Proxies

The KB and the SR services provide proxy objects that take over the communication between a DBE tool/module and the service. These objects (KB proxy, SR proxy) are powerful and intelligent proxies. In addition to the communication tasks, they are also capable of performing processing and caching tasks. This is necessary to reduce the communication cost between the service and the proxy, and to improve the response times of the proxy method calls at the client side. Also, the caching at the proxy enables the continuation of execution of a task at the client side in case of a temporary failure of the KB peer. The task can be completed and synchronised with the KB when the KB peer becomes operational again or (if that is not possible) when another KB peer can be found.

## 6.1 The KB proxy

The KB proxy provides functionality for storing and retrieving models. The KB proxy has a dual behaviour. It can act as a small and flexible proxy that takes over the information interchange with the KB service or it can become a powerful *JMI-enabled* tool that exports advanced functionality.

During the proxy's initialisation phase (after the FADA lookup and download phase) the KB proxy tries to detect if MDR libraries are present in the execution classpath. If MDR support is available the KB proxy becomes a JMI-enabled toolkit. In this case the KB proxy exploits the memory implementation feature provided by MDR to deliver local caching features. In addition the MDR-supported proxy is able to provide JMI reflective functionality or alternatively metamodel specific JMI code generation functionality that can be used by the DBE tools to generate comprehensive interfaces in order to create new models or access/browse cached models that are fetched from the KB.

The KB proxy communicates with the KB service by exchanging XMI 1.2 documents.

Special attention during the proxy design and implementation process was paid in order to keep both the proxy size and the codebase size of the proxy very low to minimise lookup and downloading delays for the KB proxy (the codebase size of the current implementation is approximately 20 Kbytes).

If MDR support is absent in the execution environment of the KB proxy the previously mentioned advanced features are disabled and the KB proxy can be used as a simple communication server proxy. This behaviour is desired and applicable in cases where a DBE application is autonomous and uses alternative software (apart from MDR) for handling XMI 1.2 documents.

At the start up phase of a KB service, a new proxy is created and registered into the FADA network. The proxy contains all the information that is necessary to contact the service as well as the remote location of the codebase for additional class loading. The registration of the KB proxy is performed using special entries in the FADA directory unique for each KB service instance. Having the proxies registered in the FADA network, a DBE module or tool, that needs access and support by a KB peer, can lookup in FADA and choose a KB proxy to access the most appropriate KB peer.

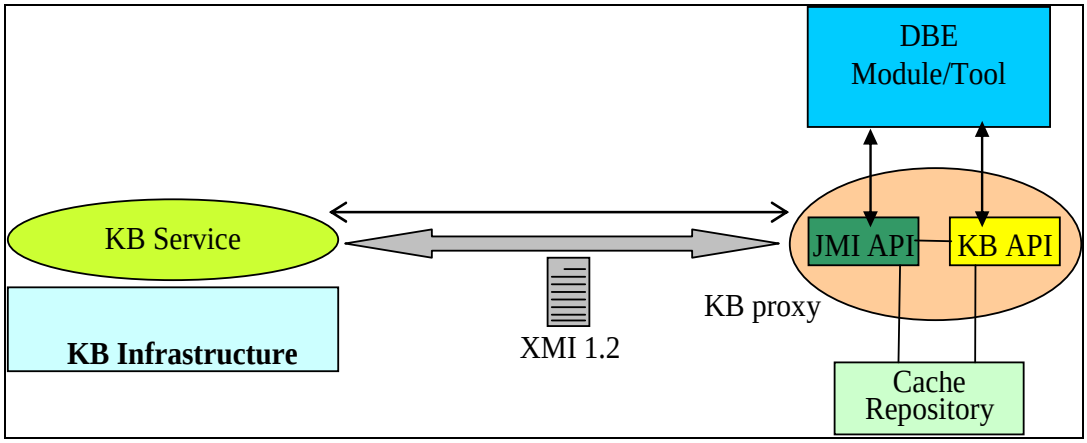


**Figure 9: KB Service – KB Proxy interaction**

The KB proxy interface currently provides methods which return objects (see Appendix 10.1) that implement specific interfaces for Ontology management, Semantic service management (part of BML), Service description management, Service Composition description management, Service Manifest management, Query Formulation and Service Usage data management (see Appendices 10.2,10.3,10.4,10.5,10.6). In the case where MDR support is enabled, an XMI document (ODM, BML, SSL, SDL, etc.) is imported from the KB in the initialisation phase of the proxy objects. This XMI document is interpreted as an instance of the MOF Model and is subsequently used in generating the internal structure of the outermost package extent inside the cache repository. After the initialisation phase the proxy is able to accept XMI documents that describe models (i.e. SDL model) corresponding to the respective MOF model, whose structure had already been built within the cache repository during initialisation phase. These documents are treated as metadata and are used to populate the cache repository.

The KB service exploits the Recommender's Query Engine to process the query, and returns a collection of qualified results (models) back to the KB proxy.

Each proxy object that is responsible for the management of instances of specific metamodels, exports JMI reflective functionality for the creation and browsing of models in the cache repository, and interfaces for storing and retrieving models into/from the KB. In addition, the proxy objects provide methods for generating metamodel specific code that can be used to access the cache repository. DBE applications that use the KB proxy are able to browse the imported models or make changes to them, or alternatively, create new models from scratch by using the metamodel – specific or reflective JMI interfaces provided by the proxy.

The KB proxies are "hard-wired" to understand MOF (when MDR is available). If they import an instance of the MOF Model, they subsequently build the package structure prescribed by the MOF Model instance. If, on the other hand, they import an instance of a metamodel (i.e., BML model) corresponding to some MOF-compliant metamodel whose structure had already been built within the cache repository, then it will treat the XMI document content as metadata and use it to populate the cache repository.

## 6.2 The SR proxy

The SR proxy has been developed in a similar way to the KB proxy. The SR proxy is used to communicate with a DBE Semantic Registry Service. The functionality of the SR service is different to the one exported by the KB service. The Semantic registry accepts service

manifest (SM) publication requests and offers a generic query mechanism for discovering published service manifests based on model and/or data characteristics (see Appendix 10.8).

As in the case of the KB proxy, the SR proxy and codebase sizes are kept very low in order to achieve fast lookup and download responses (the codebase size of the current implementation is approximately 16 Kbytes).

Since it is necessary for the semantic registry to understand the DBE models and be able to process the model data and provide advanced discovery functionality, it was developed on top of, and is supported by, the basic KB infrastructure that combines the metadata repository with a light database system.

The Service Manifest, as defined in Soluta's draft Service Manifest Software model document [18], is used to represent the published service. It contains all the specifications that describe a real service from the computing and business point of view. The SM data structure is shown in the following figure.

