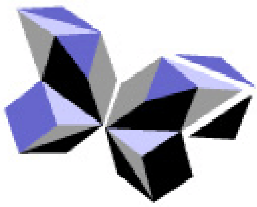

Digital Business Ecosystem

Contract n° 507953

WP14: DBE Knowledge Base

D14.3: 1st P2P distributed implementation of the DBE Knowledge Base and Semantic Registry

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°: D14.3

Due date: 31/12/2005

Delivery Date: 11/01/2005

Short Description:

This document describes the design and the first Peer-to-Peer (P2P) distributed prototype implementation of the DBE Knowledge Base (KB). It accompanies the software delivered under the DBE Studio and Swallow sourceforge projects.

The KB architecture adopts the MOF metadata framework of OMG and the implementation exploits a combination of a metadata repository and a native XML 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 three important DBE core services, the Knowledge Base Service (KB service), the Semantic Registry Service (SR service), and the Recommender Service (RC service) are presented. These services exploit the basic functionality of the DBE KB infrastructure and are used to support important tasks in the Service Factory and the Service Execution environment. DBE tools and components that wish to access these core services are able to lookup for and communicate with available service instances. These service instances cooperate with each other to form a P2P network and enable an effective distributed knowledge discovery process while at the same time guaranteeing high availability in the knowledge access. The knowledge access mechanisms that are mentioned in this document provide the underlying mechanism for querying metamodels, models and instances of DBE and form the basis for supporting recommendations.

Partners owning: TUC

Partners contributed: TUC

Made available to: All project partners and the EC

Versioning

Version	Date	Author, Organization	Description
0.1	10/3/2005	NIKOS PAPPAS – TUC FOTIS G. KAZASIS – TUC GIORGOS ANESTIS – TUC NEKTARIOS GIOLDASIS – TUC	Initial Document Creation
0.2	21/12/2005	FOTIS G. KAZASIS – TUC NIKOS PAPPAS – TUC PROF. STAVROS CHRISTODOULAKIS - TUC	Major Revisions/Additions
1.0	12/1/2005	FOTIS G. KAZASIS – TUC NIKOS PAPPAS - TUC	Final Version Submitted

Quality check

1st Internal Reviewer: Miguel Vidal – SUN (Technical Coordinator)

2nd Internal Reviewer: All DBE Computing Partners

Table of contents

Executive Summary

In the DBE environment a large number of SME nodes will be pooled together to share their resources, information and services while keeping themselves fully autonomous. The challenge in designing and developing the DBE Knowledge Base (DBE KB)[1] is to enable the effective support of rich semantics and complex queries across multiple sources in an unstructured and self-organized peer-to-peer (P2P) network such as the DBE network. The objectives of the first P2P implementation of the DBE KB were to exploit the current P2P infrastructure of the ExE environment and to offer high availability in knowledge access as well as an effective distributed knowledge discovery mechanism. The technical approach followed in order to fulfill the first objective is presented in this document whereas the one that was followed in order to fulfill the second objective is presented in Deliverable D14.4 – 2nd Release of the Recommender [38].

The document first provides an analysis on the P2P nature of the DBE KB. To do so, we initially talk about the requirements that ask for a P2P KB in the DBE as well as about the given facts of DBE and the way that they affect the design of such a KB. Afterwards, we shortly describe the current status of P2P computing and its various applications for Knowledge Management, and we present the technical approach to the P2P KB according to the current understanding of the problem. Finally we conclude with a list of the technical challenges whose meeting will be continuously in focus until the end of the DBE project since at this time point it is by no means clear that the proposed distributed management approach will work effectively in practice. Many things (requirements, assumptions, and approaches) may need to be changed as the project will advance and the issues related to Knowledge Base will be better shaped and understood.

The document accompanies the software of the 1st P2P distributed implementation of the DBE KB that has been integrated as part of the DBE Studio and Swallow projects and will be demonstrated during the 2nd DBE Audit (January 2006). This work is also associated with the prototype milestone M14.4 "1st Release of P2P distributed DBE KB and SR" (month October 2005) that provides the basic distributed platform for the DBE KB infrastructural services. The Appendix briefly presents the available interfaces for accessing the DBE Knowledge Base, the Semantic Registry and the Recommender. The source code of these services is available within the swallow project in http://swallow.sourceforge.net.

It should be noted that since the deliverable D24.3 "DBE Peer-to-Peer Architecture Design" [37] has been delivered within December 2005, the design of the P2P distributed implementation of the DBE KB has only considered earlier versions of this document provided in July 2005.

[1] With the term Knowledge Base we mean the entire infrastructure needed to handle the DBE Knowledge. This infrastructure will provide different services like Semantic Registry Service and Knowledge Base Service as it has been described in various documents.

1. Requirements for a P2P Knowledge Base

1.1 Introduction

Before starting the investigation of any technical requirement or approach to the design and implementation of a P2P distributed DBE Knowledge Base, we should answer a fundamental question:

What is the need for a P2P knowledge Base in DBE?

Regarding this issue the DBE Technical Annex states: "A crucial point about the knowledge base is related to its growth. Since the ontologies present in the knowledge base represent the map and the common vocabulary of all the data present in the Europe-wide distributed storage DBE system, it is important to manage their growth through software models and organizational forms. Preserving the distributed and open approach guarantees that improvement will not affect consistency of information. The KB is a constituent part of the DBE environment, it is part of it, and it will consist, at a more technical level, of a wide distributed network of "knowledge collectors" that will receive signals from the environment. It will not be a single server machine but it will be implemented as a distributed information system that spans the DBE. The Internet network latency, the large number of SME services, the "fast" response required and the mass of information to be captured will be a severe constraint when establishing the most appropriate technology and development approach".

A first analysis, which was attempted in [1] actually showed that, from a technical point of view, the main requirement for developing a distributed Knowledge Base in DBE is related to data availability. That is, DBE needs a knowledge base, which will not have single points of failure. Replicating the knowledge of DBE into a set of identical KB instances and allowing synchronization between them in order to provide a distributed DBE KB that will not suffer from a single point of failure could meet this requirement. However, there are DBE viability (sustainability) requirements that make the adoption of this approach difficult and ask for a fully distributed and peer-to-peer Knowledge Base.

The most important of these requirements refers to the "no central authority" issue. That is, in DBE there is no central authority that "owns" or manages a community. Each community is built (probably with the guidance of regional catalysts) and growing by itself, without a central authority that owns this community, its knowledge, etc. The assumption that the DBE Knowledge will be kept in one single machine (or replicated into a set of machines) means that some central coordination will exist and thus the community will be somehow controlled.

Another requirement regarding the viability of the DBE project is the scalability of the system being built. It is very important to design and implement the DBE KB with scalability in mind. The reason is simple: the larger the market, the more the business opportunities for participating SMEs. Taking into account that the DBE KB is managing the knowledge of the entire community (SME business models, SME service offerings, SME profiles, etc.) or outside of it (e.g. regulatory frameworks) becomes clear that its ability to scale is crucial for the success of the entire system. It should be noted however that at this moment it is difficult to estimate the volume of data that is going to be stored in the DBE KB. We thus believe that the approach to the distributed KB should ensure as much scalability as possible.

Moreover, at any point in time DBE may get spontaneous users. Users that are not part of the DBE and they want a "short time opportunistic" cooperation if possible. Such autonomy can be achieved more easily in a P2P environment where the presence/absence of a node does not affect the system's operation.

1.2 DBE Facts and Functional Requirements Regarding the P2P DBE Knowledge Base

Once we have showed why a P2P distributed knowledge Base is needed, we present in this sub-section the functional requirements for the P2P distributed DBE Knowledge Base as well as the facts that affected its design and development. These requirements are the following:

RQ1. Common Languages: DBE knowledge [33] is modeled and captured through a set of common languages. In particular, the Ontology Definition Metamodel (ODM) is used to model all the domain ontologies, the Business Modelling Language (BML) is used to model the descriptions of all SMEs participating in DBE, and Semantic Service Language is used to model the service offerings of SMEs. Data is also described in a consistent way through appropriate instantiation metamodels. The profiles of SMEs have been also uniformly described [34] with one language (exploiting the Query Metamodel Language-QML [35]) whereas the various legislations will be described with a similar way (through the Knowledge Base Model for the Regulatory Framework). The fact that common languages are used to capture specific aspects of DBE Knowledge means that although there are variant semantics, there is no proprietary knowledge representation.

RQ2. Complex (structured) Knowledge Representation and Access: In contrast to most P2P systems where resources are described by their name or at most with a set of keywords, DBE Knowledge is organized complex structured documents based on powerful languages (ODM, BML, SSL, SDL, etc.). Moreover, user queries are supposed to be complex queries (e.g. including conjunctions) with access criteria on the structured knowledge at a great granularity level (e.g. criteria on the content of the documents). That is, in contrast to the existing file sharing P2P systems where a user cannot search for parts of a file, in the DBE KB one should be able to acquire knowledge by posing complex, fine-grained, queries on parts of the knowledge that he is looking for. For example, to search for a Service Manifest that contains in its SSL description the Hotel Facility "Parking" and the functionality "Cancel Hotel Reservation". Practically this requirement means that Distributed Hash Table (DHT) approaches (like those in CAN [3], CHORD [2], Tapestry [5], etc.) for content placement and lookup cannot be directly followed for the DBE P2P KB.

RQ3. Domain Ontologies: Domain Ontologies are created (or imported) by some expert (e.g. regional catalysts) and are used as a community-wide accepted conceptualization of a business domain. More than one domain ontologies may exist for a particular business domain. Business and Service Models make use of ontology concepts in order to enable semantic interoperability between SMEs. Domain Ontologies are not modified by SMEs.

RQ4. User (Local) Ontologies: SMEs may use their own ontologies for enhancing their business and service descriptions. In such a case, mappings between domain ontologies and user ontologies may be needed [29] in order to enable semantic discovery and interoperability among partners and services. These mappings will be provided by the owners of local ontologies.

RQ5. SME's IT Infrastructure: SMEs should be able to participate in DBE without the obligation to contribute a large amount of IT resources. For the design of the P2P knowledge base we consider that not each SME has the appropriate IT infrastructure to host a KB instance.

RQ6. Regional Catalyst's Role: At the moment of writing, the operational role of Regional Catalysts in DBE has not been fully clarified. Will they contribute resources (i.e. will be nodes of the Distributed System) to a DBE community? In such a case, the P2P design of the DBE KB could be highly affected since, as it is also mentioned in [1], it is reasonable to think that these nodes will be more powerful than other typical nodes and their presence in the P2P network won't be so fluctuating. In general, the business model and exploitation plan of DBE is still not clear. For the design of the P2P KB (and the related services) the existence of special nodes (e.g. nodes of catalysts or nodes that have been found to be reliable) in the P2P network could be taken into account. The process and the criteria according to which a node is characterized/uncharacterized as special node is outside the scope of the design and implementation of the P2P KB.

RQ7. KB Components/Services: As noted, the term KB means the infrastructure needed to manage the DBE knowledge. This infrastructure consists of a set of components that serve different but complementary purposes and expose different services to the rest DBE System. It is worthy to distinguish between the different components that constitute the DBE KB. These components are:

a. **KB Model Repository:** The KB Model Repository (or simply Model Repository) manages the various kinds of Models that SMEs are creating and storing into the DBE

Knowledge Base. Such models include: BML Models, SSL Models, SDL Models, and Service Composition Models (BPEL). This model repository is contacted by other DBE components through the Knowledge Base Service (KBService). The BML, SDL, and BPEL Editors, and the Recommender are typical clients of the Model Repository.

b. **Semantic Registry:** The Semantic Registry manages the descriptions of the SME offerings in the DBE. These descriptions refer to semantic description (models and data) of SMEs and their services (BML/SSL descriptions), as well as to the technical descriptions of those services (BPEL and SDL models). These descriptions are packaged together in the form of Service Manifests (SM) and are published into the Semantic Registry where clients (SMEs or end users) search for candidate services. The Semantic Registry is contacted by other components through the Semantic Registry (SR) Service. Typical clients of the SR are (among others) the Semantic Discovery Tool, the Recommender, the Composer, etc.

c. **Ontology Repository:** The Ontology Repository manages the (Domain and User) Ontologies (created or imported) in the DBE and is accessible through the KB service. Ontologies are referenced by business and service models in order to enable semantic interoperability. Thus, there is a strong dependency between the Ontology Repository and the other components of the KB that store contents, which make use of ontologies (e.g. the model repository, and the semantic registry).

d. **Regulatory Framework Repository:** The Regulatory Framework Repository will manage the regulatory framework(s) of the DBE. At the moment of writing, the regulatory framework is thought as special kind of ontology containing legislation statements in the form of conditional rules.

RQ8. Knowledge Availability: The DBE Knowledge needs to be highly available. Availability in DBE means that once a user stores a piece of knowledge (e.g. a BML Description) in the KB, he/she should be able to access this knowledge every time. This is difficult to be achieved if this piece of knowledge is forwarded and stored in some other node in a P2P network without replication since in this case the availability of the knowledge depends exclusively on the presence of that node in the P2P network (and its accessibility). Moreover, it is reasonable to claim that each user should have access to its own knowledge (BML/SSL Models and Data, SDL Models, etc.) even if it is off-line (not connected to the network). This means that each SME should ideally have the appropriate infrastructure to store its own knowledge locally (besides the replicas that will be stored in other nodes). On the other hand, this requirement may conflict with the requirement RQ5. It is expected that each SME will be willing to have control in its own knowledge, and thus it will provide the appropriate IT infrastructure to host a KB instance. However, support for SMEs with no IT infrastructure (or SMEs unwilling to host a KB instance) needs to be provided.

