KP-LAB Knowledge Practices Laboratory – Specification of the SWKM Architecture (V1.0) and Core Services

Vassilis Christophides, Dimitris Kotzinos, Yannis Tzitzikas, Nicolas Spyratos, Hanen Belhajfrej, Mamadou Nguer, Jan Paralic, Karol Furdik, Peter Smatana, Martin Sarnovsky, et al.

27490

KP-LAB

Knowledge Practices Laboratory

Integrated Project

Information Society Technologies

T5.1. Specification of the SWKM Architecture (V1.0) and Core Services

Due date of deliverable: **31/07/2006**
Actual submission date: **31/07/2006**

Start date of project: 1.2.2006 Duration: 60 Months

Organisation name of lead contractor for this deliverable: **ICS-FORTH**

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	PU

Participants

Partner	Parther's short name	Participant	Email
	ICS-FORTH	Vassilis Christophides	christop@ics.forth.gr
	ICS-FORTH	Dimitris Kotzinos	kotzino@ics.forth.gr
	ICS-FORTH	Yannis Tzitzikas	tzitzik@ics.forth.gr
	LRI-ORSAY	Nicolas Spyratos	spyratos@lri.fr
	LRI-ORSAY	Hanen BelhajFrej	hanen@lri.fr
	LRI-ORSAY	Mamadou Nguer	nguer@lri.fr
	TUK	Jan Paralic	Jan.Paralic@tuke.sk
	TUK	Karol Furdik	kfurdik@stonline.sk
	TUK	Peter Smatana	Peter.Smatana@tuke.sk
	TUK	Martin Sarnovsky	Martin.Sarnovsky@tuke.sk
	TUK	Peter Bednar	Peter.Bednar@tuke.sk
	UEP	Pavel Smrz	smrz@fit.vutbr.cz
	UEP	Vilem Sklenak	sklenak@vse.cz
	UEP	Vojtech Svatek	svatek@vse.cz
	UEP	Martin Kavalec	kavalec@vse.cz
	UEP	Martin Svihla	svihla@vse.cz

Version history

Version	Date	Author(s)	Description
0.5	15/06/2006	Vassilis Christophides	First draft
0.8	14/07/2006	Dimitris Kotzinos	Second Draft
0.9	15/07/2006	Pavel Smrz	Text Mining Services added
0.95	20/07/2006	Karol Furdik	Text Mining Services enriched
0.98	24/07/2006	Dimitris Kotzinos	Integration
1.0	31/07/2006	FORTH	Final

Table of Contents

Executive Summary

The objective of WP5 is to develop a generic middleware supporting knowledge management services for 'trialogical' learning applications. More precisely, the KP-Lab Semantic Web Knowledge Middleware (SWKM) aims to facilitate knowledge creation processes by supporting advanced interactions of collaborating learners (or workers) with knowledge artefacts (i.e. discovery, access, evolution, recommendation and mining). In this deliverable we present the high-level functionality of SWKM along with the Service-Oriented Software Architecture of the prototype system that will be developed by M12 (V 1.0) and the subsequent phases of the project. The proposed architecture broadly distinguishes three generic modules of the SWKM, i.e. the Knowledge Repository – responsible for the provision of scalable persistence services for large volumes of knowledge artefacts' descriptions and ontologies; the Knowledge Mediator – responsible for the provision of services handling the main registry, discovery and evolution for KP-Lab knowledge artefacts; and the Knowledge Matchmaker – responsible for the provision of services that support interactions of KP-Lab users with knowledge artefacts employing their semantic descriptions. The services corresponding to each one of these modules are described along with the proposed functionality for each one, based upon the motivating scenarios and the subsequent definition of the key concepts of Trialogical Learning.

1 Introduction

Classical learning theories are based either on the knowledge acquisition metaphor (i.e. a learner individually internalizes a body of knowledge) or on the social participation metaphor (i.e. a group of learners collaboratively appropriate a body of knowledge). Although widely accepted, these theories do not sufficiently capture innovative practices of both learning and working with knowledge (i.e. knowledge practices). Only sharing of knowledge in action, i.e., sharing the process of learning itself, is a reliable base for developing a shared cognition (seen both as a group and an individual characteristic). In this context, the emerging theory of "Trialogical Learning" (TL) focuses on the social processes by which learners collectively enrich/transform their individual and shared cognition. According to TL, knowledge creation activities rely heavily on the use, manipulation and evolution of shared knowledge artefacts externalizing a body of (tacit or explicit) knowledge [Paavola et al 2004]. By representing their cognitive structures or knowledge practices under the form of artefacts, individual learners can interact among themselves as well as with external tools (e.g. computers, information resources) to negotiate the meaning of concepts and signs embodied in these artefacts and thus, finally reach a common understanding of the problem at hand. In order to communicate and meaningfully interpret their individual viewpoints, cooperating learners need to agree on a common conceptual frame of reference.

A natural candidate for establishing such a shared conceptual framework (between humans but also machines), and making meaning explicit (by definition and context), is the notion of ontologies [Gruber 1997], [Uschold et al 1996]. An ontology provides the means to define a basic vocabulary of terms, as well as, to specify (to a certain degree) their meaning. This includes definitions of concepts and how they are inter-related; these concepts collectively impose a structure on a domain of discourse and constrain the possible interpretations of terms. Unlike traditional approaches that conceive ontologies as thorough engineering artefacts objectifying a body of knowledge (and thus separate them from their original social context of creation to transfer them across the domain), in TL ontology use, creation and evolution can be seen as an adequate socially determined activity where learners constantly interact in order to grasp and negotiate meaning between their individual and group conceptualizations. Shared ontologies would thus become an emergent effect of open-ended interactions within or across groups of individuals as opposed to being a firm commitment of a closed group of domain experts following strict design processes and policies [Aberer et al 2004].

It should be stressed that knowledge artefacts are omnipresent in TL processes since they mediate all activities and tasks conducted by the learners. Knowledge artefacts essentially capture and preserve the socially shared knowledge within a community and could take various forms. They range from physical resources and tools like documents (e.g. a survey paper), software code or concept maps and ontologies shared by a group to less tangible (i.e. conceptual) artefacts like plans to scientific theories and languages. Hereafter, we shall use the term knowledge artefact to refer to resources produced/consumed by a group of learners as long as they are appropriately

described (i.e. annotated) according to the semantics codified in one or several ontologies. It is worth noticing that ontologies (taxonomies, concept maps, or domain models) are also considered as artefacts themselves capturing more or less formal forms of knowledge and thus can in turn be described and discovered using other ones. The KP-Lab Semantic Web Knowledge Middleware (SWKM) under development in WP5 aims to support advanced interactions of learners (or knowledge workers) with knowledge artefacts.

In this deliverable we present the high-level functionality of the three SWKM components, namely:

- **Knowledge Repository**, offering scalable persistence services for large volumes of KP-Lab knowledge artefacts,
- **Knowledge Mediator**, providing the main registry, discovery and evolution services for KP-Lab knowledge artefacts,
- **Knowledge MatchMaker**, supporting advanced interactions of KP-Lab users with knowledge artefacts through appropriate recommendation, and mining services,

along with the Service-oriented Software Architecture of the prototype system that will be developed by M12 (V 1.0) and the subsequent phases of the project.

2 Motivating Scenario for 'Trialogical' Learning

In this section, we briefly present one of the scenarios (Creating, Re-using and Evolving Knowledge Artefacts) proposed by KP-Lab design teams (DT5) and outline the main interaction modes and knowledge spaces, as well as the activities involved in knowledge creation processes.

Let us assume that a project team (a group of students or professionals) is engaged in solving a complex and often ill-structured problem that requires a lot of expertise in disparate domains. The assignment was posed (by an instructor or client) during an initial face-to-face meeting and both parties were engaged in negotiations (in subsequent physical or virtual meeting) about what the project is meant to achieve and how team work will be organized (in sub/groups with partitioned or overlapping tasks).

- *Bootstrapping Step*: The problem in the initial stage is still very vague, and, the instructor (or client) provides team members with an amount of digital (or non-digital) artefacts (e.g. papers, reports) as background knowledge on the project assignment. Past knowledge practices (under the form of recorded audiovisual sessions) can also be used in this respect. Whenever possible, the instructor (or client) can additionally provide an initial concept map (visualized through a user friendly interface) highlighting the core notions, processes or agents involved in the problem to be addressed. In a first step, each group member is responsible for evaluating the already provided background material with respect to its applicability and usefulness in the project. Learners start working (personal space) on the available knowledge artefacts to clarify project needs and requirements until the next team meeting (group space). For instance, a preliminary useful task could be the classification (i.e. annotation) of the proposed knowledge artefacts according to the concept map provided by the instructor (assistance from tools can be considered in this respect). When such preliminary conceptualizations are not given, learners could alternatively search for ontologies or classification schemas

employed in a relevant domain or professional community in order to progressively refine them. In both cases multiple ontologies could be used to shed light on different angles of a common problem tackled by the group.