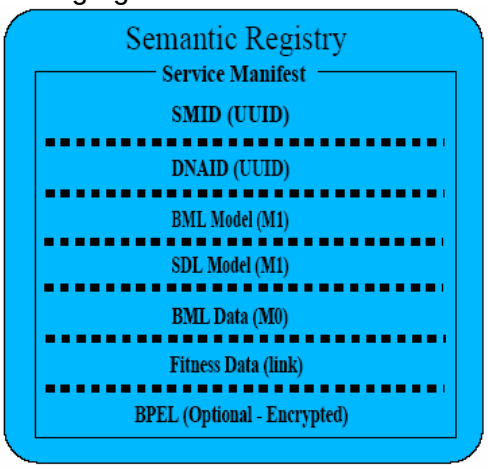


**Figure 10: The Service Manifest Structure**

Both the models and the data of a service are contained in the SM. The SM offers a mechanism to package together BML, SDL, SCM (BPEL) models and BML (currently SSL part) data. In the current implementation the SM is developed to contain the SMID, DNAID, and the bmlModel, the sdlModel, the bpelModel XML elements that carry the string representations in XMI1.2 of the SSL, SDL, SCM models respectively.

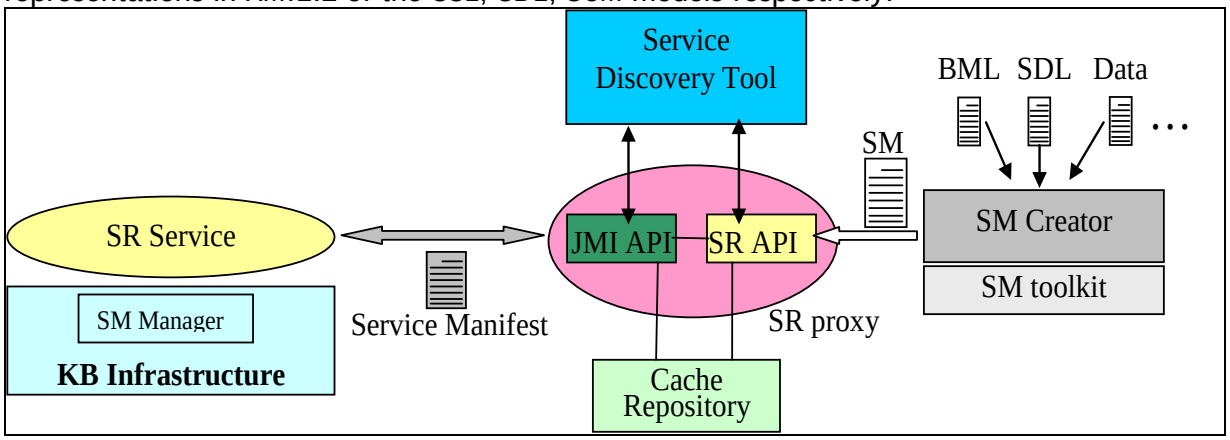


**Figure 11: SR Service – SR Proxy interaction**

During the publish procedure the SR proxy streams the SM (which has been previously created by pulling models and data from the KB) to the SR service. The SR service accepts and processes the SM document and extracts the models and data. The extracted information in XMI format is forwarded for further processing to the Model Manager as described earlier in this document and is finally organised into the database.

The SR service uses the query mechanism developed for the basic KB infrastructure in the same way as the KB service does. The SR proxy streams query models in XMI1.2 format to the SR Service. The SR service exploits the Recommender's Knowledge Access Module to process the query and returns a collection of qualified results (SMs) back to the SR proxy.

## 6.3 Use of KB and SR proxies

This section describes the way that DBE tools and other components can use the KB proxy to access and store information in the KB (in the DBE Service Factory environment), as well as the SR proxy for publishing and discovering SMs at runtime (in the DBE Service Execution Environment).

Figure 12 graphically illustrates the interaction among the KB and SR service and the DBE tools. The BML editor is a visual modelling tool and is used to construct business and service models (instances of BML metamodel). The modelling tool looks up the FADA network to find an appropriate KB proxy. In this example we assume that MDR support is available on the BML Editor side. Downloading and using the proxy, it connects to the JMI-compliant KB service and then acquires the SSL metamodel. From this point forward, the modelling tool directs its activities against local (clientside) JMI interfaces that are specific to the BML metamodels (currently only SSL is supported). The user (business analyst) might, for example, browse on the outermost package to discover the metamodel-specific packages clustered in this package. These would consist of the Core, ServiceBehaviour, Associations and Types packages.

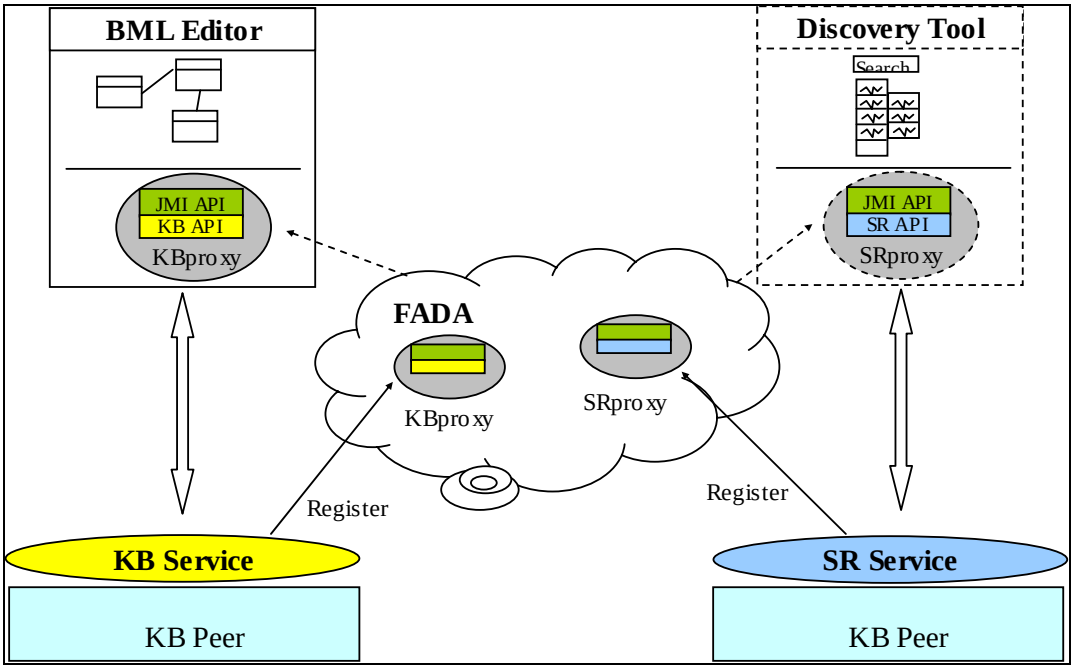


**Figure 12: KB and SR FADA proxies**

The user constructs an SSL model by selecting various classes from each of the aforementioned packages and creating instances of them.