RQ9. Knowledge Access Control: The knowledge owned by an SME should be updated only by that SME or by the system acting on behalf of that SME (e.g. the profiling mechanism when performing profile adaptation). For example, the Business Model or the Profile of an SME should not be modified by other users of the system. Moreover, parts of the Knowledge that an SME has in DBE should not be accessed by others. Thus, we should distinguish between public and private knowledge. If each SME was supposed to have its own instance of KB (a KB node) then the KB design could consider the local knowledge of an SME (the knowledge stored in it's own KB node) as private and the knowledge replicated in other nodes (in the network) as public. However, since not each SME can install a local KB node, the KB should provide access control mechanisms for keeping SME knowledge as private.

2. Approach to the P2P Knowledge Base

2.1 Introduction

Based on the facts and the functional requirements mentioned above, this section presents the first approach to a P2P Knowledge Base. Before starting the discussion on particular approaches and alternatives, it would be worthy to mention the objectives and the exploitation use cases of the DBE Knowledge Base, since the ultimate criterion according to which any approach will be selected is the degree to which it facilitates the achievement of the KB objectives and its exploitation use cases.

The goal of the DBE knowledge base is 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. This goal can be broken down into the following objectives:

- Representations of domain specific ontologies (common conceptualization in a particular domain);
- Descriptions of the SMEs themselves in terms of business models, processes, policies, strategies, views etc.;
- Description of the SME value offerings (services) and the achieved solutions (service chains/compositions) to particular SME needs.
- Description of the Regulatory Framework that applies to DBE business community.

The exploitation of the DBE knowledge can be summarized into two strategic use cases. These use cases are:

1. **Semantic Partner/Service Discovery:** Use of rich semantics to advertise and discover partners (SMEs) and services enabling more business opportunities. This use case can be refined into:
    - Service Searching: Based on the description of a desirable service *S*, the system may find Service Descriptions that match the service description of *S*.
    - Service Matching: Based on the preferences of a specific SME *A* for possible services, the system may find Service Descriptions that match the service preferences of *A*.
    - Business Matching: Based on the preferences of a specific SME *A* for possible partner SMEs, the system may find SMEs that match the business preferences of *A*.
2. **Semantic Interoperability**: Use of semantics (ontologies) to allow a common understanding among SMEs enabling them to easily inter-operate (and thus cooperate) with each other.

The accomplishment of the first use case requires a powerful knowledge access language that will allow the specification of complex queries for content discovery. This means that the DBE Knowledge Base needs to be built on top of functionality (like those provided by knowledge management systems) that allows the specification of complex queries against complex knowledge organization. Thus, content-based discovery in traditional DHT systems, where a lookup (key) operation (which returns the identity, e.g. the IP address, of the node storing the object with that key) is provided, is not satisfactory for the objectives of the DBE Knowledge Base since they only provide for exact matching content discovery.

The DBE Distributed Storage System (DSS) as a distributed storage and file system uses a DHT approach to provide both content placement and lookup (as explained in [1]) in the level of file objects. That is, one can store a file object in the DSS, which transparently (based on a hashing function) will locate this file in a certain node in the network and return a key to the file owner. The only way to lookup the file is by using that key. It is obvious that this functionality is not suitable in order to be used for knowledge management.

2.2 Technical Assumptions

When considering data management, the main requirements of a P2P system are [36]:

- *Autonomy*: an autonomous peer should be able to join or leave the system at any time without restriction. It should also be able to control the data it stores and which other peers can store its data, e.g. some other trusted peers
- *Query expressiveness*: the query language should allow the user to describe the desired data at the appropriate level of detail. The simplest form of query is key look-up, which is only appropriate for finding files. Keyword search with ranking of results is appropriate for searching documents. But for more structured data, a schema-based query language is necessary.
- *Efficiency*: the efficient use of the P2P system resources (bandwidth, computing power, storage) should result in lower cost and thus higher throughput of queries, i.e. a higher number of queries can be processed by the P2P system in a given time.
- *Quality of service*: refers to the user-perceived efficiency of the system, e.g. completeness of query results, data consistency, data availability, query response time, etc.
- *Fault-tolerance*: efficiency and quality of services should be provided despite the occurrence of peers' failures. Given the dynamic nature of peers that may leave or fail at any time, the only solution is to rely on data replication.
- *Security*: the open nature of a P2P system makes security a major challenge since one cannot rely on trusted servers. Wrt. data management, the main security issue is access control which includes enforcing intellectual property rights on data contents.

Efficiency and Security are the two requirements that are going to be considered in Deliverable D24.3 [37]. Our design of the p2p distributed Knowledge Base meets the rest of them. In specific, this report describes the mechanisms that have been elaborated in order to support autonomy, and fault-tolerance. The DBE Deliverables D17.1 "Recommender" [35] and D14.4 "2nd Release of the Recommender" [38] provide the mechanisms for query expressiveness and quality of service respectively.

The following technical assumptions have been made for the design of the P2P Knowledge Base:

1. **P2P Communication Infrastructure:** The P2P DBE KB Implementation pre-assumes the existence of a P2P communication infrastructure (P2P CI). With P2P CI we mean the primitive functionality for peer identification, logical peer addressing, peer endpoint routing, peer binding and communication, NAT and Firewall Traversal, etc. Systems that provide such functionality are (among others) the SUN's JXTA [6] and FADA [7] frameworks. The DBE Knowledge Base is a Knowledge Management Infrastructure (overlay) layer on top of a P2P communication network. Similar approaches have been followed in [8], [9], and [10] which build on top of the JXTA framework, as well as in [11] where there is also a clear separation between the communication and the knowledge management layers. In specific the DBE KB utilizes the DBE Servent's functionality that transparently encapsulates the underlying P2P network (Swallow project). In this sense the Knowledge Management Infrastructure is network-independent and therefore it could be implemented over different P2P Networks (unstructured, super-peer, etc.).
2. **System Level Optimization:** When building P2P applications one should take into account the reliability of peers (which actually means parameters of the physical systems e.g. computing power, storage, etc. and their connectivity e.g. bandwidth, latency, etc.) in order to optimize the performance of the application. The design and implementation of the DBE P2P KB pre-assumes that appropriate services (designed and built in other tasks in the project e.g. P2P architecture, P2P Implementation, etc.) will provide such "system-level" optimization functionality to the upper layer services. Another important goal that these services can serve is the indication of special nodes (nodes that can have more responsibilities – e.g. super peers).
3. **Types of Nodes (or Peers):** Following an approach similar to that proposed in [1], the P2P design of the DBE KB assumes that there will be two different kinds of nodes (peers):
    a. **Simple Nodes:** These nodes do not (or cannot) have a KB infrastructure installed locally. The reason for this may be the fact that they do not have the appropriate IT infrastructure, the fact that they are un-reliable, or their unwillingness to host a KB instance. For their Knowledge Management Requirements (e.g. storage of their Business Models, Service Manifests, etc.) they will connect to KB nodes in order to get access to the P2P KB infrastructure. The support of this kind of nodes meets the functional requirement RQ5.
    b. **KB Nodes:** These nodes are (ideally) more reliable and they are willing to have a KB instance installed (contributing this way to P2P knowledge Base). The local KB will not only store knowledge of that node, but also it will keep replicas of knowledge hosted in various

other KB nodes as well as knowledge of simple nodes. From a KB perspective these nodes will not only host a KB instance but they will also be assigned indexing responsibilities. Considering the most reliable and capable KB nodes of the P2P network one could assume that these nodes will be well connected with each other (i.e. they will have enough bandwidth and they will sustain a high number of connections).

4. **Read Frequently, Update rarely:** It is difficult to make a global assumption regarding the read/write ratio of all the KB contents. This is because the various contents of the DBE KB will not be read and updated with the same frequency. For most of the KB components reads will be much more frequent than updates. For example, a model in the model repository or a Service Manifest in the Semantic Registry will be frequently read but rarely modified. On the other hand, user profiles will be modified more frequently by the profiling mechanism, which will be tracing the user actions and transactions. It is however, more accurate (at the moment) to assume that KB contents will be read more frequently than updated.
5. **Granularity of Knowledge Replication:** Knowledge should be replicated in more than one peer (nodes) in order to ensure high availability. Replication can be done in two levels: a) the entire contents of a KB node are replicated to other nodes, and b) knowledge items (e.g. documents like models, SMs, or Ontologies) are replicated to other nodes. The DBE KB follows the second approach since it allows one to distinguish among the various contents (documents) contained in a KB Peer and to support content placement based on semantic criteria.
6. **Consistency:** The consistency requirements among replicated knowledge items are not the same for all kinds of knowledge. For instance, updates on the SDL part of a SM (of a service) need to be immediately propagated to all replicas since the consumption of that service (using the old SDL) may not be possible any more. On the other hand, the update in some ontology is not so crucial for the operation of the system and thus some delay in synchronization can be sustained. A further investigation of this issue will allow us to make more realistic assumptions of the consistency requirements.
7. **Access Control based on the Identity Service:** Access control to knowledge items needs to be provided by the DBE Knowledge Base especially for knowledge updates. To support this functionality, authorization support is needed. Authorization is closely related to Identity Management and the corresponding functionality will be exploiting the Distributed Identity Service of DBE [37], [1].
8. **Semantic Groups:** To avoid the very expensive search cost in a completely unstructured environment some grouping of nodes like interest groups, acquaintances etc. may be considered. Acquaintances may be transitive relationship. These may be used to restrict the scope of the search (i.e. to nodes that refer to common semantics and thus have good chances to satisfy a query). Such groupings may control how queries are propagated and results are assembled.

2.3 Current Status in P2P Information Systems

During the time life of the information technologies the computer systems have emerged from centralized systems to distributed systems. Distributed systems comprise computers installed at different sites, each of them performing independent data processing. Usually each computer of the distributed system is specialized to perform a range of activities or support a certain workplace.

The most widely used distributed systems are those with client-server architecture. One part of such a system (the servers) provides common operations, while another (clients) provides direct user services. The client-server model is convenient in the sense that it allows to separate specialized common tasks (servers) from individual user tasks (clients). Client tasks are usually simpler and communicate directly with the user while server tasks are much more complex.

The last years we have seen in distributed systems the emerging of the peer-to-peer computational paradigm. P2P is trying to extend and alter strategies already applied in client-server computing by blurring the role differentiation among clients and servers in order to achieve a viable dynamic computer network where peers share resources and services. P2P systems are being adopted in many application areas as a

solution to problems such as the limitations of static topologies, the heavy administration needs, and the dissemination of data sources over the Internet.

P2P can be defined as network architecture without centralized coordination. Each node/peer is autonomous (can join and leave at their will) and acts as client and server at the same time and is sharing content and/or resources by direct exchange with other peers.

The literature contains many architectural approaches that can categorize the P2P systems in two main categories: Hybrid and Pure P2P systems. In Hybrid approach some tasks are handled via centralized components while other tasks are decentralized between two peers. In pure P2P systems there is no differentiation among peers. Each peer only knows neighbours that might change during lifetime.

Pure P2P systems can be structured or unstructured. Unstructured P2P systems like Gnutella [39] use the flooding technique for discovery of appropriate data sources. In flooding a peer sends the query to all of its neighbours. If the neighbour has the result, it will notify the asking peer and can send the result directly to it, else the neighbour will decrease TTL (Time To Live) and forward the query to its neighbours as long as TTL permits it.

Other approaches proposed for unstructured P2P networks are trying to reduce the processing of queries into fewer nodes. These are the iterative deepening [19] that introduces the idea of progressing TTL, random walks [20] that tries to narrow the forwarding of queries to neighbours based on some heuristics and informed search that exploits simple indices that are being constructed from previous query results in combination to flooding [21], [22]. Informed search can be extended to support content replication in addition to index construction from previous queries [20]. The main drawbacks of the solutions applied to unstructured P2P systems are the large amount of messages exchanged, redundancy in queries and results and difficulties to identify the optimal TTL level.

On the other hand structured P2P systems distribute data items over peers according to an algorithm and each peer is not free to choose what data items will store [2], [4], [5], [3]. The main idea in such systems is that peers and data items have unique identifiers and unique keys respectively produced by hash functions. Each node is responsible for storing files that have a key that is similar to the node identifier. Given a key, a node efficiently routes the query to the node with an ID close to the key. This presupposes a rather static network of peers. In contrast, P2P networks today are dynamic and the search functionality need is much more demanding than a single hashing function. Structured P2P systems need additional replication mechanisms to achieve availability.

The super-peer architecture described in detail by [23] is a hybrid P2P solution with interesting features. In such networks the differentiation of peers (super/simple) is introduced in contrast with the pure P2P systems where all peers take over equivalent roles in the network. A super peer keeps an index over its leaf nodes and is able to perform queries on behalf of leaves. The architecture enables also the direct exchange of information between any peer. Although the super peer architecture can promise improved performance in discovery tasks there are still many open research issues that need investigation like what is good ratio of leaves to super peer, how should super-peers connect to each other (unstructured vs. structured), how can search mechanisms be improved based on semantics and data-relationships, how the system can be more reliable introducing redundancy (K-redundant mechanism?) etc.