- *Team Cooperative Work*: The subject of the teamwork is the join development of knowledge artefacts (either from scratch or by relying on existing ones) related to the problem at hand through a progressive inquiry process. During this process, learners are interacting via multiple physical or virtual meetings (e.g. synchronous, asynchronous) to grasp and negotiate meaning between their individual and group problem understanding. Different interim versions of the produced knowledge artefacts can coexist both under the personal or group knowledge space allowing team members to work either on the same shared artefacts or on their personal counterparts. At each project phase, these versions capture the evolution of the individual or group knowledge within a team. In this multi-step process, learning is not limited to an individual knowledge acquisition neither to a simple group interaction, but involves shared efforts of advancing team ideas and transforming the underlying social practices. For instance, the competent use of a conceptualization quite often also requires understanding the rationale on which it is build, its underlying theoretical foundation, as well as its historical evolution. In order to grasp the provisional character of ontologies learners might start developing a rather sophisticated set of epistemological beliefs themselves. Furthermore, when multiple conceptualizations are employed (to codify complementary viewpoints of the problem under investigation), learners are engaged in a profoundly reflective activity tackling the theoretical foundation of the employed conceptualizations (i.e. their meta-models). Last but not least, when a shared conceptualization is collaboratively formed from individual ones (to reach consensus on the concepts and relations that are relevant to the task at hand), learners are constantly exposed to cognitive conflicts or helps to unravel prevalent misunderstandings. Such efforts may even result in the revision (or contraction) of the conceptualizations given or chosen at some project phase. In all these cases, reconstruction of individual cognition requires a profound and mutual understanding of collaborators' interpretations of the problem. Thus, learners could take benefit from advanced tools support to *access*, *compare* or *merge* shared knowledge artefacts (described resources, taxonomies, concept maps, or domain models). Finally, a team may resort to the expertise of an external group by subscribing its preferences (relative to a similar or complementary problem) over the shared knowledge artefacts' space; the team will be notified about suitable recommendations when new knowledge artefacts pertaining to these preferences have been made available by other experts or peer colleagues.
- *Team Work Outcome*: At the end of each project meeting (seen as a milestone) the team summarizes its intellectual achievements under the form of elaborated concept maps (or ontologies) and document reports. Furthermore the team reviews the collection of sketches, prototypes and notes in order to reflect on their own work and to outline their lessons learned by this working/learning experience. Some clearly central but yet unresolved questions, and linked information resources, are also specifically pointed out for future work in later phases of the project.

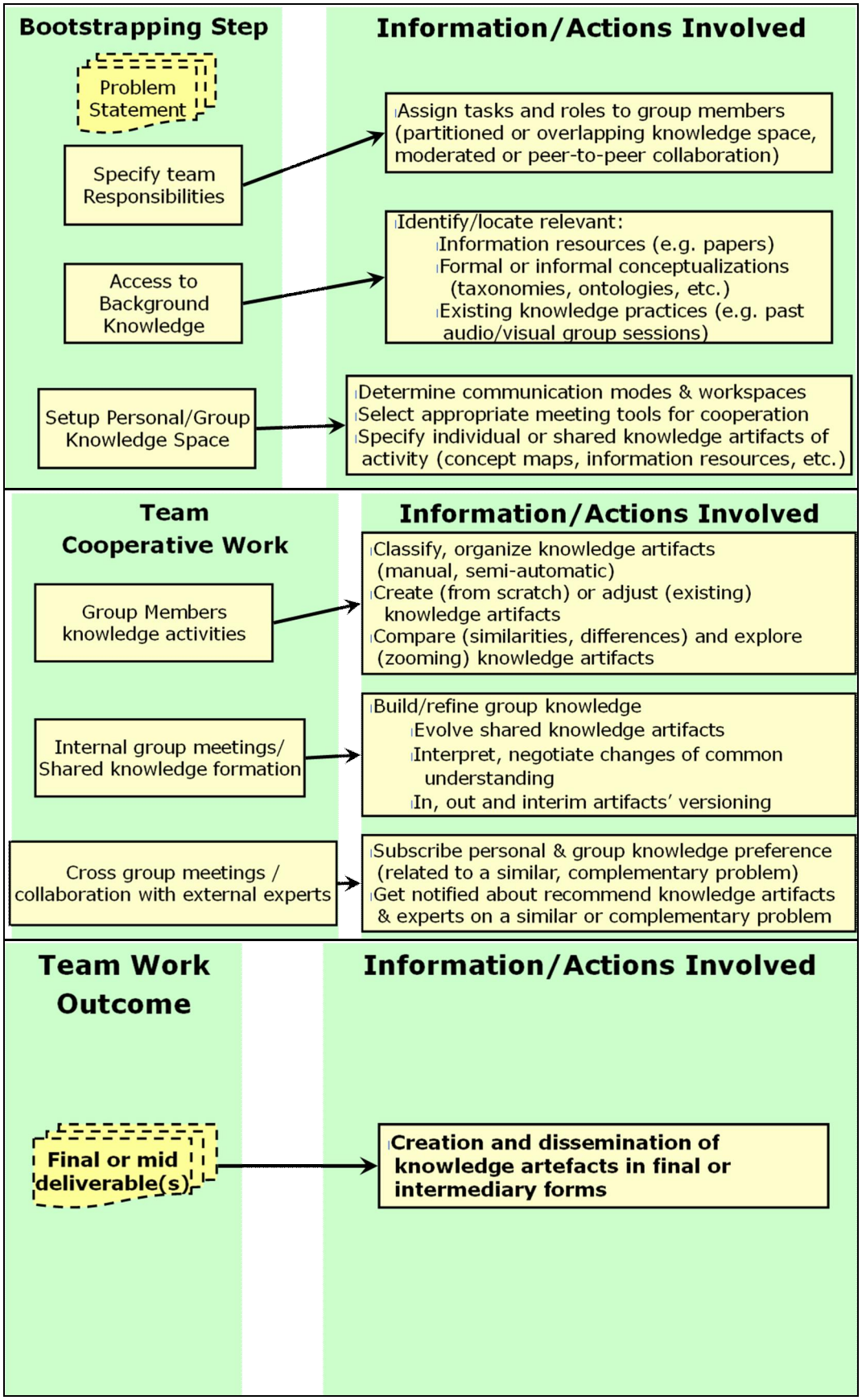

Figure 2.1: A motivating scenario for Trialogical Learning

2.1 Knowledge Artefacts

The type and form in which knowledge is shared influence strongly the process of building a shared understanding of the problem at hand and hence the effectiveness of learning. As already pointed out in the literature [Anderson 1975], knowledge may be of several types, each of which may be made explicit. Knowledge *about* something is called *declarative knowledge*. A shared, explicit understanding of concepts, categories, and descriptors lays the foundation for effective communication and knowledge sharing in organizations. Knowledge of *how* something occurs or is performed is called *procedural knowledge*. Shared explicit procedural knowledge lays a foundation for efficiently coordinated action in organizations. Knowledge *why* something occurs is called *causal knowledge*. Shared explicit causal knowledge, often in the form of organizational stories, enables organizations to coordinate strategy for achieving goals or outcomes. More generally, there exist four analytically distinct but in practice interrelated discourses on knowledge in various social sciences [Sørensen et al 2002]:

- knowledge as *object*: knowledge is seen as a representation of a pre-given world (objectification) while human intelligence can be seen as information processing and rule-based manipulation of symbols.
    - the fundamental assumption is that the world is pre-given, and its aim is to create the most accurate or 'truthful' representations of this objective world (cf. acquisition metaphor of learning).
- knowledge as *interpretation*: knowledge is associated with human intersubjective interpretations and dependent very much on 'the point of observation' of the interpreter and that the process of interpretation simultaneously shapes and is shaped by social reality
    - The fundamental assumption is that knowledge is brought forth through the creative (and communicative) act of human cognition and interpretation (cf. participation metaphor of learning; social discourse).
- knowledge as *relationship*: knowledge does not exist in an isolated state in the objective world, but rather resides within a variety of (socialcultural) contextual factors that are inseparably connected with a body of knowledge.
    - can be seen as an interconnected web of relationships in which human interpretative acts ceaselessly shape and maintain, both intentionally and unintentionally, the relational setting of the web and contextual disposition of the social reality (cf. participation metaphor of learning; interaction with environment).
- knowledge as *process*: knowledge is not a static entity but the manifestation of a dynamic process of 'knowing' by which human beings make sense of the world and reality by using external artefacts which augment intelligent activity.
    - there exists neither subject nor object that can be isolated from reality itself and both subject and object are intrinsically bound to an ongoing processes of transition of reality (cf. knowledge-creation metaphor of learning).