For example, the user might create a Service Profile for Hotel Reservation with attributes “service name” and “provider” with type “String” and functionality “checkAvailability”. Internally, the modelling tool performs local calls against the SSL metamodel-specific JMI interfaces. The following code fragment illustrates several JMI calls that would be performed in the construction of a SSL model. While the modelling tool is calling the JMI methods the representation of the SSL model is being created inside the proxy’s cache. When the method KBstoreSSLmodel() is executed an XMI 1.2 document is generated that represents the new SSL model and is streamed to the KB service for permanent storage and further processing. The XMI document that is generated is presented below:

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<XMI xmi.version = '1.2' timestamp = 'Tue Nov 29 11:52:45 EEST 2004'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
  </XMI.header>
  <XMI.content>
    <SSL.Core.SemanticPackage xmi.id = 'a1' name = 'SemanticPackage' id =
'SSL://www.music.tuc.gr/semp12309'
      Documentation = 'Test Model'>
      <SSL.Core.NameSpace.OwnedElement>
        <SSL.Core.ServiceProfile xmi.idref = 'a2'/>
        <SSL.Core.ServiceAttribute xmi.idref = 'a3'/>
        <SSL.Core.ServiceAttribute xmi.idref = 'a4'/>
        <SSL.ServiceBehavior.ServiceFunctionality xmi.idref = 'a5'/>
      </SSL.Core.NameSpace.OwnedElement>
    </SSL.Core.SemanticPackage>
    <SSL.Core.ServiceProfile xmi.id = 'a2' name = 'sProfile' id = 's001' Documentation = 'This is a test Service
Profile'>
      <SSL.Core.ServiceProfile.Functionality>
        <SSL.ServiceBehavior.ServiceFunctionality xmi.id = 'a5' name = 'checkAvailability'
          id = 's004' Documentation = ''/>
      </SSL.Core.ServiceProfile.Functionality>
      <SSL.Core.ServiceProfile.Attribute>
        <SSL.Core.ServiceAttribute xmi.id = 'a3' name = 'ServiceName' id = 's002'
          Documentation = ''>
          <SSL.Core.ServiceAttribute.AttributeType>
            <SSL.Types.DataTypeURI xmi.id = 'a6' lexicalform = 'http://www.w3c.org/XML_Schema#String'/>
          </SSL.Core.ServiceAttribute.AttributeType>
        </SSL.Core.ServiceAttribute>
        <SSL.Core.ServiceAttribute xmi.id = 'a4' name = 'Provider' id = 's003' Documentation = ''>
          <SSL.Core.ServiceAttribute.AttributeType>
            <SSL.Types.DataTypeURI xmi.id = 'a7' lexicalform = 'http://www.w3c.org/XML_Schema#String'/>
          </SSL.Core.ServiceAttribute.AttributeType>
        </SSL.Core.ServiceAttribute>
      </SSL.Core.ServiceProfile.Attribute>
    </SSL.Core.ServiceProfile>
  </XMI.content>
</XMI>
```

**XMI 1.2 document representing the test SSL model generated using KB proxy's JMI APIs**

The following table depicts a code sample with the proxy API calls that the modelling tool would use in order to construct the SSL model described in the previous example.