P2P systems mentioned above are not suitable in application areas where efficient data sharing is needed (e.g. Sharing semantic information in different databases in different peers). Many research results can be found in literature regarding the deployment of databases over a P2P network ([24], [25], [26], [27], [28] and others). In such P2P systems that are aiming to share rich semantic data the research challenges, among others, are effective schema mapping, efficient query execution over P2P, redundancy and consistency mechanisms.

Other approaches to P2P knowledge management are trying to exploit semantics (glossaries, complex metadata, taxonomies, and ontologies) in order to create appropriate indices that will facilitate the knowledge distribution and discovery. There is nowadays a clear trend in the community to move towards this direction and there are many research efforts undergoing.

The hierarchical summary indexing scheme proposed in [15] tries to build effective and compact indices (based on the glossary (list of terms) of each document in order to allow semantic-based knowledge

discovery in P2P systems. Taxonomy-based routing proposed in [16] is also a recent approach that tries to provide effective query routing based on semantic summaries of peer knowledge and indexing of these summaries in a distributed catalog at the super peer level based on DHT techniques. Metadata are also used for indexing purposes. The work presented in [17] describes a zero-administration P2P XML data base based on super peer model. The indexing scheme proposed is based on complex metadata (XML Schema) and a hierarchical approach is followed in order to build and maintain a global index.

In the last years there are also appearing approaches that use ontologies in order to build effective semantic indexing schemes for P2P knowledge management systems. Such approaches are presented in [8], [9], [10], [11], and [18] where ontology based indexing for P2P knowledge management infrastructures for the semantic web is proposed. Finally, Semantic Overlay Networks (SON) proposed in [14] is a way of organizing (logically) a knowledge aware network so that semantically related nodes are forming a semantic overlay network and queries are routed to the most appropriate SONs increasing the chances of quick matching.

Following our basic requirements and assumptions made for the design of the P2P Knowledge the DBE's approach deals with the existence of an unstructured p2p network (each peer supports the same p2p software and can directly communicate with its neighbors), the need for a decentralized schema management and the need for knowledge replication taking into account that a KB exploits as a persistency layer a native XML database.

2.4 Current Framework

This section describes the current framework approach to the P2P Knowledge Base. This framework is based on the functional and technical requirements mentioned in the previous sections as well as the technical assumptions made. As already noted the DBE Knowledge Base acts as a Knowledge Management Infrastructure (overlay) layer on top of the P2P communication network.

In the DBE environment a large number of SME nodes are pooled together to share their resources, information and services. The SMEs are sharing data that has rich structure and semantics, and thus advanced mechanisms for querying and discovering knowledge from this data are needed. The challenge in designing and developing the DBE KB is to enable the effective support of rich semantics, data transformation, data relationships, data constraints and support complex queries across multiple sources in the KB network since sharing of enormous amounts of data is not of benefit without advanced search mechanisms that will allow SMEs to quickly locate and use the desired information.

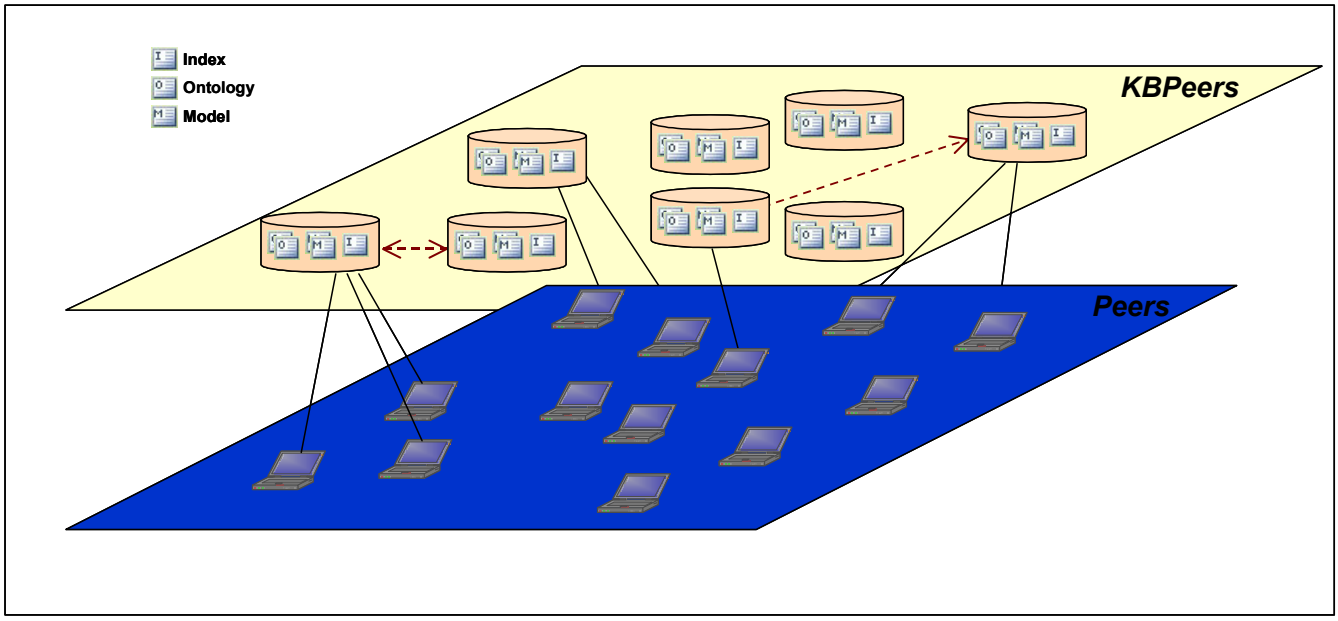

Figure 1: P2P KB Framework

The current framework (illustrated in Figure 1) follows the unstructured P2P paradigm. The P2P KB infrastructure deals with two types (layers) of peers:

1) **The KB peers** that have the appropriate infrastructure to host part of the DBE knowledge. On such a peer not only the local knowledge is kept, but also possibly replicated knowledge from other KB peers (or knowledge from peers that do not have KB infrastructure). The information regarding the replication knowledge hosted in the node along with the KB peers where replicas of the node's local knowledge are kept is maintained in a local catalog. Each KB peer is also assigned with indexing responsibilities. The local index structure that is kept in each KB peer is populated with semantic information that is used in order to improve query routing and content selection. The information kept is updated using the qualifying results for queries that are forwarded by the node to other nodes following the routing protocol provided by the underlying P2P network.
2) **The Simple Peers** that are peers that do not have KB infrastructure or they do not wish to share resources to the P2P KB. These peers will still have the ability to use services offered by KB Peers in order to store and access DBE knowledge.

The straightforward approach for query processing in such an unstructured network is to use a flooding mechanism to forward query requests to neighbor peers. While such an approach would be operational, there are many problems such as huge communication costs, redundancy of results, and high response times, that make it inapplicable. In the current framework the query flooding is used at the system bootstrap to enable the building of effective indexing in order to improve the efficiency of query processing. Semantic indices are being built in each node while the network is operational to narrow the query routing only to those peers that are more likely to qualify the query.

In such a route-learning scheme, a peer tries to estimate the most likely peers from which the replies to queries may come from. Peers organize this information into semantic indices based on the knowledge that is gradually built from queries that are routed through the peer and returned qualified results.

3. Dynamic Replication mechanisms

In general replication means that we store copies of a knowledge item. An entire knowledge item can be replicated at one or more sites. The motivation for replication is twofold:

- Increased data availability: if the node that contains a replica goes down, we can find the same data at other nodes. Similarly, if local copies of remote knowledge items are available, we are less vulnerable to failure of communication links.
- Optimized queries: queries can be executed faster by using a local copy instead of going to a remote site

In this section we focus on the approach followed to fulfill the first motivation.

3.1 Background issues

In order to ensure high availability the distributed implementation of the KB/SR requires extensive replication of the stored data. Although at the time being it is rather difficult to estimate the average size of the whole KB/SR storage it is safe to say that in order to ensure the system's scalability we have to consider the granularity of the knowledge replication at the level of an item (i.e. model and/or service manifest) instead of the entire contents of the KB/SR node.

The followed approach for the distributed KB/SR is that it should behave just like a centralized system from the point of view of a user; issues arising from distribution of the knowledge are transparent to the user, although of course, they have been addressed at the implementation level. With respect to queries this view means that the users are able to ask queries without worrying about how and where the knowledge data is stored.

With respect to replication two were the alternative approaches. In synchronous replication all copies of a modified knowledge item must be updated before the modifying transaction commits. In the asynchronous replication, the copies of the modified knowledge item are updated periodically and therefore a transaction that reads different copies of the same knowledge item may see different values. In addition, asynchronous replication comes in two flavors. In primary site asynchronous replication, one copy of the knowledge item is designated as the primary copy. Replicas of the entire item can be created at other nodes; these are the secondary copies and unlike the primary copy, they cannot be updated. A common mechanism for setting up primary and secondary copies is that the users first store the knowledge item (or publish the service manifest) at the primary node and subsequently copy (store/publish) it to the other selected nodes.

Various P2P applications consider asynchronous replication where more than one copy (although perhaps not all) can be designated as being updateable, that is, as a primary copy. In addition to propagating changes, a conflict resolution strategy must be used to deal with conflicting changes made at different sites.

In the current distributed implementation of the KB/SR we have followed an asynchronous-based primary-secondary replication approach where updating rights are held by only one primary copy at a time. Changes at the primary site are propagated to other sites and updates are actually not directly allowed at other sites. Nevertheless the update process is transparent. When there is a certified request for updating a replica item the request is transparently forwarded to the primary node (where the primary item is located), the update is taking place there and the modified item is again migrated to the replica nodes in an asynchronous manner. In order to handle possible failures of primary nodes we intent to provide (in the next implementation release) mechanisms that will support a backup procedure, i.e. to designate one node as a backup to another node. Following that, in the case that the primary node fails, the backup node takes over and updates are now permitted at (only) the backup site.

Keeping track of the knowledge that is distributed across several nodes can get complicated. If a knowledge item is replicated we must be able to uniquely identify each of its replicas. DBE does not use a global name-server to assign globally unique names (in order not to compromise the node autonomy). In order to

uniquely name a knowledge item we concatenate together two fields that during the item's lifetime are considered as a whole. These two fields are:

- A local_name field, which is the name assigned locally at the SME where the item is created. Two items at different SMEs could possibly have the same local name, but two items at a given SME cannot have the same local name.
- A SME_id field, which uniquely identifies the SME that created the item.

During the creation of an item the SME may choose the KB node that will act as the primary site and where information is maintained about all replicas of the item. Therefore a name for an item is of the form `item_name=SME_id+local_name` and after its creation cannot be decomposed. This name identifies the knowledge item uniquely. It is the same name of the item that is also used to identify its replicas. The assumption that an SME_id is unique guarantees that there will be no item that has the same name with a replica in the node where it is hosted.

In order to keep track of where replicas of a knowledge item are stored we follow an approach that preserves a local node's autonomy and is not vulnerable to a single-site failure. The approach is based on the one that was developed in the R* distributed database project, a successor to the System R project at IBM. Each node maintains a local catalog that includes all copies of items stored at that site. In addition the catalog at the original site for the item (primary node) is responsible for keeping track of where replicas of the item are stored whereas the catalog at the replica site also keeps information about the node where the primary item comes from. Whenever a new replica is created, or a replica is moved across nodes, the information in the original site catalog for the item must be updated. In order to locate a copy, the catalog at its primary node must be looked up. This catalog information for an item can be cached at other nodes (as a result to a query that contains the designated item) in order to improve access, but the cached information may become out-of-date if, for example, an item is moved or copied to additional nodes (during the repairing process that will be discussed later on). We will discover that the locally cached information is out-of-date when we use it to access the item, and at that point, we must update the cache by looking up the catalog at the primary site of the item (the primary site of the item is also recorded in each local cache that includes the item).

Availability of a replicated knowledge item is determined both by the number of replicas and the placement of the replicas. Merely increasing the degree of replication does not necessarily result in higher availability. For example, consider a scheme in which every new replica is created on the same node. That would increase the number of replicas, but if the node hosting all the replicas should fail, it would make the items unavailable. Similarly, if each new replica is placed on nodes that are vulnerable to failure of communication links, then adding new replicas does not improve the responsiveness of the system. Hence, in order to improve the availability of the system, placement decisions have to be considered when replicas are created dynamically, which is the case in a fully decentralized and unstructured P2P environment. The following decisions need to be made when creating a new replica:

1. When and how many replicas should be created?
2. Where should the new replica be placed?

In the following sections, we present the approach we have taken to address these issues.