In this context, the two main forms of human knowledge are *explicit* and *tacit* [Polanyi 1966]. Tacit knowledge resides mainly within individuals' mind and cannot easily be expressed, codified, or explicated (e.g. embodied knowledge, intuition). Examples of tacit knowledge include

- knowledge embedded in skills, i.e., rules, procedures, and know-how, that may be difficult to verbally explicate (procedural skills);

- informal knowledge that encapsulates theoretical knowledge organized around problems and cases, for instance how to approach and solve problems;
- knowledge embedded in regular and predictable patterns of organizational activities (e.g. an intuitive grasp of the relevance of data that an organization should focus on and provides a basis for an inimitable competitive advantage).

On the other hand, explicit knowledge is the knowledge that has been or can be articulated, codified, and stored in certain media and thus, can easily be accessed and shared. The most common forms of explicit knowledge are manuals, documents, procedures, and stories (but also audio-visual information). Works of art and product design can be seen as other forms of explicit knowledge where human skills, motives and knowledge are externalized. The objective of the KP-Lab Semantic Web Knowledge Middleware (SWKM) is to provide generic management services for capturing and archiving; discovering and accessing; combining, modifying and tracking [Ernst and Young 1997]) knowledge artefacts which encode more or less formal forms of explicit knowledge (i.e. declarative, procedural and causal). In particular, KP-Lab abstracts from the various media types employed to encode explicit knowledge by considering information resources that are uniquely identified (e.g. by a URI) and appropriately described according to the semantics of a domain of discourse (e.g. an ontology).

2.2 'Trialogical' Learning Activities and Interactions

TL emphasizes on the intrinsic collaborative nature of innovative practices for both learning and working with knowledge. Learning is viewed as a social process in which learners collectively enrich and transform both their individual and shared understanding. The main question in this context is how shared knowledge artefacts (or more generally objects of activity) are collaboratively formulated and developed by using mediating information resources, signs and tools. As a matter of fact, artefacts are developed and exploited in ways such that they can mediate certain activities within a community of practice and hence artefacts become an intrinsic part of this community. These artefacts are employed to alter, extend or preserve *group knowing*, *sense-making* or *decision-making* [Stahl 2003]. Sharing artefacts can be viewed as a collective group problem-solving activity for the purpose of aiding, enhancing, or improving individual and group cognition [Hutchins 1995]. Artefacts are created, used or evolved within a cognitive space and a socio-cultural environment and they are employed by the learners to configure and facilitate group decision-making, thinking and communication [Spillers et al 2003]. In this context, learners are interacting with themselves as well as with mediating (material or conceptual) artifacts in order to shape a common understanding and thus as a first step internalize a body of knowledge. This knowledge is externalized in the sequel under the form of personal or group knowledge artefacts. In TL scenario described previously we can distinguish three basic modes of knowledge artefact mediated group interaction [Stoyanova et al 2002]:

- *Peer interaction*: Group members work autonomously (asynchronous activity) and produce intermediate knowledge artefacts embodying their individual mental constructs and precepts (personal space) that are made available in the sequel to their peer colleagues. This loop is repeated until all group members reach a common vision of the problem. The process of knowledge acquisition, creation and internalization is individual.

- *Moderated interaction*: Interaction is facilitated by a group moderator (the role is taken by one of the group members) who is adjusting individually produced artefacts until a common group vision is reached. The representations of individual cognitive structures are not directly accessible but group members are involved in the process (synchronous/asynchronous activity) of negotiation of meanings and ideas that take place between them and the moderator (shared dialogue and interpretation).
- *Shared interaction*: Group members interact directly (synchronous activity) and pursue common efforts to solve the problem as a group (group space). They share their knowledge in action. Knowledge is communicated in the process of its appropriation and creation. Collaborative actions of learners are the individual inputs toward a shared cognition.

Moreover, based on activity theory [Vygotsky 1978], [Leontiev 1978], [Engeström 1999], temporal coordination of an activity is itself an activity. The object of an activity can be another activity and temporal coordination is thus in itself an activity, which seeks to integrate distributed collaborative actions. The dynamic nature of any activity implies that temporal coordination can be achieved both as an action within the overall collaborative activity and as an activity in itself directed towards another collaborative activity. In this sense coordination can be achieved both *intrinsically* within a group of collaborating actors sharing the same artefact – i.e. the actors organize and coordinate the actions themselves – and *extrinsically* to the group – i.e. the actions are organized and coordinated by someone outside the group. In particular, there exist three distinct levels of collaborative work [McGrath et al 1986]:

- *Synchronization* is an ad hoc effort aimed at ensuring that action "a", by person "i", occurs in a certain relation to the time when action "b" is done by person "j" according to the conditions of collaborative activity. Because synchronization is tied to the conditions of the activity, synchronization corresponds to the operational level of temporal coordination.
- *Scheduling* is to create a temporal plan by setting up temporal goals (i.e. deadlines) for when some event will occur or some product will be available, and is thus the anticipatory (action) level of temporal coordination.
- *Allocation* is to decide how much time is devoted to various activities. The essence of allocation is to assign resources according to the overall motives of the collaborative work setting and hence reflects a temporal priority according to different motives. Thus, allocation is the intentional (activity) level of temporal coordination.

As any other activity, temporal coordination is also mediated by artefacts [Bardram 2000]. The coordination of activities in time essentially determines *when* some event will occur or some results will be available in relation to other activities and actions. A particular effective way to do this is to establish starting times and deadlines according to some external and socially shared time measurement. Hence, a temporal artefact, such as the clock or the calendar, can be turned into a *temporal coordination artefact*, mediating the temporal coordination, when shared within a collaborating community of practice. The practical process of realizing temporal coordination cannot be detached from the conditions of the concrete situation. Temporal coordination is shaped according to the conditions of its object (i.e. the collaborative activity it tries to coordinate in time) and the conditions of the socio-cultural environment in which it takes place (e.g. an educational or organizational setting).

SWKM relies on ontologies in order to support interactions among learners and knowledge artefacts involved in learning processes. Ontology-based interactions include but are not limited to the ones that deal with organizing or annotating shared artefacts created by an individual or a group as well as sorting, classifying and retrieving background knowledge artefacts relevant to the problem at hand. Several ontologies will be used to capture declarative (about), procedural (how), causal (why) or temporal (when) knowledge regarding a problem addressed by a group. Moreover, ontologies themselves might be possibly formed collaboratively based on the individual conceptualizations, if a consensus on the concepts and relations that are relevant to the task at hand can be reached. The objective of the KP-Lab SWKM is to support ontology-based interactions inside both personal and group (or subgroup) knowledge spaces equipped with appropriate user roles and access rights.

2.3 Shared Knowledge Spaces

Collaborative "knowledgeware" technologies aim to amplify trialogical processes of learning and working with shared knowledge artefacts of inquiry. This may take place through computer-based environments that are structured around shared spaces for creating and storing, annotating and disseminating, discussing and evolving knowledge artefacts. Rather than two relatively independent cognitive systems which simply exchange messages, both personal and group spaces are viewed as a cognitive system *per se* with is own properties i.e., a shared space. In particular, KP-Lab shared spaces aims to capture various kinds of knowledge (objectified, interpretative, relationship, process) shared by a community of practice.

Shared knowledge spaces are composed of *personal* and (sub-) *group* spaces allowing to access (i.e. store, annotate, retrieve) and manipulate (i.e. compare, evolve, version) the knowledge artefacts produced/consumed by a group. More precisely, the personal space comprises knowledge artefacts (i.e. information resources and conceptualizations) either given by the instructor or discovered by the learners themselves according to the tasks assigned at a certain project phase. Having these knowledge artefacts at hand, each learner may use them exclusively within his personal workspace (e.g. to organize the *object space*), and/or share them with other peer colleagues in order to be further discussed, criticized, and evaluated (e.g. the *interpretative space*). In this context, exploring and providing feedback about interim versions of a knowledge artefact that has been made available by others is a first task for building a shared understanding about the working problem. On the other hand, the group space contains only those knowledge artefacts (eventually more than one when complementary viewpoints are required by the problem at hand) for which members had agreed (e.g. the *relationship space*).