```
import org.dbe.kb.metamodel.ssl.*;
import org.dbe.kb.metamodel.ssl.core.*;
import org.dbe.kb.metamodel.ssl.servicebehaviour.*;
import org.dbe.kb.metamodel.ssl.Types.*;
...
...
KBI kbi;
SSI sslP = kbi.getSemanticServiceProxy();
...
sslProxy.newSSLmodel("SSL://www.music.tuc.gr/semp12309","A test model");
SslPackage sslp = sslProxy.getSSLmodelPackage();
ServiceProfile sp = sslp.getCore().getServiceProfile().createServiceProfile("sProfile","s001","This is a test Service Profile");
DataTypeUri tString = sslp.getTypes().getDataTypeUri().createDataTypeUri("http://www.w3c.org/XML_Schema#String");
ServiceAttribute sa1 = sslp.getCore().getServiceAttribute().createServiceAttribute("ServiceName","s002","",tString,null);
ServiceAttribute sa2 = sslp.getCore().getServiceAttribute().createServiceAttribute("Provider","s003","",tString,null);
ServiceFunctionality sf = sslp.getServiceBehavior().getServiceFunctionality().createServiceFunctionality("checkAvailability","s004","");
sslp.getCore().getOwnsAttributeAs().add(sp,sa1);
sslp.getCore().getOwnsAttributeAs().add(sp,sa2);
sslp.getServiceBehavior().getOwnsFunctionAs().add(sp,sf);
try {
    sslProxy.KBstoreSSLmodel();
}
 catch (Exception ex) {}
```

**Example of creating a SSL model using KB proxy's JMI APIs**

The currently available functionality of the KB and SR proxies is briefly listed in the following table:

| **DBE Tools can lookup and use the KB proxy to :** | **DBE Tools can lookup and use the SR proxy to :** |
|---|---|
| 1 Create Models (Ontologies- ODM, SSL models, SDL models, SCM models) | 1 Publish an SM |
| 2 Browse models | 2 Delete an SM |
| 3 Store models in the KB | 3 Get SSL, SDL SCM models from a selected SM |
| 4 Retrieve models from the KB | 4 Get data from a selected SM |
| 5 Search for models in the KB (Using QML) | 5 Browse models |
| 6 Import/Export XMI1.2 documents | 6 Search/Discover an SM in the SR (using QML) |
| 7 Store Usage Data in the KB | 7 Import/Export Service Manifests |

# 7. Deployment of KB and SR services

The following figure shows the detailed deployment diagram of a KB or SR service. Two different nodes are represented in the diagram: the SME node that is hosting the KB or SR service and the SME node that needs access and support from the KB/SR services. Inside the nodes all the main software components are presented along with their dependencies and the exported interfaces.

During installation phase it is possible to determine whether the node is going to run a KB service offering support in the Service Factory Environment or a SR service thus enabling the capturing of knowledge and semantic discovery in the Service Execution Environment at DBE runtime.

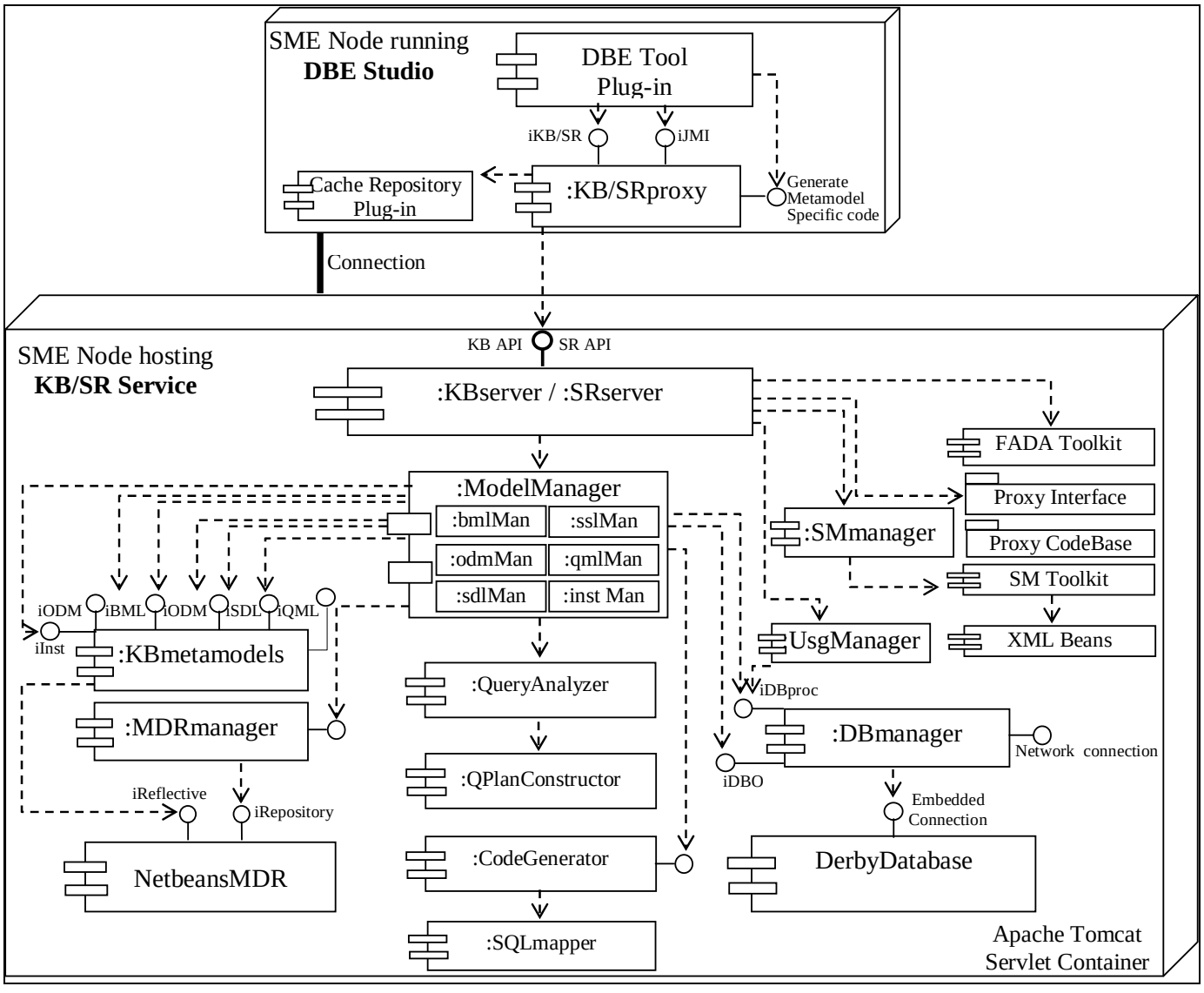


**Figure 13: The KB/ SR service deployment diagram**

Currently, the KB and SR installation procedure in a SME peer is an easy task. The KB system is bundled in a web archive file (.war) that can be deployed to any J2EE compliant application server or servlet engine (e.g. Apache Tomcat).

The KB web archive contains the relational java database, configuration files, metamodel definition files, database files, and all the software necessary to run the components of the basic KB infrastructure and the KB service.

When the KB peer server starts, it tries to register all the proxies on a FADA node. The kind of proxies a specific KB peer must register, the services to enable, and the addresses of FADA nodes that can be used for registration/lookup are all available, for the current implementation, in configuration files.

After deploying the KB peer server we can test the status by accessing the page http://installationMachine:port/KB/server. If there are no errors, a page like the one shown in the following figure appears.

From that page one can see more details about the status of the specific KB peer and also browse or download the available proxy interfaces (KB proxy) or a light version of MDR if MDR functionality support is needed on the client side.

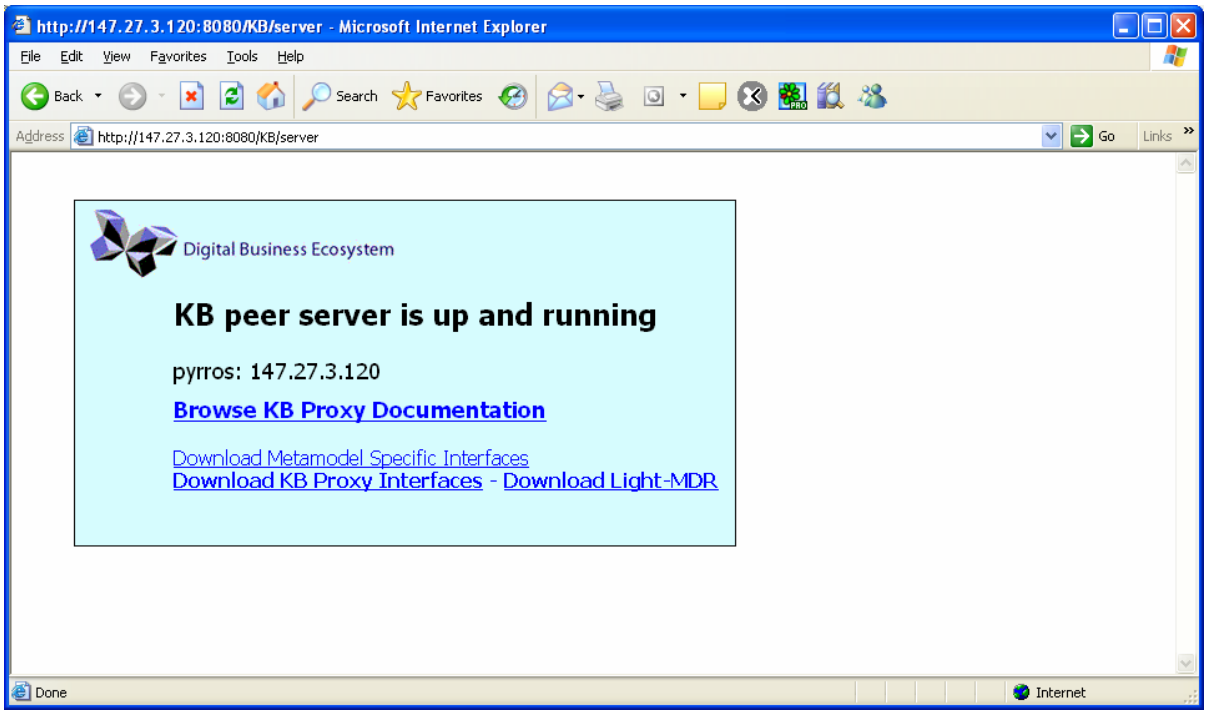


**Figure 14: The Start up page of a KB peer server**

The installation of a Semantic Registry service is the same as for the KB service. The SR web archive contains the relational java database, configuration files, metamodel definition files, database files, and all the software necessary to run the components of the basic KB infrastructure and the Semantic Registry Service.

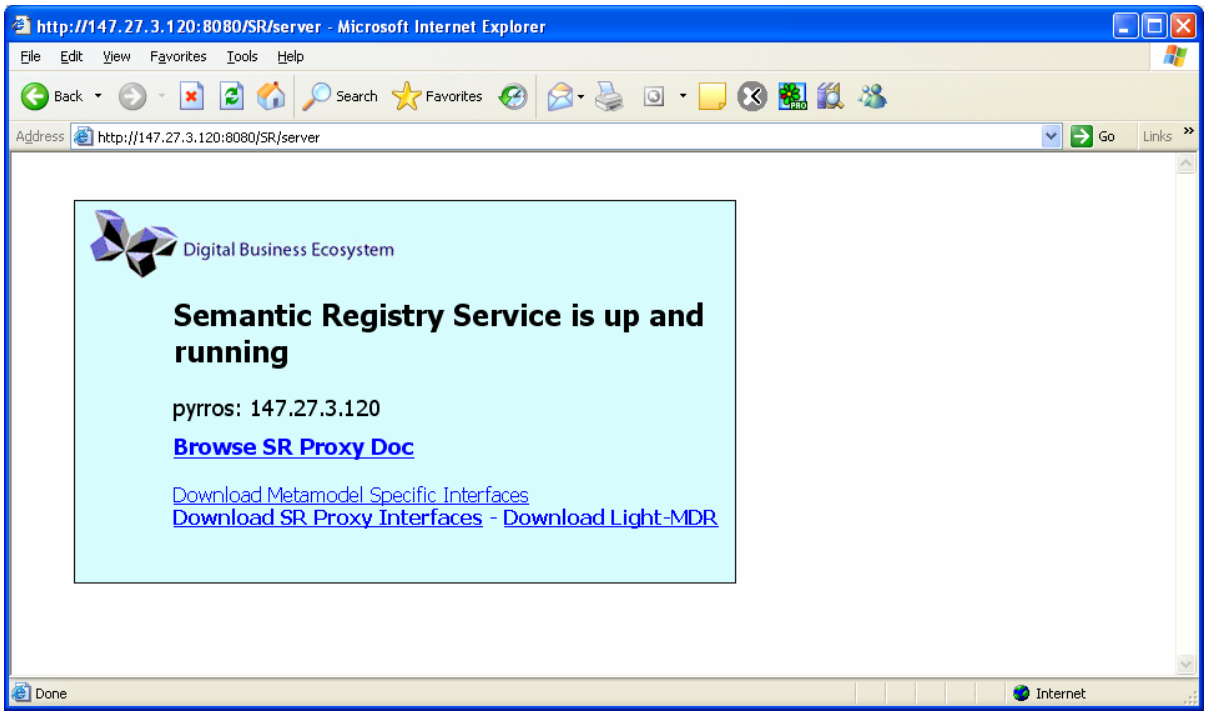


**Figure 15: The start-up page of a Semantic Registry Service**

# 8. 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 |
| DBMS | Database Management System: A software system that allows efficient manipulation (storage, organisation, indexing, and querying) of large amounts of data. |
| ExE | Execution Environment: It is where services live, where they are registered, deployed, searched, retrieved and consumed. This word is sometimes referred to as the "runtime of the DBE". |
| IR | Information Retrieval: Technology for retrieving personalised information from large collections of unstructured, semi-structured, or structured data. |
| JCP | Java Community Process: The "home" of the international developer community whose charter it is to develop and evolve Java technology specifications, reference implementations, and technology compatibility kits |
| JDBC | Java Data Base Connectivity: A technology that provides cross-DBMS connectivity to a wide range of relational databases, as well as access to other tabular data sources such as spreadsheets and flat files |
| JMI | Java Metadata Interface: A Java Community Process (see JCP description) specification of a standard Java API (see description of API) for metadata access and management based on the MOF specification. |
| KAM | See Knowledge Access Module |
| 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. |
| Knowledge Access Module | A component used to provide uniform access to the DBE Knowledge. |
| 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 (see JMI description) |
| 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 [1] |