3.2 General description of the approach

Each KB/SR node has a set of mechanisms that it can use in order to decide on the number of replicas, for any primary knowledge item contained in its storage space, which is needed in order to maintain the desired item's availability. Each node applies these mechanisms considering its knowledge about the status of the system at a specific point of time. In the current implementation this knowledge (that most probably will be incomplete) refers mainly to the information that the node has about the replication status of its primary knowledge items and could be obtained by utilizing its locally maintained catalog and a replica location service that can locate (and therefore provide the number of) the currently existing replicas for each one of these items. Moreover this knowledge is also augmented with information regarding the availability of hosts

in the system. The primary node uses this mechanism in order to track the availability of the nodes that store the replicas of its primary knowledge items. According to this information the primary node can dynamically adapt its mechanisms for efficiently maintaining the target level availability of its primary knowledge items (as will be explained later on). The knowledge about the status of the system can be extended with various other types of information about availability of its nodes. Possible types of information for a node may include (based on TCD's report [1]):

- Bandwidth and latency
- Processing performance
- Level of trust
- Number of connections
- Open IP addresses
- Willingness to share resources

3.2.1 Computing the number of replicas

There are several ways to determine the number of replicas that need to be created on demand. We can either create a constant number of replicas or choose the number adaptively based on feedback from the system. Our approach has been merely inspired by the work described in [31] and makes use of the probability of node failures as input to an analytical model to determine the number of replicas that need to be created to provide a certain degree of availability.

We consider the following parameters:

- Individual node stability p_i – the availability of an individual node *i* is governed not only by failures (node and communication ones), but also by user decisions to disconnect from the network. Here one should consider two cases. First is the case of a short-term availability that models the probability of transient disconnections that typically have no impact on the durability of the knowledge stored on the disconnected nodes. Second is the case of non-transient failures that result to the removal of the knowledge for indefinite intervals. In short, this parameter is expressed as the average probability of a node that can store a replica being up. Considering as T_i the total time that node *i* is being up[2] and T_{all} as the total time that the overall DBE system is running, then $p_i = \frac{T_i}{T_{all}}$.
- Target level of availability *A* – that represents the probability that a knowledge item can be accessed at any time
- Total number of replicas *c* for a specific knowledge item

Considering a node *i*, then the probability of the node being unavailable is $(1- p_i)$. In order to guarantee desired availability A for a specific knowledge item that will be replicated in *c* replicas we follow an incremental approach (by counting the product terms) so that

$$(1-\prod_{i=1}^{c}(1-p_i)) \geq A \qquad (1)$$

Intuitively, we calculate the probability that all the nodes up to the number of *c* will be unavailable. We consider that the values of A and p_i are known from the beginning.

For example, considering five nodes with stability factors as shown in the next table (where for example a stability factor 0.5 for a node equals the fact that probability *p* of the specific node to be up is 50%) then a

[2] In the current implementation we are actually counting the time that the KB/SR service that is hosted in the node is alive and we are assuming that for all this time the node is accessible.

sequentially incremental approach with the indicative replication factor can guarantee the designated target availability:

Node	Stability Factor p	Replication Factor c	Target Availability A
P1	0.8	1	0.8
P2	0.5	2	0.9
P3	0.3	3	0.93
P4	0.7	4	0.98
P5	0.6	5	0.99

Table 1: Computing number of replicas

Therefore considering that the values of A and p's are known, we can assume that 5 replicas of each primary item can guarantee a target availability of 99%. Again we have to say that in this example we have followed a sequential incremental approach. Other approaches could be more effective. For example a replication factor of 2 considering nodes P1 and P4 can guarantee a target availability of 94% comparing to the 90% that is obtained by selecting P1 and P2.

As already mentioned this analysis considers short-term availability – the probability that at a given instant there is sufficient redundancy to mask transient disconnections and failures. For the case of non-transient failures the system must repair the lost redundancy by dynamically store additional redundant knowledge at new nodes. The two parameters in repairing file data are the degree of redundancy used and the frequency that the system detects and reacts to node departures. [32] considers two repair policies: eager and lazy. Eager repair uses a smaller degree of redundancy to maintain file availability guarantees by reacting to node departures immediately, but at the cost of additional communication overhead. In contrast, lazy repair uses additional redundancy and therefore additional storage overhead, to delay repair and thereby reduce communication overhead.

Next we present the mechanisms elaborated by DBE in order to support replication and repairing.

First, each primary node uses a mechanism (based on its local catalog) that updates (if needed) the availability of the nodes that store the replicas for those knowledge items for which it is the primary node based on the information provided by the nodes themselves. The KB/SR service that is hosted in each one of these nodes is responsible for checking that the availability information provided by the node is the correct one (i.e. by calculating/keeping itself the stability factor p for the node)[3]. The time period (we call it Watch_Period) for this process is parameterized and normally it is rather short. Based upon individual node availability, this mechanism also computes a metric that measures the target availability for a specific knowledge item considering all tracked nodes that were available at a given time in a specific (parameterized) maintenance time interval (we call it Maintenance_Period). For example considering a 48 hours time maintenance interval and the specific time 12pm the metric may equal to 0.5, i.e. target availability 50%. The two time periods used are not globally set, i.e. each node is allowed to determine their values.

Considering the quite small average size of the DBE knowledge items (e.g. on average 30KB for a model and 50KB for a Service Manifest) we have chosen a pure replication approach, i.e. we keep full secondary copies for a primary knowledge item. The replication mechanism is activated when a new primary knowledge item is created/stored. During this process the mechanism uses the equation (1) to determine the number of replicas that it has to create. For that it uses the availability threshold associated with the item (this is considered as a known parameter that for most cases it would be desirable to tend to 1) and the stability factor (s) of candidate nodes (the way that a node is selected as a candidate is explained later on).

[3] Since DBE is considered as an open and untrusted environment a separate mechanism is needed to assure that the stability factor calculated for the node corresponds to its actual status. The DBE Decentralized Identity Service will provide this mechanism.

The way equation (1) is used and evaluated is also followed during the repairing phase. In DBE we have chosen a short-term dynamic repair mechanism. For a given Maintenance_Period, the primary node is using the information that is currently available in its local catalog to recalculate the availability of a knowledge item. Here we consider a few cases. First, is the case that the stability factor information of the replica nodes has not changed. In this case, it uses a replica location service that can locate (and therefore provide the number of) the currently existing replicas for this item. If any of the nodes is not available the primary node knows that it has to create additional copies of the item and migrate them to additional nodes (after locating them). If all the nodes are available no further action is needed. The second case under consideration is the one where the stability factor information of any of the replica nodes has changed. Based on the new values the primary node will decide (again using the equation) whether it needs to create additional replicas or not.

For example considering that the nodes P1, P2, P3 along with their respective stability factor information indicated in Table 1 are the replica nodes that correspond to the catalog information of a primary node for one of its knowledge items. That means that for the given item the replication process (as described previously) has created 3 replicas in order to guarantee a target availability threshold of 93%. Considering that within the maintenance interval the stability factor of node P1 is reduced to 0.4 then the mechanism will calculate that the availability level of the item has been also reduced to 0.79. Assuming that the value is below the desired threshold, this will dynamically trigger the repairing process that will try to locate additional nodes to migrate new copies of the item. Saying that the selection process (that will be discussed in the next section) returns the nodes P4, P5 as possible candidates, then applying equation (1) utilizing also the stability factor of P4 results to target availability of 0.93 and the repairing process ends. It is obvious that in the case that the stability factors of the replica nodes were increased there is no need to trigger the repairing process since the target availability will be also increased.

3.2.2 Replica Placement

As mentioned earlier, it is important to locate the replicas appropriately, in order to make the best use of replication. There are several ways to select the node that will host a new replica. The simplest way is to choose the node at random. However, if the goal of creating a replica is to improve the responsiveness for the users, then choosing the placement becomes a more interesting and challenging problem. One possible approach is to monitor the access patterns of the clients and locate the replica in close proximity to the clients that access the service more frequently. Alternatively, the new replica may be located near the clients that have most stringent time constraints. In a wide-area network in which some of the links can potentially become slow or congested, it would be more appropriate to use a dynamic placement scheme that uses feedback from the system to guide the location, so that we can avoid placing the replicas on bottleneck links.

In our approach, the replica placement mechanism first provides a set of nodes as candidates for storing the new replicas. The criteria followed for the candidate's selection are:

- A candidate node should not already contain a copy of the item that is to be replicated
- A candidate node should have available storage (or at least it is willing to replace an existing replica)
- A candidate node should be "close" to the primary peer in order to have a reasonable transfer time (below a certain maximum)
- A candidate node should have a stability factor greater that an indicative factor

It is obvious, that the approach is not limited to the aforementioned placement criteria. Other criteria (some of them have already mentioned before) can be also utilized. Alternatively (as discussed in [1]) a node utility (a weighted sum of the various parameters) can be calculated for each node. In this case candidate nodes would be the ones that have a utility greater than a specific threshold.

Next we present the replica placement mechanism.

To place a new replica for one of its primary knowledge items, the primary node first uses the mechanism to query the system in order to identify candidate nodes that qualify against the storage and stability factor criteria that mentioned above. It has to be noted that the stability factor threshold of the query has not been globally set, instead the primary node is allowed to determine it's value. The storage threshold depends on

the size of the replicated item. Following the communication mechanisms provided by the overlay network (in the current servent implementation it is based on pure flooding-based communication) the query returns as a result a list of qualifying nodes where the additional criterion for not already containing a replica of the item has been also satisfied.

Once the candidates are identified the mechanism should select the best of them up to the desired number of replicas. One solution could be to consider the least loaded peers. Other heuristics could presume system information provided by the p2p network (e.g. network delays, bandwidth, and topology) or the peer itself (e.g. processing performance, reputation). In the current implementation the obtained list is ordered following the proximity criterion, i.e. the "closer" nodes are on the top of the list.

3.2.3 Updates & consistency

Taking into account that DBE is an untrusted environment and that the KB/SR data is distributed among the various DBE nodes it should be ensured that update operations to the KB/SR knowledge items are authenticated, i.e. a node should be certified that it has the privileges to update the knowledge item.

Following the assumption that in the DBE environment we do not have frequent updates, in the current implementation we have followed the simplest design for handling updates. That is, a modify/delete operation of a knowledge item in the KB/SR can take place only on the primary copy of the item. Nevertheless for the user point of view this constraint is transparent. If an update is directed to a replica of the knowledge item, this update is transparently forwarded to the primary node and then it is the primary node itself that is responsible for accepting the update and propagating the updated item to all replicas (based on the information kept in its catalog).

Considering that the DBE will also support a distributed identity service the current implementation can be also revised in order to support updates on any of the replicas of a knowledge item. Following the DBE case where each SME is the owner of a specific knowledge item and using the provided authentication mechanisms it can be ensured that there will be no replica divergence (i.e. writing conflicts). However in this case additional mechanisms should be provided in order to support consistency among the replicas. This also holds for the case where a replica or even the primary peer has to synchronize after a transient failure. The current framework for a replica peer can be easily extended so that the information kept in its local replica catalogue to include information not only about the primary peer for each one of the replicated items that it hosts but also about the other peers that hold a replica of this item. In this way the updated information about new/modified/deleted knowledge items could be propagated between these peers. The peer that accepts the update request could originate this process. These mechanisms will be considered in the next implementation. In addition, more ambitious scenarios could be supported, i.e. allowing updates on an item not only by its owner SME but also from another SME (assuming there is some kind of affiliation between these SMEs).

4. The KB Infrastructure and Common Components

The DBE Knowledge Base is deployed across the SME nodes that 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 while designing the KB infrastructure was to make possible to install a basic KB infrastructure to 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 in mind 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 and wishes to host DBE core services should have a local basic KB infrastructure installed.

The KB infrastructure has evolved from the first version and now incorporates a native XML Database, exploits a MOF metadata repository system and provides all the necessary software components to support the execution of the DBE Infrastructural services: Knowledge Base (KB-Service), Semantic Registry (SR-Service) and Recommender (RC-Service). These infrastructural services form a P2P network and cooperate to offer an effective way to discover, describe, share, analyze, 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 organizations 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 infrastructure are the following:

- The KB supports OMG standards for modeling and model interchange (standards for information organization and interchange) (MOF 1.4, XMI 1.2). The adoption of existing standards, ensure the compliance with industry directions and enables the sharing of models among various data modeling tools.
- The KB extends easily to (accommodate) other information model and sources. The KB includes multiple metamodels for Ontologies, Businesses and Services. Additional metamodels, that are going to be used in the future, can easily be implemented supported and managed. The KB enables different business users to model the enterprise and the services they offer, as well as use, integrate or modify existing models.
- The KB could support different information views. In other words, different access levels to the same metadata from different perspectives.
- The infrastructural services (KB/SR/RC) are designed and implemented entirely for the Java Platform and are distributed with the DBE Servent. The KB infrastructure can be deployed across SMEs regardless of existing operating systems. It is compliant with the Java Community Process interface standards and the existing XML standards.

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, accounting/metering data)

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) and is exploiting as a persistency layer a powerful native XML Database (Berkeley DB XML).

The KB infrastructure is able to fully support the currently available DBE metamodels Ontology Definition Metamodel (ODM) [33], Business Modelling Language (BML) [40], Semantic Service Language (SSL) [33], and Service Description Language (SDL) [41] and provides basic functionality for processing, storing and retrieving models following the above metamodels.

Figure 2 shows the architecture of the KB infrastructure. The KB is compliant with the JMI 1.0 specification that is the result of a Java Community Process (JCP) effort to develop a standard Java API for metadata access and management. The advantages to comply 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.