In order to reach consensus on a group knowledge artefact, learners are interacting among themselves as well as with the knowledge artefacts (as described previously) available in both personal and group spaces (i.e. the *process space*). The shared space essentially provides the means to devise group knowledge artefacts conversationally by supporting appropriate (meeting) tools to discuss and record argumentation regarding concepts' specifications. Additionally, it may provide tools to negotiate the meaning of commonly specified concepts by inspecting the effect of the proposed changes when personal ontologies are mirrored to the group one. Going one step further, shared space tools could also assist learners to come up with some decision,

for instance when an individual suggestion is either accepted by all or by most of peer colleagues.

Any group member can make available to the shared space a new knowledge artefact, by specifying the appropriate access rights as well as by describing its subject, scope and purpose. Inversely, any collaborator can replicate in his personal space an artefact version developed by others, re-specify, change, enrich, compare, merge it with her own, exploit it, and publishing it back to the shared space by posting new issues, arguments and so forth. The latter may also include the possibility to inspect the impact to personal knowledge artefacts of an interim group conceptualization.
Thus, a *shared knowledge space* is a virtual collaboration space offering facilities for interacting with knowledge artefacts during a cooperative learning (or working) process along with graphical tools for representing and accessing the involved artefacts, processes and participants. To support a collaborative building of knowing, shared knowledge spaces should provide functionality for:

- *Social interaction*: facilitating complex interactions, coordinating learning tasks, helping participants to maintain an overview of them, allowing participants to negotiate group decisions and knowledge building.
- *Social awareness*: displaying or comparing alternative interpretations of different participants in collaboration and keeping track of who knows or does what, when, where.
- *Social decision support*: producing, integrating, and synthesizing diverse interpretations of a problem at hand so that all participants can come to respect and understand the differences caused by diverse viewpoints and interests of the contributing group members, and there can be a movement towards consensus on at least some of the issues involved.
- *Personal and group knowledge management*: annotating, archiving and accessing knowledge artefacts that arise in group interactions, including argumentation from broad discourses, and organize them in a flexible way according to various perspectives for further manipulation and sharing.

3 Knowledge Creation Processes and Trialogical Learning

Knowledge *creation* is at the heart of TL and in general of knowledge development (including also knowledge adoption, distribution, review and revision) within an organization [Bhatt 2000]. A popular model[1] considers knowledge creation as a process including four conversion patterns [Nonaka & Takeuchi, 1995]: *socialization*, which involves sharing tacit knowledge between individuals, *externalization*, as tacit knowledge is articulated or translated into readily understandable forms (explicit), *combination*, the conversion of explicit knowledge into more complex sets of explicit knowledge, and *internalization*, where people identify explicit knowledge that is relevant to their work and apply it in their actions and practices (tacit knowledge). Although knowledge is created and transformed spirally at various levels (individual, group, organizational or inter-organizational), the individual remains at the kernel of a knowledge creation process because new ideas emerge only through their cognitive efforts. Since knowledge creation aims to enrich and deepen the knowledge of participating learners (or workers), it is important to investigate how they can develop

[1] Employed by several EU projects such as OntoKnowledge, OntoWeb, Onto-Logging, KnowledgeWeb, PROLEARN, etc.

tacit knowledge that would facilitate the creation of new explicit knowledge. As a matter of fact, to capture the intrinsic cooperative nature of the teamwork presented in the TL scenario of Section 2 we need more expressive frameworks such as [Kukkonen 2001] which distinguishes between the following four activities: *comprehension*, as a complex process of surveying and interacting with the social context as well as integrating with background knowledge to identify problems, needs and opportunities, *communication*, for exchanging knowledge artefacts, *conceptualization*, as a reflection process articulating tacit knowledge to form explicit conceptualizations, and finally *collaboration*, which is the team interaction using the produced/consumed knowledge artefacts (see also Figure 3.1). Comprehension and communication are similar respectively to internalization and socialization, while conceptualization includes both externalization and combination. Collaboration now is viewed in TL as a process of innovative inquiry where the aim is to progressively expand one's knowledge and skills based on social interaction, decision support and awareness. It is characteristic of this kind of knowledge advancement that it takes place within innovative knowledge communities rather than only within individuals (although individuals with different skills have an important role) [Paavola et al 2004]. Building and maintaining a shared conception (understanding) of a problem at hand is both a group and an individual characteristic while possesses personal meaning for each group member. All knowledge artefacts and pieces of the shared conception are meaningfully integrated into the cognitive structure of collaborators and are interpreted on a similar frame of reference.

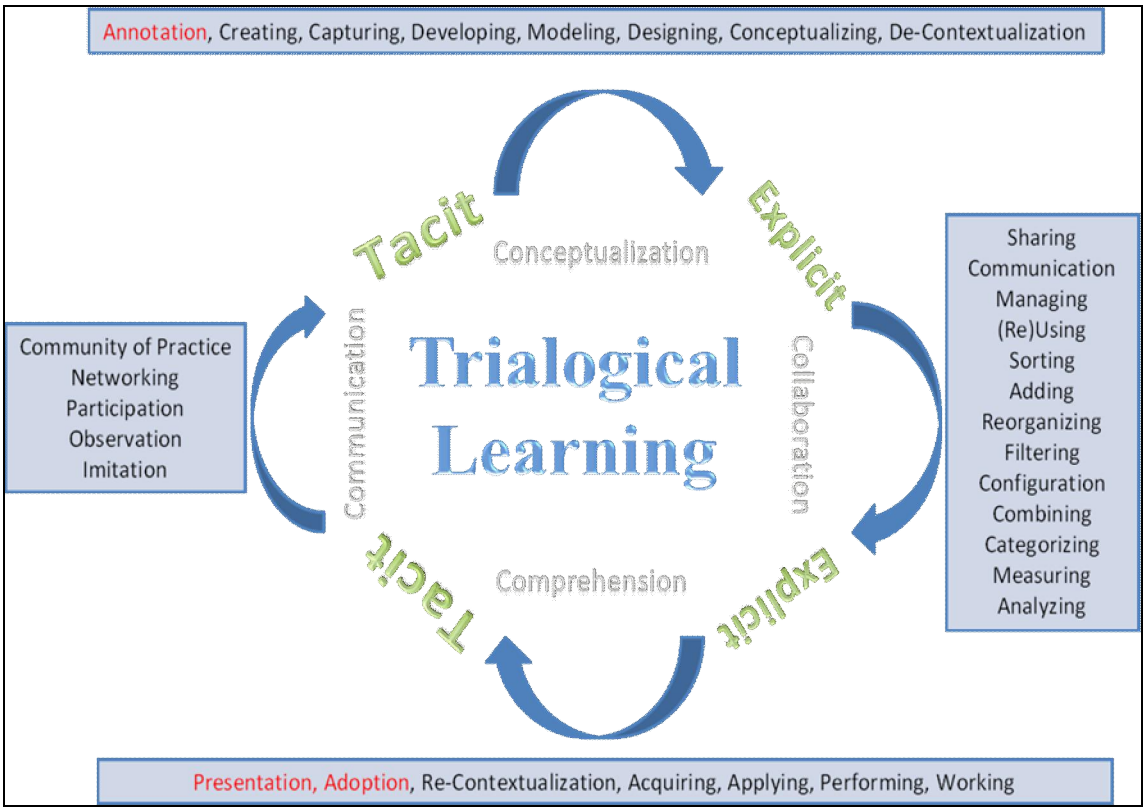

Figure 3.1: Knowledge Management Aspects in Trialogical Learning

In the context of SWKM, ontologies [Gruber 1997], [Uschold et al 1996] are employed to capture such a shared conceptual framework (between humans but also machines), and make meaning explicit (by definition and context). An ontology

provides the means to define a basic vocabulary of terms, as well as, to specify (to a certain degree) their meaning. This includes definitions of concepts and how they are inter-related, which collectively impose a structure on a domain of discourse and constrain the possible interpretations of terms. Unlike traditional approaches that conceive ontologies as thorough engineering artefacts objectifying a body of knowledge (and hence separate them from their original social context of creation to transfer them across the domain), in TL ontology use, creation and evolution can be seen as an adequate socially determined activity where learners constantly interact in order to grasp and negotiate meaning between their individual and group conceptualizations. Ontologies would thus become an emergent effect of open-ended interactions within or across groups of individuals as opposed to be a firm commitment of a close group of domain experts following strict design process and policies [Aberer et al 2004]. We believe that a TL perspective is particularly useful when the original community evolves either because members leaving and entering at free will or because their commitments are changing, and thus a new consensus may shape up invalidating the knowledge already codified in the original ontology [Mika2004].