| 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. |
| Query Analyzer | A component of the Knowledge Access Module that is used to analyse queries against the metamodel (used for knowledge representation) specific semantics. |
| Query Code Executor | A component used (by the Knowledge Access Module) to execute the generated query code |
| Query Code Generator | A component of the Knowledge Access Module that takes as input the query syntax tree and generates the code to be executed in the appropriate query language |
| Query Execution Plan Constructor | A component of the Knowledge Access Module that evaluates the QML expressions already analysed into a syntax tree representation. |
| Query Formulator Tool | A front-end tool, developed in DBE, allowing the user to formulate queries against the DBE knowledge using a tree-view representation of the Knowledge Structure. |
| RDBMS | Relational Database Management System: A DBMS (see DBMS description) based on the relational model. |
| Recommender | A DBE (autonomous) Core Service that will provide users (SMEs) with personalised knowledge by exploiting their profiles |
| 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). |
| SFE | Service Factory Environment: This is devoted to service definition and development. Users of the DBE will utilise this environment to describe themselves, their services and to generate software artefacts for successive implementation, integration and use |
| SMIF | Stream-based Metadata Interchange Format: A general format to save and exchange data of programs that are implementations of expositions models. |
| SQL | Structured Query Language: A language for querying relational data |
| 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 |
| XMI | XML Metadata Interchange: An SMIF (see SMIF description) standard specification based on XML. |
| XQuery | A Query language from the W3C that is designed to query collections of XML data. |

# 9. Bibliography


[1] The Object Management Group, ODM RFP, http://www.omg.org/cgi-bin/apps/doc?ad/03-09-06.pdf

[2] The Object Management Group, The Meta-Object Facility Specification http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf

[3] Object Management Group, XML Metadata Interchange Specification (v1.2): http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf

[4] W3C, XML 1.0 (third edition), W3C Recommendation. http://www.w3.org/TR/2004/REC-xml-20040204

[5] Java 2 Platform Enterprise Edition: http://java.sun.com/j2ee/index.jsp

[6] Java Metadata Interface Specification (v1.0): http://java.sun.com/products/jmi/index.jsp

[7] Java DataBase Connectivity (JDBC): http://java.sun.com/products/jdbc/

[8] NetBeans MetaData Repository: http://mdr.netbeans.org/

[9] The Object Management Group, MDA Guide Version 1.0.1: http://www.omg.org/docs/omg/03-06-01.pdf, 2003

[10] The Apache Derby Project: http://incubator.apache.org/derby/

[11] Object Management Group, UML 2.0 Infrastructure: http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-15.pdf

[12] ISUFI, DBE Deliverable: D16.1-BML First Release: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID=16&expandFolder=16&folderID=0

[13] SOLUTA.net, DBE Deliverable, D16.1 – Service Description Models and Language Definition: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID=16& expandFolder=16&folderID=0

[14] TCD, DBE Deliverable, D17.2 – Composer: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID =16& expandFolder=16&folderID=0

[15] TUC, DBE Deliverable, D17.1 – Recommender: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID =16& expandFolder=16&folderID=0

[16] TUC, DBE Deliverable, D14.1 – DBE Knowledge Representation Models: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID=16&expandFolder =16&folder ID=0

[17] The Object Management Group, Common Warehouse Metamodel (CWM) Specification v 1.1: http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf

[18] SOLUTA.net, DBE Internal Draft Document: “SM Software Model”, https://dev.digital-ecosystem.net/servlets/GetAttachment?list=dev&msgId=439&attachId=1

# 10. Appendix A: KB, SR proxy interfaces

## 10.1 KB Proxy Interface

This is the top-level interface for the KB proxy. The KB proxy can be used to access a KB peer for managing ontologies, semantic service descriptions, service descriptions, service compositions, storing service usage data and querying the KB.

| | |
|---|---|
| **ODI** | **getOntologyProxy()**<br>Get the proxy object that implements the ODI interfaces for the management of Ontologies in the DBE that are compliant with the ODM metamodel |
| **SSI** | **getSemanticServiceProxy()**<br>Get the proxy object that implements the SSI interfaces for the management of Semantic Service Descriptions (SSL Meta-model) |
| **SDI** | **getServiceDescriptionProxy()**<br>Get the proxy object that implements the SDL interfaces for the management of Service Descriptions (SDL Meta-model) |
| **SCI** | **getServiceCompositionProxy()**<br>Get the proxy object that implements the SDI interfaces for the management of Service Composition Descriptions (SCM Meta-model) |
| **UDI** | **getUsageDataProxy()**<br>Get the proxy object that implements the SSI interfaces for the management of Semantic Service Descriptions (SSL Metamodel) |
| **OQI** | **getQueryProxy()** |
| | |

## 10.2 SSL Proxy Interface

This is the interface for the SSL proxy object. The SSL proxy can be used to access a KB peer for managing semantic service descriptions. The interface SSI provides methods for creating, browsing, searching and storing SSL models in the KB

| | |
|---|---|
| void | **addToSSLModel(ServiceConcept sconcept)** |
| void | **exportSSLmodel(java.io.OutputStream outStream)** |
| void | **exportSSLmodel(java.lang.String filename)** |
| javax.jmi.reflective.RefPackage | **getSSLmodelPackage()** |
| void | **importSSLmodel(java.io.InputStream inStream)** |
| void | **importSSLmodel(java.lang.String filename)** |
| void | **KBgetSSLmodel(java.lang.String semanticPackage)** |
| void | **KBstoreSSLmodel()** |
| java.util.Collection | **KBlistSSLmodels()** |