The KB infrastructure provides a common persistence and knowledge management layer in the digital ecosystem. It offers a set of APIs and tools for accessing the DBE Knowledge (M1 Model information and M0 data). The components that comprise the basic KB infrastructure are described briefly in the following of this section:

- The Peer Manager takes over the management of the peer (KB or SR or RC), lying in an SME, and is responsible for the P2P cooperation across the KB peers. The Peer Manager keeps statistics regarding the operational time and network connectivity and periodically calculates estimation for the independent stability of the peer. Such information is useful for the P2P and replication mechanisms applied.
- The Model Manager integrates the metadata repository 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, optimization and query mechanisms of the underlying native XML 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, and extracts useful information. Currently the Model manager is capable of processing models that comply to the following metamodels: the ODM (Ontology Definition Metamodel), BML (Business Modelling Language), SSL (Semantic Service Metamodel), SDL (Service Description Metamodel), QML (Query Metamodel)

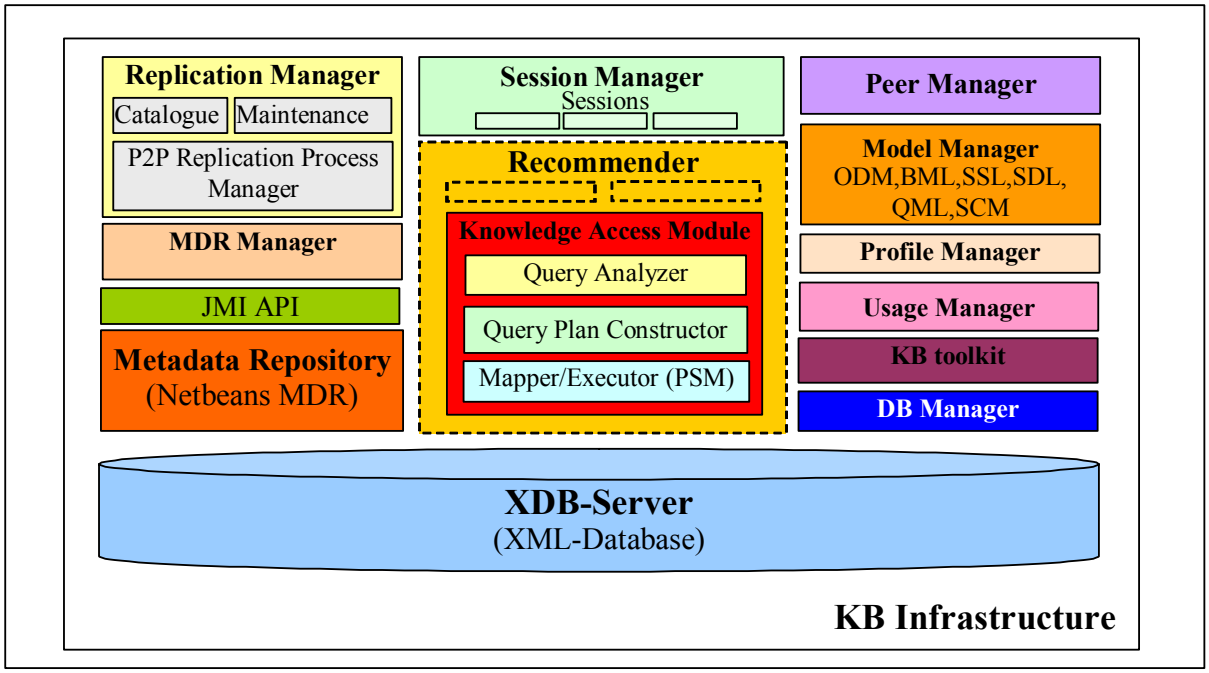

Figure 2: the architecture of the KB infrastructure

- The Usage Manager handles the service usage data of DBE services and exports APIs to be used by the Metering infrastructural services for storing data usage records into the database. This data may be used in the future to automatically create new or adapt existing SME profiles for improving system recommendations
- The Profile Manager manages the storage, retrieval and updates of the SME profiles in the database. The profiles are represented as XMI documents following the User Profile Metamodel (UPM)
- The Recommender Module implements the components related to the evaluation of candidate services and candidate business partners in the process of discovering or composing services and establishing partnerships respectively. The Recommender Module 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 Profile Manager to store preferences and all recommendation mechanisms operate on top of the native XML database to implement the necessary matching functions between preferences and business/service descriptions.
- The Query Engine is a component of the Recommender and takes over the processing of queries submitted to KB. A Query Metamodel (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 utilizing the JMI reflective functionality. These queries will be streamed to the server (encoded as XMI 1.2 documents) and are stored into the metadata repository. The Query Engine is comprised by the Query Analyzer, the Query Evaluator and the Query Mapper/Executor. The Query Analyzer interprets and analyzes the query model using the JMI interfaces of the Metadata Repository. The Query Evaluator evaluates the query and constructs a valid query syntax tree, and the Mapper/Executor module accepts as input the query syntax tree produces suitable XQuery statements and executes them to the data management system.

- The Replication Manager is the component that manipulates the replication mechanisms in each peer (calculating the number and handling the placement of the replicas) in order to ensure high availability in knowledge access
- The MDR Manager provides functionality for managing the Metadata Repository
- The KB toolkit contains useful software components and export APIs for Query Formulation, Service Manifest processing and others.
- The DB manager administrates the XML DBMS and exports database specific functionality that can be used by other modules in KB.
- The XDB-Server deploys the XML Database as an independent network server that can support access from multiple services

For describing instances of models in the KB an Instantiation MOF model has been developed and is used in the current implementation to support the descriptions of instances of the BML/SSL models. The model instances are being streamed to the KB service as XMI1.2 documents in order to be processed in the KB and finally stored in the database. Similar approach will be followed also for model data of other metamodels in the case where M0 data exist for models.

4.1 The Core Services supported by the KB infrastucture

On top of the KB basic infrastructure are running the DBE core services related to the Knowledge Base: the KB Service, the Semantic Registry Service and the Recommender Service. 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, the KB Service or the RC Service respectively. These DBE core services export the functionality that can be supported by the underlying infrastructure components.

As far as model information is concerned the KB, SR and RC services are JMI-enabled. The KB is the central metadata store for the DBE system.

In Figure 3 is presented the deployment of a KB service inside the DBE ServENT in a SME peer. 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 specialized business knowledge features. The KB service accepts through network connections XMI 1.2 documents that represent DBE models. These documents are streamed to the metadata repository, are being processed by the Model Manager and are finally stored in the database.

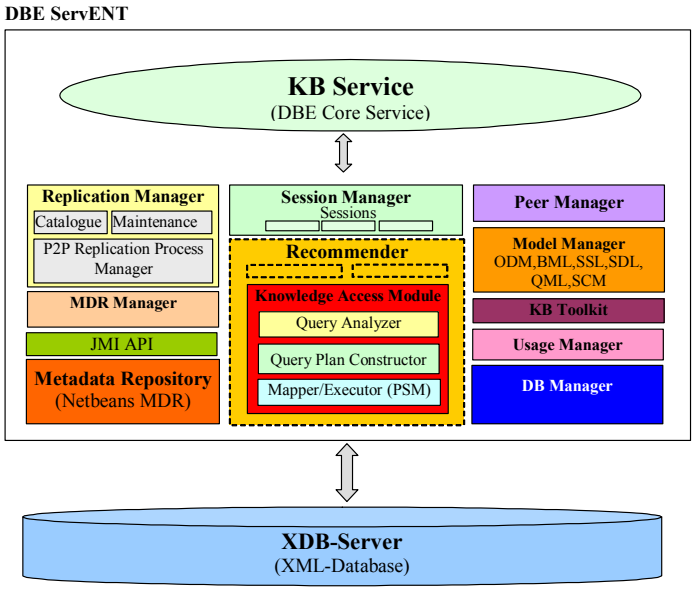

Figure 3: The Knowledge Base Service (KB-Service)

Figure 4 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 others are bundled together in an XML document constructing the Manifest of an SME service offering.

The Service Manifest (SM) is published in the SR service where is being processed and the information extracted is properly organized in the database enabling effective semantic discovery tasks. The SR service contains an additional component the SM toolkit (inside the KB toolkit). The SM toolkit provides functionality for handling the SM XML documents. This component exports APIs for creating, updating and extracting particular information elements from the SM documents. The SM toolkit is used by the SR service to disassembly an incoming SM document into its parts. As mentioned above these parts (among optional others) constitute the BML, SSL, SDL models and BML and SSL instances and are represented as autonomous XMI documents within the original service manifest. These XMI documents are processed by the Model manager to extract useful information and are stored in the database.

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

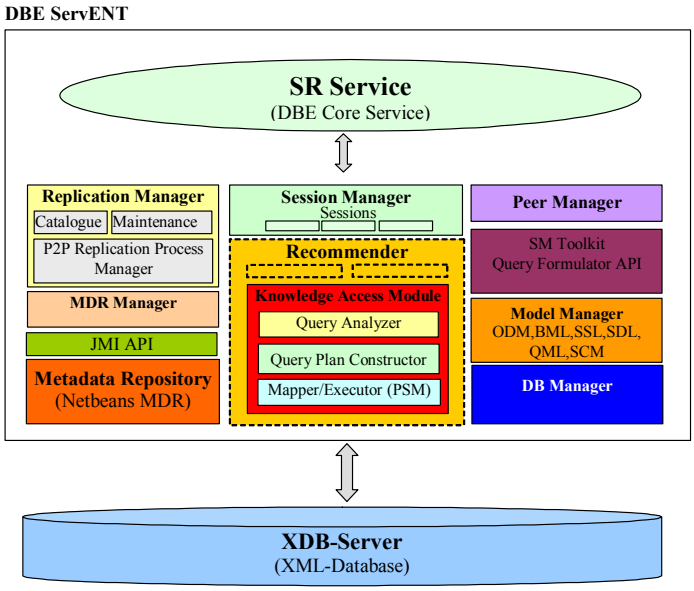

Figure 4: The Semantic Registry Service (SR-Service)

The Recommender Service acts as an autonomous process that manages SME preferences (either business preferences or service preferences) and matches these preferences with available business descriptions and service descriptions. The Recommender Service exploits the matching mechanisms of the Recommender module of the KB infrastructure. The Profile Manager searches in the database for stored SME profiles that are active and creates a new session for each profile. Each session takes over the process of searching in the P2P network of Semantic Registries for published services that match the preferences of the specific profile. The results are aggregated and processed per session and are available each time a recommendation based on a profile is requested or the SR service can automatically trigger a notification to the interested user whenever there is a change in the recommendations due to new service publishing or due to updates of existing services.

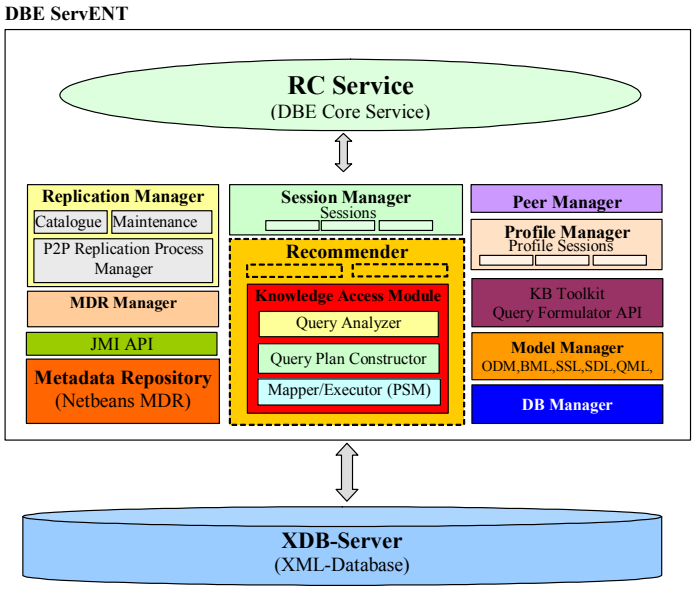

Figure 5: The Recommender Service (RC-Service)

5. Knowledge discovery and replication mechanisms in the DBE P2P KB network

In the DBE environment a large number of SME nodes will be pooled together to share their resources, information and services while keeping themselves fully autonomous.

The challenge in designing and developing the DBE KB is to enable the effective support of rich semantics and complex queries across multiple sources in an unstructured and self-organized P2P network such as the DBE network. The objective of the first P2P implementation of the DBE KB was to exploit the current P2P infrastructure of the ExE environment, and to offer high availability in knowledge access and an effective distributed knowledge discovery mechanism.

The Core Services (KB/SR/RC), running in multiple peers, will have to cooperate to help the semantic discovery process. The P2P communication among different service instances is done using the server-client P2P container offered in the DBE (DBE ServENT) by its sourceforge Swallow project[4]. The queries in the current implementation have the following characteristics:

- The queries in the DBE KB are expressed as instances of the QML metamodel and are represented as XMI documents
- Each query has a TTL value and is forwarded between peers until TTL reaches to 0.
- Each query message has a unique identifier. This is necessary to avoid the routing loops in an unstructured P2P network.
- A peer can send the qualifying results back to the origin peer (using the appropriate query identifier)
- The query result set contains the references to all the resources that matched the query in the replying peers.