Hereafter, we shall use the term knowledge artefact to refer to information resources produced/consumed by a group of learners as long as they are appropriately described (i.e. annotated) according to the knowledge codified in one or several ontologies. It is worth noticing that ontologies (taxonomies, concept maps, or domain models) themselves are also knowledge artefacts and thus can in turn be described and shared using other ontologies. Knowledge artefacts of a TL activity are subject of an endless re-interpretation and interactive articulation by a collaborating group of learners (or workers). To support cooperative teamwork, SWKM will provide advanced interaction services with knowledge artefacts encoded using Semantic Web (SW) languages (e.g. RDF/S, OWL). These services are offered by the following tree main SWKM components:

- *Knowledge Mediator*: provides the main registry, discovery and evolution services for knowledge artefacts. It essentially mediates access to and changes of knowledge artefacts by employing personal or group conceptualizations under the form of RDF/S ontologies.
- *Knowledge MatchMaker*: supports advanced knowledge recommendation and mining services for knowledge artefacts. It essentially enables to match information resources with the employed ontologies as well as knowledge artefacts produced/consumed within a group according to various learners' preferences.
- *Knowledge Repository*: offers scalable access (querying/updating, loading/exporting) services for large volumes of RDF/S resource descriptions and schemas stored in a persistent store.

4 Functionality of the Semantic Web Knowledge Middleware (SWKM)

4.1 Overview

The Semantic Web Knowledge Middleware (SWKM) is responsible for providing the means to the rest of the tools to create, store and subsequently retrieve, share and

transfer knowledge. These functionalities will be implemented by WP5 in terms of the appropriate services, as can be seen in Figure 4.1. This array of services is based on the proper use of namespaces and graphspaces (see section 4.2.1) and is broken to three conceptual and also development modules:

- the knowledge repository,
- the knowledge mediator, and,
- the knowledge matchmaker,

which are further explained in the subsequent sections.

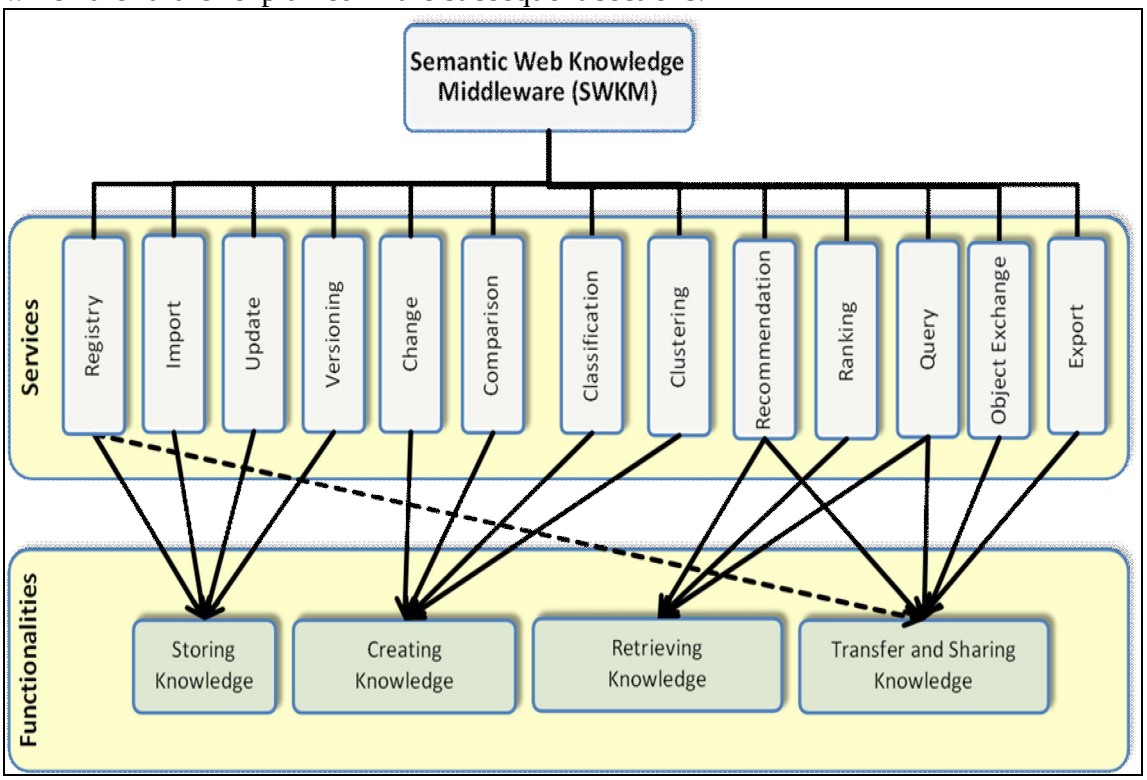

Figure 4.1: SWKM Services for Knowledge Management

4.2 Knowledge Repository [ICS-FORTH]

The *Knowledge Repository* will provide scalable persistence services for large volumes of knowledge artefacts' descriptions and ontologies. Access and manipulation will be supported by declarative query [Karvounarakis et al 2003], view [Magkanaraki et al 2004] and update [Magiridou et al 2005] SW languages exhibiting important optimization opportunities. Compared to API-based knowledge application development, the languages supported by the *Knowledge Repository* aim to satisfy the requirements of SWKM services and KP-Lab applications for *expressiveness* (the ability to express accurately what the query/update initiator wants), *generality* (the ability to implement easily new query/update functionality) and *performances* (the ability to respond in a fast way to a query/update request). The repository will be built upon the FORTH-ICS RDFSuite open source platform[2].

[2] http://139.91.183.30:9090/RDF/

4.2.1 RDF/S Namespaces and Graphspaces

The descriptions of knowledge artifacts as well as their involved conceptualizations will be in represented and handled in SWKM as RDF/S schemas and resource descriptions (OWL extensions will be considered in the later phases of the project). In order to support *personal* and *group* knowledge management based on *multiple conceptualizations* (i.e. ontologies) the knowledge repository should be able to distinguish schemas and descriptions according to the actors (individual or group) involved in their creation. To this end, the SWKM knowledge repository will be able to store, retrieve and update RDF/S schemas and descriptions based on the *name* or *graph spaces* they belong:

- *Namespace* (or named schemas): a collection of RDFS class or property names (i.e. graph labels), identified by a URI reference, which can be employed in the description of a knowledge artefact. Given that these names are unambiguously defined in an RDFS schema, classes and properties are universally qualified by their name, which is prefixed by the URI of the namespace (i.e. the schema) they belong. Thus, namespaces provide a standard way of distinguishing among classes and properties that carry the same names in different schemas regardless whether their meaning is the same or not.
- *Graphspace* (or named graphs): a collection of RDF triples (i.e. graph edges), identified by a URI reference, that form the description of a knowledge artefact. Given that these URIs essentially identify RDF/S (sub-)graphs, they can be also employed as source or target resources of properties, so forming complex RDF/S hyper-graphs (i.e. graphs whose nodes are graphs). There are no constraints regarding the contents (disjoiness or containment) of a graphspace: the same (subset of) triples may belong to different graphspaces while graphspaces may be composed of both schema and data triples. Thus, graphspaces provide a standard way for distinguishing among descriptions provided by different actors (or sources), for restricting information access and supporting access control, versioning etc.

Name or graph spaces will be the core mechanism for abstracting from the syntax and semantic peculiarities of the RDF/S data model (see *object exchange service*) in order to support KP-Lab end-user applications dealing with personal and group knowledge spaces involved in teamwork. Their unique URIs unambiguously identifies name or graph spaces produced and consumed in a KP-Lab application space.

4.2.2 Knowledge Storage Tuning and APIs

Several RDF stores have been developed during the last four years for supporting real-scale Semantic Web applications [Theoharis et al 2005]. They usually rely on (main-memory) virtual machine-implementations or on (object-) relational database technology while exploit a variety of storage schemes. Two are the most popular database representations for shredding RDF/S resource descriptions into relational tables: *schema-oblivious* (also called generic or “vertical” representation) and *schema-aware* (also called specific or “binary” representation). Given that the range of KP-Labs applications are quite varied, we need to benchmark the performances of the database storage schemes supported by RDFSuite with respect to the characteristics of the KP-Labs ontologies, descriptions and query/update workload (when a first version of the KP-Lab demonstrators will be available). In particular, we will focus on the scalability of the knowledge repository to support concurrent users under a mix of query and update workloads. Finally, for each of these database storage schemes, we

are currently implementing appropriate APIs for bulk loading and exporting of both resource descriptions and schemata (given a name or graph space) in file streams (RDF/XML or triple-based ascii formats) to provide a simple communication channel with KP-Lab applications not featuring sophisticated SW update and query functionality (see *import/export services*).