| void | **newSSLmodel(java.lang.String semanticPackage, java.lang.String documentation)** |
| | |
| void | **generateSemanticServSpecificJMI(String directory)**<br>Generate code - SSL specific JMI interfaces –to target directory to manage models in the cache repository |
| | |
| The method generateSemanticServSpecificJMI() can generate code interfaces that are listed in the next column. OdmPackage is the outmost RefPackage. More information can be found at installed KB server pages. | AetypeAs, Association, AssociationClass, AssociationEnd, AssociationEndClass, AssociationsPackage, HasSourceAs, HasTargetAs, Source ,SourceClass, Target, TargetClass, Belongs2CategoryAs ,Belongs2DbedomainAs, ContactInformation, ContactInformationClass, CorePackage, DbeserviceDomain, DbeserviceDomainClass ,IndustryDomain, IndustryDomainClass, NameSpace, NameSpaceClass, OwnsAs OwnsAttributeAs, OwnsIdentifierAs, SemanticPackage, SemanticPackageClass, ServiceAttribute, ServiceAttributeClass, ServiceConcept,ServiceConceptClass ServiceIdentifier, ServiceIdentifierClass, ServiceParameter, ServiceParameterClass ServiceProfile, ServiceProfileClass, Condition, ConditionClass, FunctionalityParameter FunctionalityParameterClass, HasConditionAs,HasInputAs, HasOutputAs ,HasPreConditionAs OwnsFunctionAs ,ServiceBehaviorPackage ,ServiceFunctionality ServiceFunctionalityClass, ServiceInput ,ServiceInputClass ,ServiceOutput ServiceOutputClass,Interfaces ,DataRangeUri ,DataRangeUriClass DataTypeUri ,DataTypeUriClass,EnumTypeUri ,EnumTypeUriClass, Literal,LiteralClass,Multiplicity ,MultiplicityClass ,OntologyClassUri, OntologyClassUriClass, OntologyThingUri ,OntologyThingUriClass, TypesPackage, TypeUri, TypeUriClass Urireference, UrireferenceClass |

## 10.3 ODM Proxy Interface

This the interface for the ODM proxy object. The ODM proxy can be used to access a KB peer for managing DBE ontologies. The interface SSI provides methods for creating, browsing, searching and storing ontologis (ODM models) in the KB

| | |
|---|---|
| void | **exportOntology(java.io.OutputStream outSt)** |
| void | **exportOntology(java.lang.String filename)** |
| javax.jmi.reflective.RefPackage | **getODMPackage()** |
| void | **importOntology(java.io.InputStream inStream)** |
| void | **importOntology(java.lang.String filename)** |
| void | **init(java.lang.String serverAddr)** |
| void | **KBgetOntology(java.lang.String namespace)** |
| void | **KBstoreOntology()** |
| java.util.Collection | **KBlistOntologies();** |
| void | **newOntology(java.lang.String namespace)** |
| void | **generateOntologySpecificJMI(String directory)**<br>Generate code - ODM specific JMI interfaces –to target directory to manage models in the cache repository |
| | |
| The method generateOntologySpecificJMI(..) can generate code interfaces that are listed in the next column. SdlPackage is the outmost RefPackage. More information can be found at installed KB server pages. | *AllValuesFromAs, CardinalityAs, ClassAnnotationAs ,ClassesPackage ComplementOfAs, DeprecatedClass ,DeprecatedClassClass, DisjointWithAs, EquivalentClassAs, HasValueAs ,IntersectionOfAs MaxCardinalityAs ,MinCardinalityAs ,Odmclass, OdmclassClass, OneOfAs, OnPropertyAs, Restriction, RestrictionClass, SomeValuesFromAs, SubClassOfAs ,UnionOfAs, Value, ValueClass, ValueRange ValueRangeClass, AnnotationObject, AnnotationObjectAs, AnnotationObjectClass, AnnotationProperty AnnotationPropertyClass ,CorePackage ,Element, ElementClass, NamedElement, NamedElementClass, NameSpace, NameSpaceClass, OwnsAs, DataRange, DataRangeClass, Datatype, DataTypeAs DatatypeClass DatatypesPackage, DefinitionUriAs ,DepricatedDatatype ,DepricatedDatatypeClass, Enumeration, EnumerationClass,Literal ,LiteralClass NonNegativeInteger ,NonNegativeIntegerClass, OneOfAs PlainLiteral,PlainLiteralClass, PrimitiveType, PrimitiveTypeClass, TypedLiteral,TypedLiteralClass ,Urireference,UrireferenceClass, AllDifferent, AllDifferentClass ,ClassThing ,ClassThingClass, CttypeAs, DataTypePropertyThing, DataTypePropertyThingClass, DifferentFromAs, DistinctMembersAs, DtptdomainAs, DtptrangeAs, DtpttypeAs, IndividualsPackage, ObjectPropertyThing, ObjectPropertyThingClass ,OptdomainAs ,OptrangeAs, OpttypeAs, SameAsAs, Thing, ThingAnnotationAs, ThingClass, DomainAs, OntoAnnotationAs, Ontology, OntologyClass, OntologyPackage,OntologyProperty,OntologyPropertyClass, RangeAs ,DatatypeProperty, DatatypePropertyClass, DeprecatedDatatypeProperty ,DeprecatedDatatypePropertyClass DeprecatedObejctProperty, DeprecatedObejctPropertyClass, EquivalentPropertyAs, FunctionalDatatypeProperty, FunctionalDatatypePropertyClass ,FunctionalObjectProperty, FunctionalObjectPropertyClass, HasDataRangeAs,HasDomainAs, HasRangeAs, InverseFunctionalProperty, InverseFunctionalPropertyClass, InverseOfAs,ObjectProperty, ObjectPropertyClass, PropertiesPackage, Property,Property AnnotationAs, PropertyClass ,SubPropertyOfAs SymmetricProperty ,SymmetricPropertyClass, TransitiveProperty ,TransitivePropertyClass* |
| | |

## 10.4 SDL Proxy Interface

This the interface for the SDL proxy object. The SDL proxy can be used to access a KB peer for managing DBE service descriptions. The interface SSI provides methods for creating, browsing, searching and storing service descriptions (SDL models) in the KB

| | |
|---|---|
| void | **exportSDLmodel(java.io.OutputStream outStream)** |
| void | **exportSDLmodel(java.lang.String filename)** |
| javax.jmi.reflective.RefPackage | **getSDLPackage()** |
| void | **importSDLmodel(java.io.InputStream inStream)** |
| void | **importSDLmodel(java.lang.String filename)** |
| void | **init(java.lang.String serverAddr)** |
| void | **KBgetSDLmodel(java.lang.String semanticPackage)** |
| java.util.Collection | **KBlistSDLmodel()** |
| void | **KBstoreSDLmodel()** |
| void | **newSDLmodel(java.lang.String definitions)** |
| void | **generateSemanticServSpecificJMI(String directory)**<br>Generate code -SDL specific JMI interfaces– to target directory to manage models in the cache repository |
| | |