The discovery of resources in unstructured P2P networks is usually performed using a flooding mechanism. In flooding, each node receiving the query forwards the query to all of its neighbours. Although it is simple and robust this approach, causes too much bandwidth to be wasted, because most of the time the query is forwarded towards neighbors from where no answer returns. Intelligent routing decision allows a node to forward a query only to a subset of peers. There exists a substantial amount of work on reducing query overhead in an unstructured P2P network. We followed an approach that will utilize the semantics and ontologies used in the content that is shared. We just require each peer to learn about the semantics of the resources managed at other peers in the network by only monitoring the query and query results sent to or received from their neighbors and other peers.

In the bootstrap process the core services are using the flooding mechanism to forward queries to the other peers in the network and discover new sources. While more and more queries are issued in the DBE KB network appropriate semantic indexes are build in each node using a learning mechanism and the KB network becomes more efficient. The applied distributed discovery mechanism is described in [38]. The process of distributed discovery is presented graphically in Figure 6. When a discovery request is issued in a KB or SR service the Session Manager triggers the creation of a new session that is identified by a unique identifier (SID) and accepts the XMI representation of the query. The Session Manager maintains a hash table called session table and stores the session identifiers of the queries that have received. A peer receiving a query extracts the identifier (SID) and searches that SID in the session table. If the SID is previously processed, which means that an entry with the same SID appears in the table, then the query is ignored. In this way the distributed KB network becomes free of routing loops. After checking the query in the session table, the new session analyzes the query with the support of the Knowledge Access module and utilizes the Semantic Index to identify the most appropriate peers that could answer the query. The peer forwards the query to these peers that are likely to contain knowledge items semantically related to the query. Concurrently with the query forwarding, the peer checks its shared resources for a local match

[4] http://swallow.sourceforge.net

(Recommender module). If a peer holds content that match the query it replies with a result set that contains knowledge item references and ranking values. The result set is also augmented with a list of the peer ids that host a replica for each one of the qualifying knowledge items. The qualified results that are received from the replying nodes together with the local results are processed and aggregated in one result set at the session running in the origin peer, while at the same time the set is sorted and redundant references are eliminated. Further work is needed here in order to improve the access using the replica information provided in the result. In addition to query routing based on the Semantic Index, the flooding can be used as a mechanism to discover new sources or capture updates in the KB network.

In order to access a specific knowledge item the peer performs a service call, to the peer indicated in the resource reference returned in the result set, and the knowledge item is retrieved.

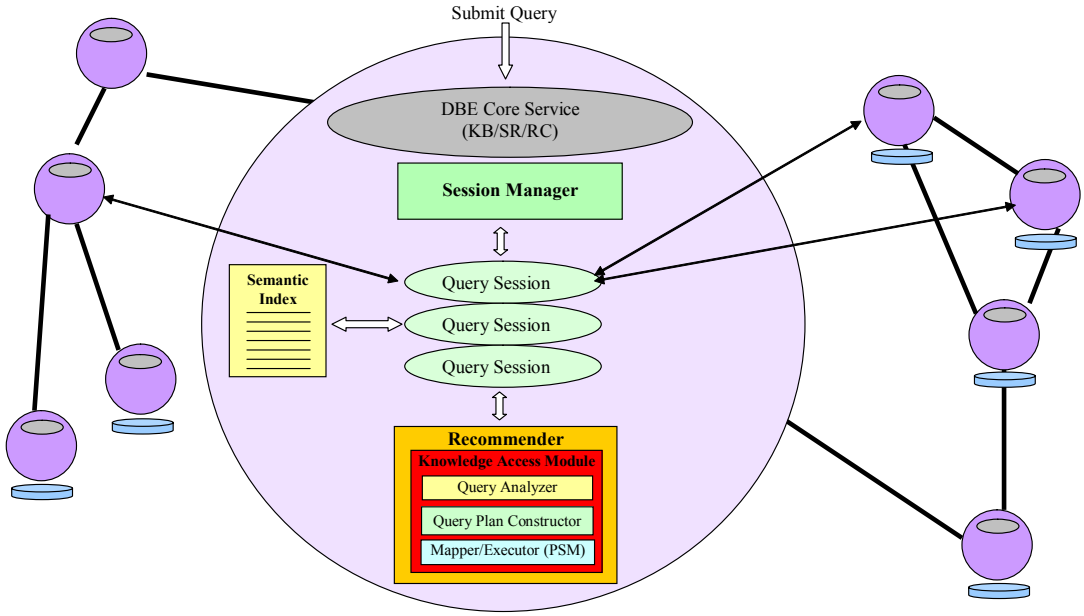

Figure 6: Discovery in the P2P Network

The objective of high availability is covered by the application of the replication mechanism that is presented in section 3. Each peer running a core service (KB/SR/RC) is able to calculate an independent stability factor. The stability estimation is done by the Peer Manager and is based on statistics of total operation time and network connectivity. At the service start up the Replication Manager submits a query in the KB network searching for nodes with an acceptable stability level that are willing to share storage resources of specific volume[5]. The Replication Manager collects the replies from various peers and then processes the results and constructs a list of candidate replica peers. Based on this list the Replication Manager identifies the peers that are needed to store information replicas in order to achieve a certain availability level based on the formula presented in subsection 3.2. After the identification of replication peers new Replication Threads are fired for each replica peer that are responsible for replicating information to the respective peers. When a storage request is accepted by the core service the document is processed and is stored in the local database system. At the same time the document is also registered in the Replication Manager as a pending item for replication. The Replication Threads periodically ask the Replication Manager for pending documents and forward them to the replica peers for storage. Also, the Replication Threads are checking periodically the network accessibility of the respective nodes and adapt the peer stability factor to reflect the real behavior of the peer. In this way malicious and un-trusted peers are identified and ignored while at the same time any change in the reliability of replication peers is captured.

[5] TCD in [1] proposes to direct the query to the more reliable neighbours and in this way all peers storing replicas will be clustered in the highly connected core. The current implementation could be used for that assuming that the neighbour peers can be identified.

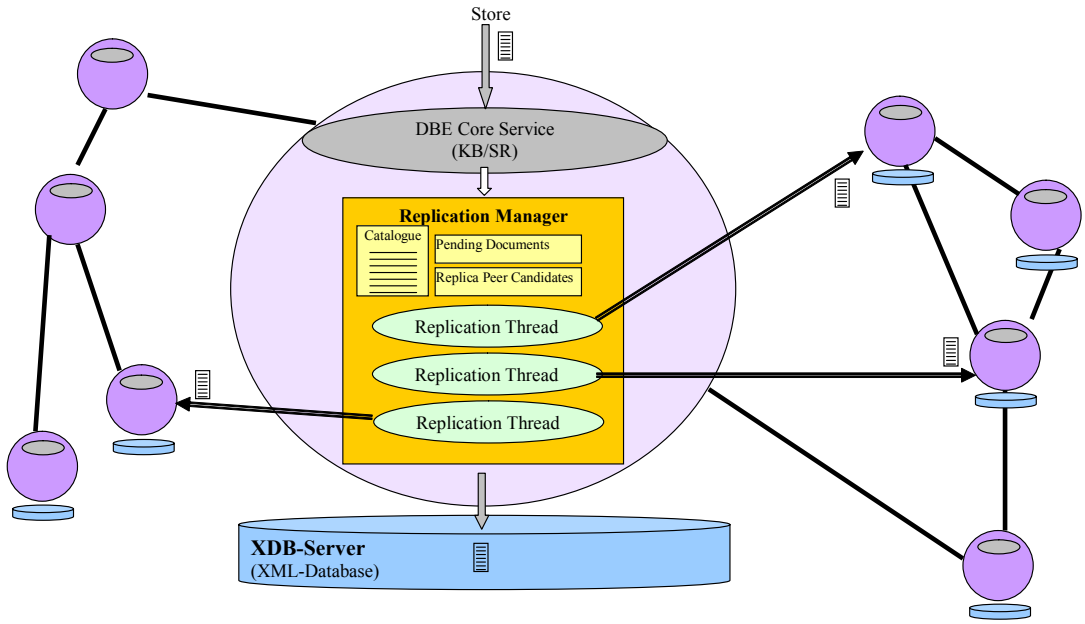

Figure 7: The Replication Manager of the DBE Core Services

The Maintenance module of the Replication Manager is activated periodically and based on the current peer stability factors recalculates the availability level offered by the current replication scheme. If the availability level drops below the acceptable level the Replication Manager searches the list of candidates replica peers and fires new Replication Peers until the desired availability level is reached again. In the case where no more candidate peers can be used, the Replication Manager repeats the procedure of discovering new replica nodes by issuing the appropriate query to the KB network.

6. Deployment of DBE Core Services (KB/SR/RC Services)

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

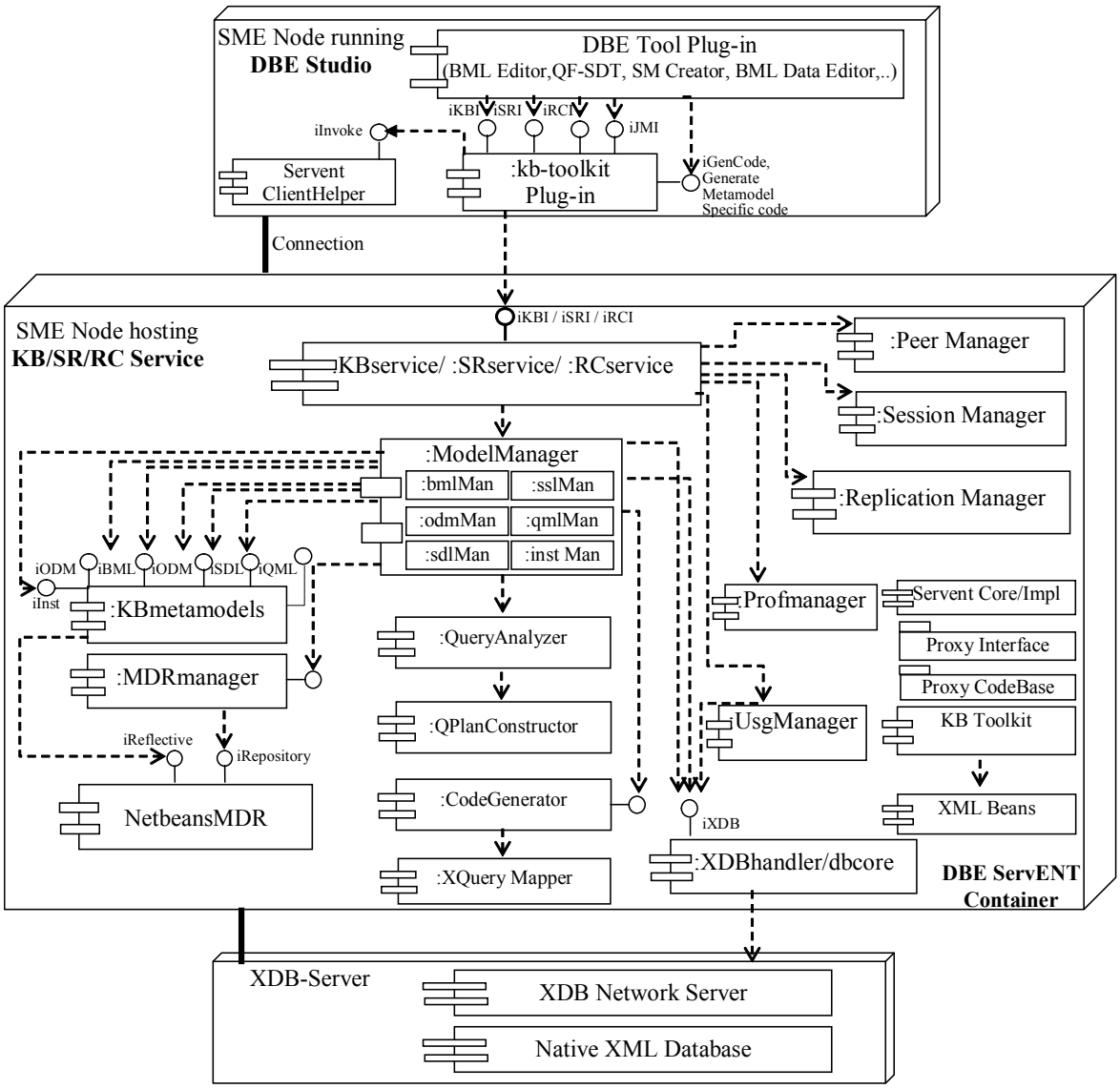

Figure 8: Core Services (KB/SR/RC) deployment diagram

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. The Recommender service aims in contributing in both environments providing useful recommendations for models and services according to defined preferences.

The KB/SR/RC core services are available in the DBE Sourceforge Swallow project and are distributed together with the main Exe installer that contains the DBE ServENT.

7. Conclusions, Challenges and Objectives

There are many challenges that had to be met in the design and implementation of the aforementioned framework. Although many of them are referring to the creation, maintenance, and exploitation of effective semantic-based indices that will support efficient query processing in the P2P KB, there are also others that are also open research issues in P2P knowledge management area. Such challenges refer to ontology storage and maintenance, to support of mappings between domain and local ontologies, to knowledge distribution and replication strategies, etc. In what follows we shortly discuss some of them identifying actually the research objectives of the Knowledge Base Work Package (14) with respect to P2P design and implementation that will be further considered until the end of the project.