4.2.3 Knowledge Query and Update Languages

Several query languages (e.g. RQL[3], SPARQL[4]) have been developed during the last five years for supporting declarative access to ontologies and resources descriptions available on the Semantic Web. One of the unique features of RQL is its ability to match filtering/navigation patterns against RDF/S graphs by taking into account (or ignoring) the semantics (e.g. transitivity of subsumption relationships) of the ontologies employed to describe knowledge artifacts (see [Haase et al 2004] for a detailed comparison of SW QL expressiveness). This functionality is useful for abstracting the technicalities of the RDF/S data model from the end-user KP-Lab applications while it has been efficiently implemented in secondary memory [Christophides et al 2003]. For these reasons we rely on the RDFSuite RQL implementation, to support the SWKM *query service*. We are currently extending RQL filtering/navigation patterns to support graphspaces (namespaces are already supported). Furthermore, there is ongoing implementation of the first declarative language for inserting/deleting/modifying arbitrary RDF/S (or fragments of OWL) (sub)graphs. The language, called RUL [Magiridou et al 2005], ensures that the execution of the update primitives on nodes and arcs neither violates the semantics of the RDF model (e.g. insert a property as an instance of a class) nor the semantics of a specific RDFS schema (e.g. modify the subject of a property with a resource not classified under its domain class). This main design choice has been made given that type safety for updates is even more important than type safety for queries: the more errors we can catch at update specification time the less costly runtime checks (and possibly expensive rollbacks) we need. The rest of the design choices concern (a) the granularity of the supported update primitives; (b) the deterministic or not behavior of the executed sequences of update statements; (c) the smooth integration with an underlying RDF/S query language like RQL. For these reasons we rely on the RDFSuite RUL implementation, to support the SWKM *update service*.

4.3 Knowledge Mediator [ICS-FORTH]

The Knowledge Mediator provides the main registry, discovery and evolution services for KP-Lab knowledge artefacts. It essentially mediates access to and changes of knowledge artefacts by employing personal or group ontologies. These services will be implemented on top of the functionality supported by the knowledge repository and will be used by the end-user KP-Lab applications (developed in WP6) for contextualizing access to knowledge artefacts, as well as enabling to evolve them in a consistent way w.r.t to their intended meaning.

4.3.1 Knowledge Discovery

In order to be able to access and manipulate knowledge artefacts pertaining to their needs, learners will need to firstly locate the ontologies (i.e. schema namespaces) that are used to describe them. If these ontologies do not exist learners should import them

[3] http://139.91.183.30:9090/RDF/RQL/

[4] http://www.w3.org/TR/rdf-sparql-query/

into the repository and start their knowledge retrieval process thereafter. To this end, a Registry of ontologies (i.e. schema namespaces) will be supported (see *respective service*) containing descriptions the subject, scope and purpose of the available ontologies (using a suitable registry schema). Learners will be able to query the registry in order to discover the namespaces (and version IDs) of the ontologies best matching their needs. Once an ontology is found, learners can then access and manipulate the knowledge artefacts which are described using this ontology by issuing queries or updates (see *respective services*). In particular, learners can navigate and browse through the classes and properties of the ontology in order to retrieve or modify the involved knowledge artfacts and this can be done seamlessly for any ontology available in the Registry.

4.3.2 Knowledge Evolution

Knowledge artefacts will be formally described according to the ontologies conceptualizing the domains of interest within the scope of KP-Lab learning applications. The required services will be provided so as to support the evolution of both these descriptions and the related KP-Lab ontologies. The former will enable to insert/modify/delete (sub)graphs of semantic descriptions through the same conceptual navigation/querying metaphor used for accessing the knowledge artefacts (see update service). The latter will enable to consistently revise KP-Lab ontologies even after their deployment. By supporting non-trivial evolution for ontologies expressed in RDF/S or other formalisms with an axiomatic definition, learners have the ability to revise knowledge in a provably consistent manner. The system will verify that the required changes will have the minimum possible impact on the ontology, while learners will have the option to review and verify that the changes proposed by the system are the intended ones (see change service). Once these changes are approved, learners can create a new persistent version of the modified ontology (see versioning service). The importance of services capturing knowledge dynamics lies in their ability to assess the consequences of ontology evolution, and thus involves reasoning about the indented meaning of KP-Lab ontologies while taking into account the axiomatic definition of the employed knowledge representation formalism (e.g. RDF/S). Finally, learners will have the ability to compare two versions of the same ontology (or description graphs) created independently by their peer colleagues (see comparison service).

4.4 Knowledge Matchmaker

The *Knowledge Matchmaker* aims to support advanced interactions of KP-Lab users with knowledge artefacts through appropriate mining and recommendation services employing their semantic descriptions. It essentially enables to match information resources with the employed ontologies as well as knowledge artefacts produced/consumed within a group according to various learners' preferences.

4.4.1 Knowledge Recommendation [LRI-ORSAY]

The objective of this module is to support users accessing the ontology registry by taking into account their preferences in order to provide customized results. This will be done by designing and implementing an Ontology Recommendation and Ranking Service.

The recommendation service [Frej et al 2006] will use a publish/subscribe mode, whereby the users subscribe their preferences at the publish/subscribe service in the form of terms (keywords) describing the ontologies of interest. When a new ontology enters the registry repository the system passes its URI together with its description over to the matching module. The matching module compares then the description to the subscriptions of the subscription repository and notifies (automatically) those users whose subscription matches the description of the new ontology (and only those users). The main issue here is the design of an efficient matching algorithm to support the recommendation service. In somehow more formal terms, a user's subscription can be seen as a user-defined query that is stored by the registry and executed whenever a new ontology enters the registry repository (sometimes also called a "continuous query"); if the new ontology belongs to the answer, then a notification is sent to the user.

The ranking service [Spyratos et al 2005] will use preferences provided online (by the user), together with a query against the registry, in order to rank the results of the query. A preference is an expression of the form s→t, where s and t are terms (keywords) describing ontologies, meaning that ontologies described by s precede those described by t, in the user's preferences. As a consequence, in answering the user query, ontologies described by s should be presented by the system before those described by t. In contrast to a subscription, which is a stored query expressing long-range interests of the user, a query with preferences is an online query expressing current needs and preferences of the user. Its answer is a ranked list of ontology URIs.

4.4.2 Knowledge Mining [TUK, UEP]

Knowledge extraction services will be developed to assist users when creating or updating the semantic descriptions of KP-Lab knowledge artefacts. The semi-automatic generation of these descriptions or even of new KP-Lab ontologies will rely on the textual content or annotations.

Textual resources related to the content or the annotations of various knowledge artefacts, will be analyzed using different text mining techniques. As a result of this process, relevant concepts from the KP-Lab ontologies will be suggested to the users during the formal description of knowledge artefacts. Moreover, unsupervised text mining techniques, such as clustering, will be used to find some unseen concepts (or clusters) in the set of analyzed textual resources. These may lead to, e.g., the suggestion to upgrade existing KP-Lab ontologies, as the knowledge of a KP-Lab team evolves.

The fundamental tasks for the envisioned services are artefact *classification* & *clustering* – grouping of a given set of artefacts into predefined or ad hoc categories and automatic extraction of terms describing key concepts and their relationship. Various kinds of algorithms will be considered for classification - simple term matching, kNN, SVM, Winnow, Perceptron, Naive Bayes (multinomial and binomial), boosting, decision rules, decision trees (various combination of growing and pruning methods) as well as for clustering – kMeans, SOM, GHSOM etc.

5 Architectural Design of the SWKM Prototype

5.1 Overview

The architectural design of the SWKM adopts the SOA (Service Oriented Architecture) principles. As depicted in Figure 5.2, the functionality of the various SWKM logical components described in the previous section is implemented by a series of interrelated services.