| The method generateOntology SpecificJMI(..) can generate code interfaces that are listed in the next column. SdlPackage is the outmost RefPackage. More information can be found at installed KB server pages. | CmpdefinitionsInterface, CmpdefinitionsMessage, CmpdefinitionsType<br>CmpinterfaceOperation, CmpmessagePart, CmptypeComplexType<br>ComplexType, ComplexTypeClass, Definitions, DefinitionsClass<br>DocumentedElement, DocumentedElementClass, Element<br>ElementClass, nterface, InterfaceClass, Message, MessageClass<br>Operation, OperationClass, Part, PartClass, ReffaultMessageOperation<br>RefimputMessageOperation, RefoutputMessageOperation<br>RefpartType ,SdlInteger ,SdlIntegerClass, SdlpackagePackage<br>SdlReal, SdlRealClass, SdlString, SdlStringClass, SemanticElement<br>SemanticElementClass, SimpleType, SimpleTypeClass, Type , TypeClass |
| | |

## 10.5 SCM Proxy Interface

This is the interface for the SCM proxy. The SCM proxy can be used to access a KB peer for managing DBE service composition descriptions. The interface SCI provides methods for creating, browsing, searching and storing service composition descriptions (SCM models) in the KB

| | |
|---|---|
| void | **exportSCMmodel(java.io.OutputStream outStream)** |
| void | **exportSCMmodel(java.lang.String filename)** |
| javax.jmi.reflective.RefPackage | **getSCMmodelPackage()** |
| void | **importSCMmodel(java.io.InputStream inStream)** |
| void | **importSCMmodel(java.lang.String filename)** |
| void | **KBdeleteSCMmodel(java.lang.String pname)** |
| void | **KBgetSCMmodel(java.lang.String ns)** |
| void | **KBstoreSCMmodel()** |
| void | **generateServCompositionSpecificJMI(String)**<br>Generate code -SDL specific JMI interfaces– to target directory to manage models in the cache repository |
| | |
| The method generateOntology SpecificJMI(..) can generate code interfaces that are listed in the next column. SdlPackage is the outmost RefPackage. More information can be found at installed KB server pages. | AHasScopes, AHasSource ,AHasTarget ,Activity ,ActivityClass ,ActivityOnMessage ,AnyUri ,AnyUriClass ,AsHasCopy ,Assign ,AssignClass ,BooleanExpr ,BooleanExprClass ,BpelPackage ,Bpelcase ,BpelcaseClass ,Bpelcatch ,BpelcatchAll ,BpelcatchAllClass ,BpelcatchClass ,Bpelfrom ,BpelfromClass ,Bpelinvoke ,BpelinvokeClass ,Bpelprocess ,BpelprocessClass ,Bpelswitch ,BpelswitchClass ,Bpelthrow ,BpelthrowClass ,Bpelto ,BpeltoClass ,Bpelwait ,BpelwaitClass ,Bpelwhile ,BpelwhileClass ,CHasActivities ,CatchActivities ,CatchAllActivity ,CatchAllCompensate ,CatchHasVar ,CatchInvoke ,Compensate ,CompensateClass ,CompensationHandler ,CompensationHandlerClass ,Copy ,CopyClass ,CopyFrom ,CopyTo ,Correlation ,CorrelationClass ,CorrelationSet ,CorrelationSetClass ,CsHasProperties ,Empty ,EmptyClass ,EventHandlers ,EventHandlersClass ,FHasActivities ,FHasLinks ,FaultHandlers ,FaultHandlersClass ,Fhcatch ,FhcatchAll ,Flow ,FlowClass ,HasCors ,HasInput ,HasOutput ,HasPartnerLink ,InvHasCatchAll ,InvHasCh ,Link ,LinkClass ,MessageActivity ,MessageActivityClass ,OMhasPartnerLink ,OMhasVar ,OnAlarm ,OnAlarmClass ,OnAlarmEh ,OnMessage ,OnMessageClass ,OnMessgCorrelations ,OnMessgEh ,Otherwise ,OtherwiseActivity ,OtherwiseClass ,PHasActivities ,PHasChandler ,PHasCorSets ,PHasEhandlers ,PHasFhandlers ,PHasPartLink ,PHasPartLinks ,PHasPartners ,PHasVars ,Partner ,PartnerClass ,PartnerLink ,PartnerLinkClass ,Pick ,PickClass ,PickOnAlarm ,PickOnMessage ,Property ,PropertyClass ,Receive ,ReceiveClass ,RechasVar ,RepHasVar ,Reply ,ReplyClass ,SHasCase ,SHasLink ,SHasOtherwise ,ScHasActivities ,ScHasChandler ,ScHasCorSets ,ScHasEhandlers ,ScHasFhandlers ,ScHasVars ,ScmPackage ,Scope ,ScopeClass ,Sequence ,SequenceClass ,Source ,SourceClass ,SqHasActivities ,THasLink ,Target ,TargetClass ,Terminate ,TerminateClass ,ThrHasVar ,TypesPackage ,Variable ,VariableClass ,XsdataType ,XsdataTypeClass |
| | |

## 10.6 QML Proxy Interface

This is the interface for the QML proxy. The QML proxy can be used to formulate a query model on DBE metamodels. The interface QMI provides methods for creating, browsing, submitting query models and getting collections of results from the KB

| | |
|---|---|
| void | **exportOueryModel(java.io.OutputStream outStream)** |
| void | **exportOueryModel(java.lang.String filename)** |
| javax.jmi.reflective.RefPackage | **getOueryModelPackage()** |
| void | **importOueryModel(java.io.InputStream inStream)** |
| void | **importOueryModel(java.lang.String filename)** |
| void | **KBdeleteOueryModel(java.lang.String pname)** |
| void | **KBgetOueryModel(java.lang.String ns)** |
| java.util.Collection | **KBsearchOueryModel(java.lang.String name)** |
| void | **KBstoreOueryModel()** |
| | |
| void | **generateQuerySpecificJMI(String directory)** |
| The method generateOntology SpecificJMI(..) can generate code interfaces that are listed in the next column. OqlPackage is the outmost RefPackage. More information can be found at installed KB server pages. | AppliedPropertySourceAssoc, AssociationEndCallExp, AssociationEndCallExpClass, AssociationEndNavigationSourceAssoc, AssociationEndReferredAssociationEndAssoc, AttributeCallExp, AttributeCallExpClass, AttributeCallExpReferredAttributeAssoc, AttributeContextDecl, AttributeContextDeclClass, AttributeContextDeclTypeAssoc, BagType, BagTypeClass, BaseExpResultAssoc, BooleanLiteralExp, BooleanLiteralExpClass, CollectionItem, CollectionItemClass, CollectionKind, CollectionKindEnum, CollectionLiteralExp, CollectionLiteralExpClass, CollectionLiteralExpPartsAssoc, CollectionLiteralPart, CollectionLiteralPartClass, CollectionLiteralTypeAssoc, CollectionRange, CollectionRangeClass, CollectionRangeLastAssoc, CollectionType, CollectionTypeClass, CollectionTypesElementTypeAssoc, ConditionIfExpAssoc, ConstraintContextDecl, ConstraintContextDeclClass, ContextDeclaration, ContextDeclarationBodyExpressionAssoc, ContextDeclarationClass, ContextDeclarationsPackage, CorePackage, DefinitionContextDecl, DefinitionContextDeclAttributesAssoc, DefinitionContextDeclClass, DefinitionContextDeclOperationsAssoc, ElseExpressionIfExpAssoc, EnumLiteralExp, EnumLiteralExpClass, EnumLiteralExpReferredEnumerationAssoc, ExpressionsPackage, FirstCollectionRangeAssoc, IfExp, IfExpClass, InLetExpAssoc, InitializedVariableInitExpressionAssoc, IntegerLiteralExp, IntegerLiteralExpClass, InvariantContextDecl, InvariantContextDeclClass, IsMarkedPre, IsMarkedPreClass, ItemCollectionItemAssoc, IterateExp, IterateExpClass, IteratorExp, IteratorExpClass, LetExp, LetExpClass, LetExpVariablesAssoc, LiteralExp, LiteralExpClass, LoopExp, LoopExpBodyAssoc, LoopExpClass, LoopExprIteratorsAssoc, ModelPropertyCallExp, ModelPropertyCallExpClass, NumericalLiteralExp, NumericalLiteralExpClass, OclExpessionTypeAssoc, OclExpression, OclExpressionClass, OclModelElementType, OclModelElementTypeClass, Operation, OperationCallExp, OperationCallExpClass, OperationCallExpReferredOperationAssoc, OperationClass, OperationContextDecl, OperationContextDeclClass, OperationContextDeclOperationAssoc, OperationDefinition, OperationDefinitionBodyExpressionAssoc, OperationDefinitionClass, OperationDefinitionParametersAssoc, OperationDefinitionTypeAssoc, OperationParametersAssoc, OperationTypeAssoc, OrderedSetType, OrderedSetTypeClass, ParentOperationArguments, PrimitiveLiteralExp, PrimitiveLiteralExpClass, PropertyCallExp, PropertyCallExpClass, QualifiersAssociationEndCallExpAssoc, QueryContextDecl, QueryContextDeclClass, QueryContextDeclTypeAssoc, RealLiteralExp, RealLiteralExpClass, SequenceType, SequenceTypeClass, SetType, SetTypeClass, StringLiteralExp, StringLiteralExpClass, ThenExpressionIfExpAssoc, TubleTypeVariablesAssoc, TupleLiteralExp, TupleLiteralExpClass, TupleLiteralExpTuplePartAssoc, TupleType, TupleTypeClass, TypesPackage, VariableDeclaration, VariableDeclarationClass, VariableDeclarationTypeAssoc, VariableExp, VariableExpClass, VariableExpReferredVariableAssoc, VoidType, VoidTypeClass, |

## 10.7 UD Proxy Interface

This is the interface for the UD proxy. The UD proxy can be used to access a KB peer for storing service usage data

| | |
|---|---|
| **void** | **flushData()**<br>Flush data to KB. |
| **void** | **insertCSdata(int zip, java.lang.String city, java.lang.String streetName, int streetNum)**<br>Store Address Information at client side |
| **void** | **insertCSPdata(java.lang.String operationName, long replyTS, long servReplyTime, long roundTripTime, long reqTS, java.lang.String serviceUri, long servReplyTS)**<br>Data available during service execution at client side |
| **void** | **insertSSPdata(java.lang.String operationName, long reqTS, long client2servTime, long clientReqTS, long reqExecutionEndTS, long serviceTime, java.lang.String serviceUri)**<br>Data available during service execution at service side |
| **void** | **setCacheSize(int size)**<br>Set the size of cache at proxy side.<br>Set the size of cache at proxy side. When the data kept in the proxy exceed the cache limit all the data collected are sent in KB (batch mode) and the cache is cleared. |
| **void** | **SSdata(java.lang.String SOAPbody, java.lang.String encoding, java.lang.String SOAPactionUri, java.lang.String transportType, long clientTimeout, java.lang.String operationStyle, java.lang.String operationRequested, java.lang.String RemoteIPAddress, java.lang.String serviceUri, java.lang.String serviceName, java.lang.String msgType)**<br>Store Service- specific data |
| | |

## 10.8 SR Proxy Interface

This is the interface for the SR proxy. The SR proxy can be used to publish search and delete Service Manifests from the Semantic Registry Service. The proxy can also be used to formulate a query model on DBE metamodels. The SR proxy provides methods for creating, browsing, submitting query models and getting collections of results (service manifests) from the Semantic Registry

| | |
|---|---|
| void | **publishSM(java.io.InputStream ins)**<br>Reads a service manifest from the input stream and streams it to the Semantic Service Registry and publish it. |
| void | **eraseSM(String SMID)**<br>Deletes a service manifest from the Semantic Registry. |
| java.io.InputStream | **getSM(String SMID)**<br>Retrieves specific Service Manifest from the Semantic Registry and returns an input stream to read the service manifest. |
| OQI | **getQueryProxy()**<br>Get the query proxy object to perform queries in the semantic registry. |
| | |

## 10.9 SM Toolkit

This is the interface of the SM Toolkit. The toolkit is an auxiliary library that exports functionality to construct new or alter existing Service Manifests. It is used by the SM manager to extract specific elements (models and data) that are bundled together in the SM document

| | |
|---|---|
| void | **exportServiceManifest(java.lang.String smid, java.io.OutputStream outs)**<br>Export the current service manifest in the specified output stream with a specific id |
| void | **exportServiceManifest(java.lang.String smid, java.lang.String filename)**<br>Export the current service manifest in the specified file with a specific id |
| java.io.InputStream | **getBMLdata()**<br>Extract the BML data part of the current Service Manifest |
| java.io.OutputStream | **getBMLdataOutStream()**<br>Get an output stream to write the BML data document that should be included in the SM |
| java.io.InputStream | **getBMLmodel()**<br>Extract the BML model part of the current Service Manifest |
| java.io.OutputStream | **getBMLmodelOutStream()**<br>Get an output stream to write the BML model document that should be included in the SM |
| java.io.InputStream | **getSDLmodel()**<br>Extract the SDL model part of the current Service Manifest |
| java.io.OutputStream | **getSDLmodelOutStream()**<br>Get an output stream to write the SDL model document that should be included in the SM |
| void | **importServiceManifest(java.io.InputStream ins)**<br>Import and process a Service Manifest by reading it from the specified input stream |
| void | **importServiceManifest(java.lang.String filename)**<br>Import and process a Service Manifest by reading it from the specified file |
| void | **setSMID(java.lang.String smid)**<br>Set the ID of the current Service Manifest |
| | |