The current framework indicates an initial approach to the P2P Knowledge Base. As the project advances, this framework needs to be elaborated in depth in order for its various parts to be further analyzed. As mentioned, such a detailed analysis and design process will be feasible only when various functional and viability (sustainability) requirements of the project will be elicited, elaborated, and documented. The following challenges among others are important for the design and development of the next release of the P2P Knowledge Base:

1. **Ontology Storage and Maintenance:** Ontology storage and maintenance is a major issue in a P2P environment. There are two types of ontologies in the P2P KB: Domain ontologies and user ontologies. While user ontologies could be stored in the user's KB Peer, the domain ontologies need to be known and accessible by every one. They also need to be replicated in more than one nodes of the P2P KB. On the other hand, the dependencies that models and (thus) SMs have with ontologies (models refer ontology concepts) needs also to be taken into account in the design of the ontology storage and replication in the ontology repository component of the KB in order to facilitate various other functions of the system (e.g. instantiation of a model). Ontologies on the other hand, are already being used for indexing purposes; this means that the KB peer network should necessarily host the domain ontologies. Finally, although updates on ontologies will be relatively rare, maintaining consistency is clearly an important issue.
2. **Semantic Indexing:** As discussed above and in [38], indexing is fundamental in the proposed framework for the P2P Knowledge Base. Taking into account the functional and technical requirements reported in the previous sections, semantic-based indexing is the most appropriate for the DBE P2P Knowledge Base. The challenge in developing the P2P KB is exploit DBE ontologies and languages to enable effective semantic indexing schemes. Moreover, the adoption of MOF metadata architecture, which imposes by default a hierarchical data and metadata organization, has already been proved valuable in designing and developing an effective semantic indexing technique.
3. **Knowledge Discovery, Distribution, and Replication:** Depending on the developed route learning semantic indexing scheme complex query processing mechanisms of the P2P KB could be further supported. Semantic indexing is not only used for knowledge discovery, but also for knowledge distribution and network organization as shown earlier. So, more appropriate strategies for semantic-based knowledge distribution and mechanisms for knowledge replication should be further investigated. At this time point it is by no means clear that the proposed redundancy management approach will work effectively in practice.

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".
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
OMG	Object Management Group: International standardisation body
P2P	Peer-To-Peer
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.
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
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.
XML Database	A database that is specialized for storing XML data and stores all components of the XML model intact. It has an XML document as its fundamental unit of (logical) storage and it is not required to have any particular underlying physical storage model (e.g. it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files)
XQuery	A Query language from the W3C that is designed to query collections of XML data.

9. References

1. Bartosz Biskupski, Jan Sacha, Jim Dowling, And Edmonds, Jean-Marc Seigneur: Peer-to-Peer architecture for DBE, release 0.5, July 2005
2. Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan , "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications'", IEEE/ACM Transactions on Networking, Vol. 11, No. 1, pp. 17-32, February 2003
3. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In Proc. ACM SIGCOMM 2001, August 2001
4. A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems," Proc. 18th IFIP/ACM Int'l. Conf. Distributed Systems Platforms, 2001
5. Zhao, B.Y., Kubiatowicz, J., and Joseph, A.D., "Tapestry: an infrastructure for fault-tolerant wide-area location and routing", Technical report, University of California, Berkeley, 2001
6. Sun Microsystems: "Project JXTA 2.0 Super-Peer Virtual Network, http://www.jxta.org/ project/www/docs/JXTA2.0protocols1.pdf
7. Sun Microsystems: "Core FADA", http://fada.jini.org/docs/CoreFADA.pdf
8. Nejdl et al. EDUTELLA: a P2P networking infrastructure based on RDF. In Proc. of the International World Wide Web Conference 2002 (WWW2002), pages 604–615, Honolulu, Hawaii, USA, May 2002.
9. Daniel Elenius, Magnus Ingmarsson: "Ontology-based Service Discovery in P2P Networks". In the proc. of the 1st international workshop in P2P Knowledge Management, P2PKM 2004.
10. Verma, K., et. Al.: "METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services", Journal of Information Technology and Management
11. A. Castano, et. al.: "Ontology-Addressable Contents in P2P Networks", In the proceedings of the 1st Workshop on Semantics in Peer-to-Peer and Grid Computing, Budapest, 2003
12. Clarke, I., Sandberg, O., Wiley, B., and Hong, T. Freenet: A distributed anonymous information storage and retrieval system. In Proceedings of ICSI Workshop on Design Issues in Anonymity and Unobservability. Berkeley, California (June 2000); freenet.sourceforge.net
13. C. Sangpachatanaruk, T. Znati: "Semantic Driven Hashing (SDH): An Ontology-Based Search Scheme for the Semantic Aware Network (SA Net)". In the Proc. of the IEEE International Conference on Peer-to-Peer Computing 2004: p. 270-271
14. Garcia-Molina, Hector; Crespo, Arturo. Semantic Overlay Networks for P2P Systems, http://dbpubs.stanford.edu/pub/2003-75
15. Heng Tao Shen, Yanfeng Shu, Bei Yu: "Efficient Semantic-Based Content Search in P2P Networks", IEEE Transactions on Knowledge and Data Engineering, July 2004 (Vol. 16, No. 7), pp. 813-826
16. Alexander Löser: "Towards Taxonomy based Routing in P2P Networks",
    The Second Workshop on Semantics in Peer-to-Peer and Grid Computing at the Thirteenth International World Wide Web Conference, 2004
17. Carlo Sartiani, Paolo Manghi, Giorgio Ghelli, Giovanni Conforti: "XPeer: A Self-Organizing XML P2P Database System", EDBT Workshops 2004: 456-465
18. Wolfgang Nejdl, et. al.: "Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks", ACM International Conference on World Wide Web, WWW2003, May 2003.
19. Efficient Search in Peer-to-peer Networks. Beverly Yang and Hector Garcia-Molina. In ICDCS, 2002
20. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks", International Conference on Supercomputing (ICS'02). ACM, 2002.
21. V. Kalogeraki, D. Gunopulos and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks", Eleventh International Conference on Information and Knowledge Management (CIKM), McLean, VA, November 2002
22. D. Menascie, L. Kanchanapalli, "Probabilistic Scalable P2P Resource Location Services," ACM Sigmetrics Performance Evaluation Review, Vol. 30, No. 2, September 2002
23. Yang, Beverly; Garcia-Molina, Hector, "Designing a Super-peer Network", 19th International Conference on Data Engineering, ICDE 2003
24. Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan and Ao Ying Zhou "PeerDB: A P2P-based System for Distributed Data Sharing". 19th International Conference on Data Engineering, ICDE 2003

25. Query Processing Over Peer-To-Peer Data Sharing Systems, Ozgur.D. Sahin, A. Gupta, D. Agrawal, A. El Abbadi, UCSB Technical Report, (2002-28)
26. Halevy, A.Y., Ives, Z.G., Mork, P., Tatarinov, I.: "Piazza: data management infrastructure for semantic web applications", Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, 20-24 May 2003, ACM (2003) 556–567
27. Vasiliki Kantere, Iluju Kiringa, John Mylopoulos, Anastasios Kementsietsidis, Marcelo Arenas. "Coordinating Peer Databases Using ECA Rules" International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), September 2003.
28. Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller, John Mylopoulos. "The Hyperion Project: From Data Integration to Data Coordination". In SIGMOD Record, Special Issue on Peer-to-Peer Data Management, 32(3):53-58, 2003
29. Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt, "C-OWL: Contextualizing ontologies", In Proceedings of the 2nd International Semantic Web Conference (ISWC2003), 20-23 October 2003
30. Chunqiang Tang, Zhichen Xu, Mallik Mahalingam, "PeerSearch: Efficient Information Retrieval in Peer-to-Peer Networks", HP Internet Systems and Storage Laboratory, Internal Report, 2003, http://www.hpl.hp.com/techreports/2002/HPL-2002-198.pdf
31. Kavitha Ranganathan, Adriana Iamnitchi, Ian Foster, "Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities" In Proceedings of the Global and Peer-to-Peer Computing in Large-Scale Distributed Systems (CCGRID), 2002
32. R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management", In Proceedings of the First ACM/Usenix Symposium on Networked Systems Design and Implementation (NSDI), 2004
33. TUC, DBE Deliverable, D14.1 – DBE Knowledge Representation Models: https://dbe.digital-ecosystem.net/servlets/ProjectDocumentList?folderID=16&expandFolder =16&folder ID=0, May 2005
34. 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
35. TUC, DBE Deliverable, D17.1 – Recommender: https://dbe.digital-ecosystem.net/ servlets/ProjectDocumentList?folderID =16& expandFolder=16&folderID=0, March 2005
36. N. Daswani, H. Garcia-Molina, B. Yang, "Open Problems in data-sharing peer-to-peer systems", In Proceedings of the Int. Conference on Database Theory, 2003.
37. TCD, DBE Deliverable, D24.3 - Final version of the P2P architecture for service search, October 2005
38. TUC, DBE Deliverable, D14.4 – 2nd Release of the Recommender, December 2005
39. Gnutella. http://www.gnutelliums.com/
40. ISUFI, DBE Deliverable: D15.1 - BML First Release: https://dbe.digital-ecosystem .net/servlets/ProjectDocumentList?folderID=16&expandFolder=16&folderID=0
41. 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

10. APPENDIX Knowledge Base/Semantic Registry/Recommender Service Interfaces

The Appendix briefly presents the available interfaces for accessing the DBE Knowledge Base, the Semantic Registry and the Recommender. The source code of these services is available within the swallow project in http://swallow.sourceforge.net.

Knowledge Base Core Service

This is the Knowledge Base Service interface.
Defines the functionality supported by the KB service (**org.dbe.kb.service.KBI**)

Functionality	
void	**closeSession**(java.lang.String sessionID) Close an active session.
java.lang.String	**createNewSession**() Create a new session for submitting a query.
void	**deleteBMLdiagram**(java.lang.String id)
void	**deleteBMLmodel**(java.lang.String id) Delete a specific BML model
void	**deleteIMMmodel**(java.lang.String id) Delete a specific IMM model (Instance)
void	**deleteODMdiagram**(java.lang.String name, java.lang.String ontologyID)
void	**deleteODMmodel**(java.lang.String id) Delete a specific ODM model (Ontology)
void	**deleteSDLmodel**(java.lang.String id) Delete a specific SDL model
void	**deleteSSLdiagram**(java.lang.String id)
void	**deleteSSLmodel**(java.lang.String id) Delete a specific SSL model
void	**deleteUsageData**(java.lang.String id)

void	**fwdQuery**(java.lang.String sessionID, java.lang.String query, java.lang.String servKey)
java.lang.String	**getBML**() Get the BML metamodel specification
java.lang.String	**getBML**(java.lang.String version) Get a specific version of the BML metamodel specification
java.lang.String	**getIMM**() Get the IMM metamodel specification
java.lang.String	**getIMM**(java.lang.String version) Get a specific version of the IMM metamodel specification
java.lang.String	**getODM**() Get the ODM metamodel specification
java.lang.String	**getODM**(java.lang.String version) Get a specific version of the QML metamodel specification
java.lang.String	**getQML**() Get the QML metamodel specification
java.util.Collection	**getResults**(java.lang.String sessionID) Get new results for a specific query, as they are aggregated from the KB P2P mechanism
java.lang.String	**getSDL**() Get the SDL metamodel specification
java.lang.String	**getSDL**(java.lang.String version) Get a specific version of the SDL metamodel specification
java.lang.String	**getServiceID**()
java.lang.String	**getSSL**() Get the SSL metamodel specification
java.lang.String	**getSSL**(java.lang.String version) Get a specific version of the SSL metamodel specification
java.lang.String	**getUsageData**(java.lang.String id)
boolean	**isAlive**()
java.util.Collection	**listBMLmodels**() List the stores BML models
java.util.Collection	**listIMMmodels**() List the stores IMM models (Instances)

java.lang.String[]	**listODMdiagrams**(java.lang.String ontologyID)
java.util.Collection	**listODMmodels**() List the stores ODM models (Ontologies)
java.util.Collection	**listSDLmodels**() List the stores SDL models
java.util.Collection	**listSSLmodels**() List the stores ssl models
java.lang.String	**putResults**(java.lang.String sessionID, java.util.Collection results)
java.util.Collection	**queryUsageData**(java.lang.String xQuery)
java.lang.String	**retrieve**(java.lang.String id, java.lang.String mm, java.lang.String key)
java.lang.String	**retrieveBML**(java.lang.String id) Retrieve a specific BML model
java.lang.String	**retrieveBMLdiagram**(java.lang.String id)
java.lang.String	**retrieveBMLmodel**(java.lang.String id) Retrieve the XML bundle of the BML/SSL model
java.lang.String	**retrieveIMMmodel**(java.lang.String id) Retrieve a specific ODM model (Instance)
java.lang.String	**retrieveODMdiagram**(java.lang.String name, java.lang.String ontologyID)
java.lang.String	**retrieveODMmodel**(java.lang.String id) Retrieve a specific ODM model (Ontology)
java.lang.String	**retrieveSDLmodel**(java.lang.String id) Retrieve a specific SDL model
java.lang.String	**retrieveSSLdiagram**(java.lang.String id)
java.lang.String	**retrieveSSLmodel**(java.lang.String id) Retrieve a specific SSL model
java.util.Collection	**searchByKeywords**(java.lang.String sessionID, java.lang.String metamodelName, java.util.Collection keywords) Submit a query using simple keywords in Knowledge Base

void	**store**(java.lang.String id, java.lang.String data, java.util.Vector metadata, java.lang.String container, boolean enableReplication)
void	**storeBMLdiagram**(java.lang.String id, java.lang.String data)
void	**storeBMLmodel**(java.lang.String id, java.lang.String data) Store BML model
void	**storeIMMmodel**(java.lang.String id, java.lang.String data) Store IMM model (Instance)
void	**storeODMdiagram**(java.lang.String name, java.lang.String data)
void	**storeODMmodel**(java.lang.String id, java.lang.String data) Store ODM model (Ontology)
void	**storeSDLmodel**(java.lang.String id, java.lang.String data) Store SDL model
void	**storeSSLdiagram**(java.lang.String id, java.lang.String data)
void	**storeSSLmodel**(java.lang.String id, java.lang.String data) Store SSL model
void	**storeUsageData**(java.lang.String id, java.lang.String data)
java.util.Collection	**submitQuery**(java.lang.String sessionID, java.lang.String query) Submit a query following the QML metamodel in Knowledge Base
void	**updateBML**(java.lang.String version, java.lang.String data) Update the current BML Metamodel with a new version
void	**updateIMM**(java.lang.String version, java.lang.String data) Update the current IMM Metamodel with a new version
void	**updateODM**(java.lang.String version, java.lang.String data) Update the current ODM Metamodel with a new version
void	**updateSDL**(java.lang.String version, java.lang.String data) Update the current SDL Metamodel with a new version
void	**updateSSL**(java.lang.String version, java.lang.String data) Update the current SSL Metamodel with a new version

Semantic Registry Core Service Interface

This is the Semantic Registry Service Interface.