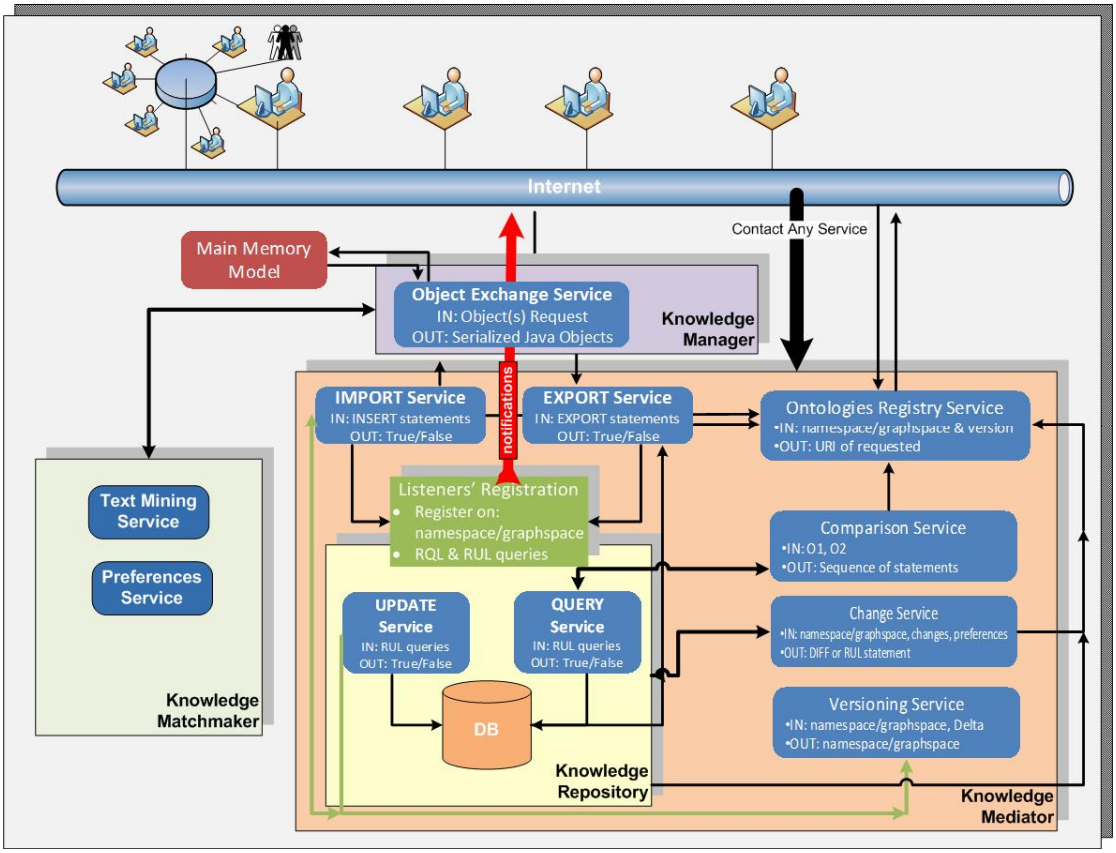

Figure 5.1: The overall SWKM architecture

In terms of availability two major categories of services can be identified:

(1) those available for the M12 prototype (V1.0), namely the:

a. *Query service*
b. *Export service*
c. *Update service*
d. *Import service*
e. *Object Exchange Service*

(2) those implemented in the next versions of the KP-Lab prototype, namely the:

a. *Registry Service*
b. *Comparison service*
c. *Versioning service*
d. *Change service*
e. *Classification service*
f. *Clustering service*
g. *Recommendation Service*

h. Ranking Service

This implementation plan is also imposed by the nature of the services, since the latter are almost based on the former (see Figure 5.1 for service interconnections). KP-Lab applications can directly invoke any of the above services as long as credentials and access rights of the involved users are respected. It should be stressed that in the current design of SWKM, user authentication and authorization is assumed that has successfully taken place in other components of the KP-Lab prototype (see WP4) and thus any request addressed to the middleware can safely be served.

The services described hereafter can be implemented either as Java services to support a tightly coupled interaction among Java based clients and servers (and to be contacted by means of Java based distributed systems and services, e.g. RMI) or as web services that will be based on messaging to transport requests and responses among clients and the services by using e.g. SOAP. Nevertheless the implementation decisions will exactly follow the specifications presented in this chapter.

5.1.1 The Main Memory Model

For the main memory representation of the RDF data model primitives we have designed an object oriented class hierarchy as depicted in Figure 5.2. The classes depict both individual objects (like RDF_Class(-*es*) and RDF_Property(-*ies*), based on the RDF_Resource class) and representation of schemas (RDF_Namespace) and collection of resources (RDF_Graphspace).

The object-oriented hierarchy is based on the generic class named *RDF_Resource*, which essentially corresponds to any resource described in RDF/S. An *RDF_Resource* is described by its URI and a set of possible annotations *like*, *comment*, *label*, *seeAlso*, *isDefinedBy*. In short: *comment* provides a human readable description of the resource, *label* is used for the graphical representation of the schemas and *seeAlso*, *isDefinedBy* provide support for provenance information of class and property definitions. There are four classes that extend *RDF_Resource*, namely *RDF_Class*, *RDF_Property*, *RDF_Container* and *RDF_ClassInstance*. These classes refine *RDF_Resource* with some additional attributes and functionality. These classes include accessors and mutators (where necessary) for each one of their members described below.

RDF_Class represents the “rdfs:Class” primitive. It disposes a Collection of superClasses (ancestors, which we can directly obtain through rdf:subClassOf statements), a Collection of subClasses (descendants, which we can indirectly obtain through rdf:subClassOf statements) and a Collection of properties declaring the class as their domain. Additionally each *RDF_Class* object maintains a Collection of metaClasses of classes (*RDF_MetaClass* object) which are specified in RDF with one or more rdf:type statements. *RDF_Property* class represents the “rdf:Property” primitive. It disposes a Collection of superProperties (which we can directly obtain through rdf:subPropertyOf statements) and a Collection of subProperties (which we can indirectly obtain through rdf:subPropertyOf statements). Each *RDF_Property* object maintains a Collection of *RDF_Class* objects that represents its domain classes and a Collection of *RDF_Class* objects that represents its range classes as they are defined by rdf:domain and rdf:range statements accordingly. Additionally each *RDF_Property* maintains a Collection of metaClasses of properties (*RDF_MetaClass* object) which are specified with one or more rdf:type statements.

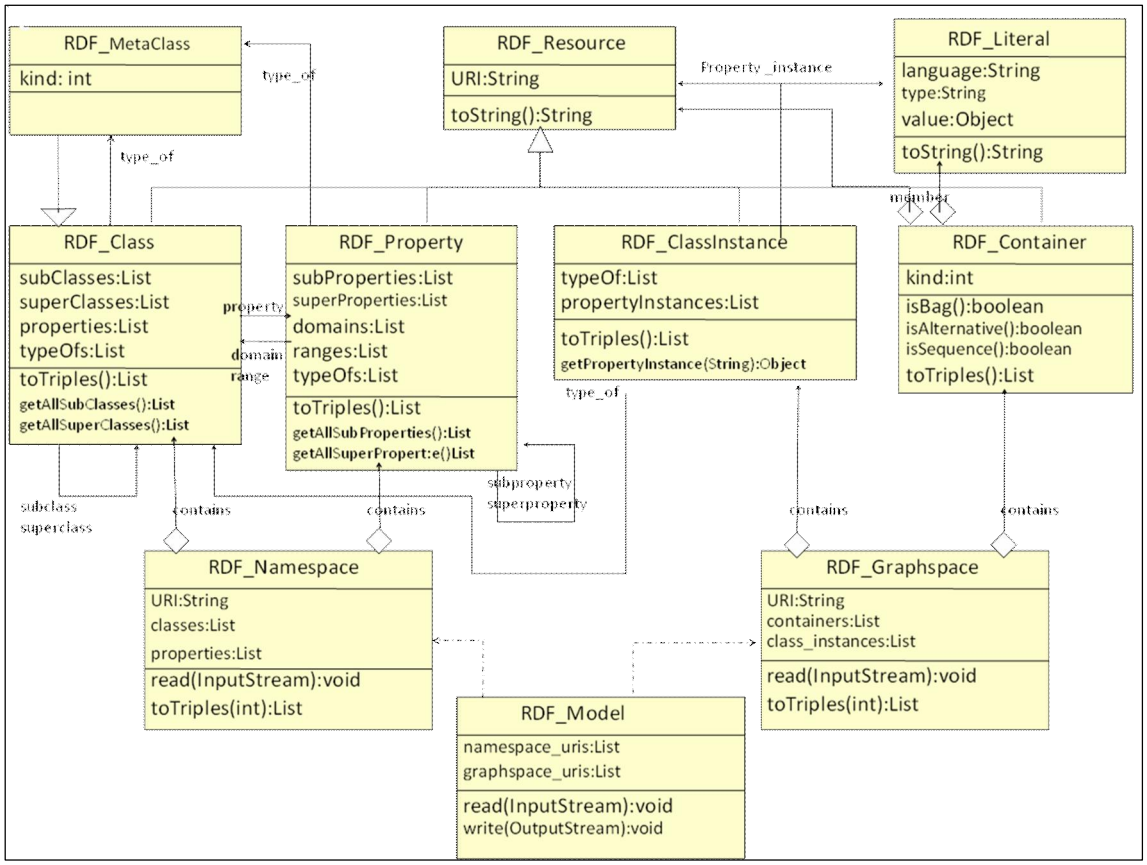

Figure 5.2: Main Memory Representation of the RDF Data Model in UML

RDF_Container class represents the rdfs:container primitive which may be either an rdf:Bag or an rdf:Seq or an rdf:Alt. For this purpose a "kind" attribute is attached to the *RDF_Container* class with value 0,1 or 2 respectively. An *RDF_Container* object maintains a Collection of members that can be either of type *RDF_Resource* or *RDF_Literal* (described later on). In this way, we can represent hierarchies of nested containers at arbitrary depth.