Defines the functionality supported by the SR service (**org.dbe.kb.service.SRI**)

Functionality	
void	**closeSession**(java.lang.String sessionID) Close an active session.
java.lang.String	**createNewSession**() Create a new session for submitting a query.
void	**deleteODMmodel**(java.lang.String id)
void	**eraseSM**(java.lang.String SMID) De-register a Service Manifest from the Semantic Registry
java.util.Collection	**findBMLdepSMs**(java.lang.String sessionID, java.lang.String id) Find all Service Manifests that contain a specific BML model id
java.util.Collection	**findSDLdepSMs**(java.lang.String sessionID, java.lang.String id) Find all Service Manifests that contain a specific SDL model id
java.util.Collection	**findSSLdepSMs**(java.lang.String sessionID, java.lang.String id) Find all Service Manifests that contain a specific SSL model id
void	**fwdQuery**(java.lang.String sessionID, java.lang.String query, java.lang.String servKey)
java.lang.String	**getBML**() Get the BML metamodel specification
java.lang.String	**getBML**(java.lang.String version) Get a specific version of the BML metamodel specification
java.lang.String	**getBMLmodel**(java.lang.String BMLID) Retrieve a specific BML model
java.lang.String	**getBMLmodel**(java.lang.String BMLID, java.lang.String key)
java.lang.String	**getIMM**() Get the IMM metamodel specification
java.lang.String	**getIMM**(java.lang.String version) Get a specific version of the IMM metamodel specification

java.lang.String	**getODM**() Get the ODM metamodel specification
java.lang.String	**getODM**(java.lang.String version) Get a specific version of the ODM metamodel specification
java.lang.String	**getQML**() Get the QML metamodel specification
java.lang.String	**getQML**(java.lang.String version) Get a specific version of the QML metamodel specification
java.util.Collection	**getResults**(java.lang.String sessionID) Get new results for a specific query, as they are aggregated from the SR P2p mechanism
java.lang.String	**getSDL**() Get the SDL metamodel specification
java.lang.String	**getSDL**(java.lang.String version) Get a specific version of the SDL metamodel specification
java.lang.String	**getServiceID**()
java.lang.String	**getSM**(java.lang.String SMID) Retrieve a Service Manifest with specific ID
java.lang.String	**getSM**(java.lang.String SMID, java.lang.String key)
java.lang.String	**getSSL**() Get the SSL metamodel specification
java.lang.String	**getSSL**(java.lang.String version) Get a specific version of the SSL metamodel specification
java.lang.String	**getSSLmodel**(java.lang.String BMLID) Retrieve a specific SSL model
java.lang.String	**getSSLmodel**(java.lang.String BMLID, java.lang.String key)
boolean	**isAlive**()
java.lang.String[]	**listMostUsedBMLmodels**()
java.util.Collection	**listODMmodels**()
java.util.Collection	**listSMs**() List Service Manifests registered in Semantic Registry
java.lang.String	**publishSM**(java.lang.String data)

	Publish a Service Manifest
java.lang.String	**putResults**(java.lang.String sessionID, java.util.Collection results)
java.lang.String	**retrieveBMLdata**(java.lang.String SMID) Retrieve the BML data part of a specific Service Manifest
java.lang.String	**retrieveBMLdata**(java.lang.String SMID, java.lang.String key)
java.lang.String	**retrieveBMLmodel**(java.lang.String SMID) Retrieve the BML model part of a specific Service Manifest
java.lang.String	**retrieveBMLmodel**(java.lang.String SMID, java.lang.String key)
java.lang.String	**retrieveODMmodel**(java.lang.String id)
java.lang.String	**retrieveSDLmodel**(java.lang.String SMID) Retrieve the SDL model part of a specific Service Manifest
java.lang.String	**retrieveSDLmodel**(java.lang.String SMID, java.lang.String key)
java.lang.String	**retrieveSSLdata**(java.lang.String SMID) Retrieve the SSL data part of a specific Service Manifest
java.lang.String	**retrieveSSLdata**(java.lang.String SMID, java.lang.String key)
java.lang.String	**retrieveSSLmodel**(java.lang.String SMID) Retrieve the SSL model part of a specific Service Manifest
java.lang.String	**retrieveSSLmodel**(java.lang.String SMID, java.lang.String key)
java.util.Collection	**searchBMLByKeywords**(java.lang.String sessionID, java.util.Collection keywords) Submit a query using simple keywords in Semantic Registry searching for BML models
java.util.Collection	**searchByKeywords**(java.lang.String sessionID, java.util.Collection keywords) Submit a query using simple keywords in Semantic Registry
void	**store**(java.lang.String id, java.lang.String data, java.util.Vector metadata, java.lang.String container, boolean enableReplication)

void	**storeODMmodel**(java.lang.String id, java.lang.String data)
java.util.Collection	**submitQuery**(java.lang.String sessionID, java.lang.String query) Submit a query following the QML metamodel in Semantic Registry
void	**updateBMLdata**(java.lang.String SMID, java.lang.String data) Update the BML data part of a specific Service Manifest
void	**updateBMLmodel**(java.lang.String SMID, java.lang.String data) Update the BML model part of a specific Service Manifest
void	**updateSDLmodel**(java.lang.String SMID, java.lang.String data) Update the SDL model part of a specific Service Manifest
void	**updateSSLdata**(java.lang.String SMID, java.lang.String data) Update the SSL data part of a specific Service Manifest
void	**updateSSLmodel**(java.lang.String SMID, java.lang.String data) Update the SSL model part of a specific Service Manifest

Recommender Core Service Interface

This is Recommender Service Interface.

Defines the functionality supported by the RC service (**org.dbe.kb.service.RCI**)

Functionality	
void	**collectRecommendations**() Start the process for collecting recommendations for all the localy stored profiles.
void	**deleteUPMModel**(java.lang.String id) Delete a UPM model based on the document ID containing it.
java.util.Collection	**getProfileResults**(java.lang.String uid)
java.lang.String	**getUPM**() It is not implemented in the recommender.
java.util.Collection	**listUPMs**() Returns a collection with all the stored UPM models.
java.lang.String	**retrieveUPMModel**(java.lang.String id) Retrieve a specific UPM model based on the document ID containing it.
void	**storeUPMModel**(java.lang.String id, java.lang.String data) Used for storing a UPM model-document.

KB Toolkit API

The KB toolkit is a collection of Java APIs that are available in the DBE Studio as a plug-in and help the clients and the tools in various tasks when accessing the DBE Knowledge Base or the Semantic Registry (i.e. Processing of XMI documents, exploiting the DBE Metamodel-specific JMI interfaces, processing of Service Manifest XML documents etc.)

JMI Tool

The JMI Tool API offers functionality for the management of a MOF metadata repository inside Eclipse

JMItool()

JMItool(boolean persistInFilesystem)

True if we wish to persist the repository in the file system
False if we wish to have a main memory repository implementation

Functionality

void	**addModel**(java.lang.String modelData) Add a model in the metadata repository. modelData is the representation of the model in XMI format
void	**clearCache**() Clear the current extent in the metadata repository
java.lang.String	**exportModel**() Export model information from the current extent as a String in XMI format
void	**exportModel**(java.io.OutputStream outStream) Export model information from the current extent in XMI format to the given output stream
void	**exportModel**(java.lang.String filename) Export model information from the current extent in XMI format to the given filename
java.lang.String	**exportPackage**(javax.jmi.reflect.RefPackage rp)

void	**generateModelSpecificJMI**(java.lang.String modelExtName, java.lang.String directory) Generate the metamodels-specific JMI interfaces of the given metamodels extent into the specified directory
java.lang.String	**getCurrentExtent**() Get the name of the current extent
javax.jmi.reflect.RefPackage	**getModelPackage**() Get the root model package of the current extent
void	**importModel**(java.io.File file) Import model information from file in XMI format
void	**importModel**(java.io.InputStream inStream) Import model information from stream in XMI format
void	**importModel**(java.lang.String modelData) Import model information from String in XMI format
void	**initRepository**(java.lang.String metamodelName, java.lang.String extent, java.io.File file) Initialize the metadata repository for a specific metamodel
void	**initRepository**(java.lang.String metamodelName, java.lang.String extent, java.lang.String metamodelData) Initialize the metadata repository for a specific metamodel
static void	**setInMemory**(boolean inMemory) Force the in-memory repository implementation
static void	**setLogFile**(java.lang.String repLog) Specify the log file
static void	**setRepositoryDirectory**(java.lang.String repDir) Specify the directory of repository when the filesystem persistency is enabled
void	**switchRepository**(java.lang.String extent) Switch extents and make the given extent the current one

IQueryFormulator

An API for formulating QML queries and Expressions

Functionality

void	**clearQuery** () Keeps track of all Objects created by the formulator and clears every time needed.
org.dbe.kb.metamodel.qml.contextdeclarations.InvariantContextDecl	**formulateConstaint**(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 opearation, eg "=", 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 opearation, eg "=", and a String value.
org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression	**formulateExpressions**(java.util.Collection expresions, int type)
org.dbe.kb.metamodel.qml.ocl.expressions.OclExpression	**formulateFuzzyExpression**(java.util.Collection exprs, double[] weights, int type)
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

SMtool

Service Manifest Tool. This API enables the creation and processing of Service Manifests

SMtool()

Functionality

java.lang.String	**exportServiceManifest**(java.lang.String smid) Export the Service Manifest as a String XML document
void	**exportServiceManifest**(java.lang.String smid, java.io.OutputStream outs) Export the Service Manifest as an XML document to the given output stream
void	**exportServiceManifest**(java.lang.String smid, java.lang.String file) Export the Service Manifest as an XML document to the given file
java.lang.String	**getBMLdata**() Get the BML-data part of the Service Manifest
java.lang.String	**getBMLid**() Get the BMLID part of the Service Manifest
java.lang.String	**getBMLmodel**() Get the BML model part of the Service Manifest Contains the BML/SSL)
java.lang.String	**getBusinessPart**() Get T]the BML part of the Service Manifest
java.lang.String	**getDescription**() Get the description of the Service Manifest
java.lang.String	**getIconURL**() Get the icon URL part of the Service Manifest
java.lang.String	**getLastUpdateDate**() Get the Last Update of the Service Manifest
java.lang.String	**getName**() Get the name of the Service Manifest
java.lang.String	**getPublicationDate**() Get the publication date of the Service Manifest
java.lang.String	**getSDLmodel**() Get the SDL-model part of the Service Manifest

java.lang.String	**getServicePart**() Get the SSL part of the Service Manifest
java.lang.String	**getServiceType**() Get the service type of the Service Manifest
java.lang.String	**getSMID**() Get the SMID of the Service Manifest
java.lang.String	**getVersion**() Get the version of the Service Manifest
void	**importServiceManifest**(java.io.File file) Import a Service Manifest from a file
void	**importServiceManifest**(java.io.InputStream ins) Import a Service Manifest from an input stream
void	**importServiceManifest**(java.lang.String data) Import a Service Manifest from a String
void	**setBMLdata**(java.lang.String bmlxmi) Set the BML-data part in the Service Manifest
void	**setBMLmodel**(java.lang.String bmlxmi) Set the BML-data part in the Service Manifest
void	**setDescription**(java.lang.String desc) Set the description the Service Manifest
void	**setSDLmodel**(java.lang.String sdlxmi) Set the SDL part in the Service Manifest
void	**setSMID**(java.lang.String smid) Set the SMID in the Service Manifest