RDF_ClassInstance class represents the actual resources described in RDF. Each such resource may be described as instance of one or more schema classes and these classes are maintained in *RDF_ClassInstance* object as a Collection of *RDF_Class* objects. Each resource can be "connected" to other resources or literal values using instances of the properties of the classes it is instance of. These resources/literals are maintained in *RDF_ClassInstance* class in a Collection named *property_instances* that contains *RDF_ClassInstance* objects and/or *RDF_Literal* objects.

RDF_Literal class represents the rdfs:literal primitive. Since it has a string representation with no URI identity, RDF_Literal is not a subClass of RDF_Resource. From the textual representation of the rdf:literal we can identify three possible parts, it "type", its "value" and its "language". *RDF_Literal* class maintains instance variables for each of these attributes.

In order to represent in main memory RDF/S schema and resource description graphs we have introduced the class *RDF_Model* as a Collection of the Namespaces and Graphspaces by which it is defined. Rather than duplicating the graph nodes or edges of all the consistent Namespaces and Graphspaces, for each Model holds two Collections, respectively with the namespace and graphspace URIs it is composed of. The content of each Namespace and Graphspace is maintained only once by a singleton object and can be accessed on demand by a referring Model. For each namespace there is a class named *RDF_Namespace* that holds a Collection of *RDF_Class* objects and a Collection of *RDF_Property* objects that defines. For each graphspace there is a class named *RDF_Graphspace* that holds a Collection of *RDF_Container* objects and a Collection of *RDF_ClassInstance* objects that defines. An *RDF_Model* object has methods for receiving input from a stream of triples belonging to a given namespace and graphspace or from an RDF/XML file. Additionally methods are provided for generating the triples associated with a model and writing them into a file stream.

Finally, each container class carries the necessary iterators so as to allow iteration over its member collections in order to produce the required RDF triples by using the toTriples() method implemented throughout the hierarchy. This method ensures that the information carried within the classes can be exported to a standard format of RDF triples (e.g. TRIG[5]) so as to be readily useful to other applications.

5.1.2 Listener

As depicted in Figure 5.1 a *Registration* – ***Listening Mechanism*** should be supported by the repository in order to provide the required services in both a consistent and efficient way. This mechanism will be basically responsible for providing the notification means for KP-Lab applications interested in a specific name or graph space. User applications can specify their interest either directly, i.e. by registering themselves as interested on a specific name or graph space or indirectly by asking to retrieve the name or graph space. That way changes on the specific name or graph space will be also part of the listening mechanism and the applications can pole this mechanism to learn if a change has occurred, in which case they can subsequently perform a query to retrieve the object(s) of interest. Otherwise the applications would have to constantly monitor the repository by performing costly queries, which implies an unnecessary communication and processing overhead. Except from taking the burden of the server for such a tedious task, the Listening Mechanism acts also as a single place of convergence where all SWKM service requests dealing with name or graph spaces will be registered.

It is obvious that the corresponding services described hereafter should also take care of notifying the Listening Mechanism for successful queries or updates, so that in turn the application will be notified to undertake the appropriate actions. Note here that it is not certain that an application, which learns that the retrieved e.g. name or graph space has changed, will all the time try to retrieve it again - it just might try to save it as a new version, keep it for personal reference, etc. More precisely, the Update Service will notify the Listener when any of the registered name or graph spaces changes. The applications that use any of the Object Exchange, Query, Comparison, Versioning and Change services, along with the Main Memory Model should also

[5] http://www.wiwiss.fu-berlin.de/suhl/bizer/TriG/

register to the Listener in order to receive the notifications created upon the commit of any change in the repository that affects any of the name or graph spaces they are interested in.

5.2 Knowledge Repository Services

5.2.1 Query Service

The *Query Service* (see Figure 5.3) is responsible for executing RQL queries. The service will return its results in an RDF/XML serialization as a bag of resources or a container of Java objects if a binary representation is requested. The query results can contain both schema and data information from one or several name and graph spaces. So the programming interface for the service will look like:
String query(String RQL_Query);
RDF_Resource[] query(String RQL_Query);

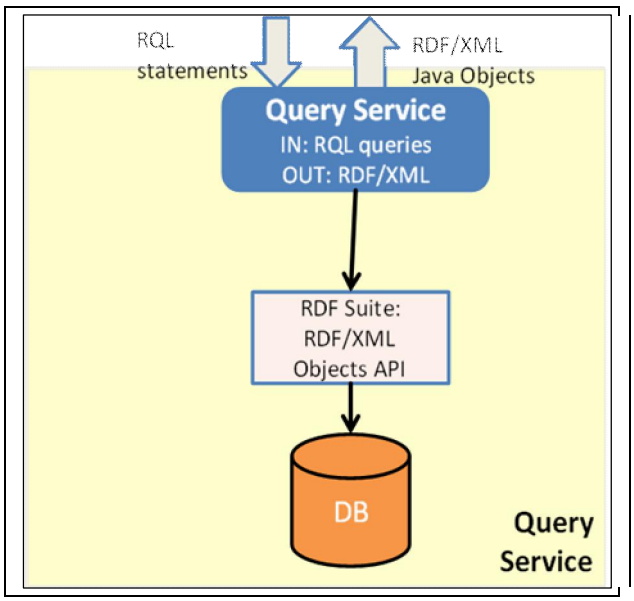

Figure 5.3: The Query Service

The Query Service relies on the *RQL Interpreter* which is used for both parsing and executing the query at hand, as well as on its multithreading capabilities for supporting concurrency control. Since every RQL query posted to the Query Service is essentially a *read transaction* the service also provides the required error handling in case of empty results to notify the application that issued the transaction about the reason of the failure (due to syntax or typing errors).

5.2.2 Update Service

The *Update Service* (see Figure 5.4) is responsible for executing RUL updates involving one or several name or graph spaces. Updating includes construction, modification and deletion of objects in the repository and returns a Boolean value "true" or "false" for successful (commit) or unsuccessful (abort) execution. The programming interface of the query will be:
boolean update(String RUL_Query);

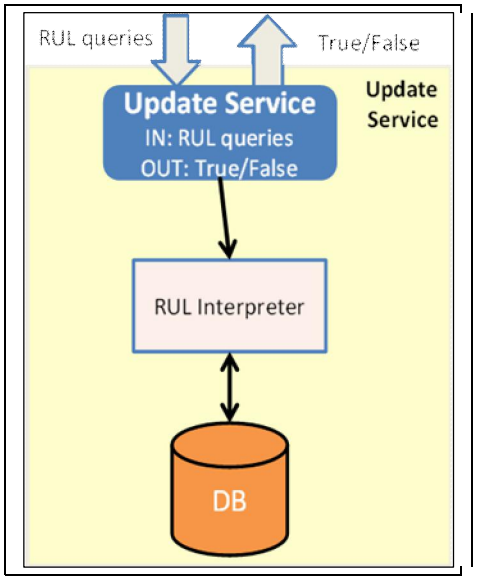

Figure 5.4: The Update Service

The service upon reception of the update will identify the involved name or graph spaces and register it with the Listener as described previously. The Update Service relies on the *RUL Interpreter* which is used for both parsing and executing the update at hand as well on its multithreading capabilities for supporting concurrency control. Also since every RUL update posted to the Update Service is essentially a *write transaction*, the service also provides the required error handling for FALSE results to notify the application that issued the transaction about the reason of the failure (due to syntax or semantic errors). The Update service can be called either by the Object Exchange Service for namespace or graphspace update or directly by the user application for updating any of the RDF objects stored in the repository.

5.2.3 Export Service

The *Export Service* (see Figure 5.5) is responsible for dumping into a file stream (in RDF/XML serialization or TRIG triple-based formats) the contents of the name or graph spaces given as input (along with their version ID). The user of the service needs only to specify which one needs to be exported. The programming interface for the service will look like:
Stream export(List[String] name_or_graphspaceURI, List[String] versionID);

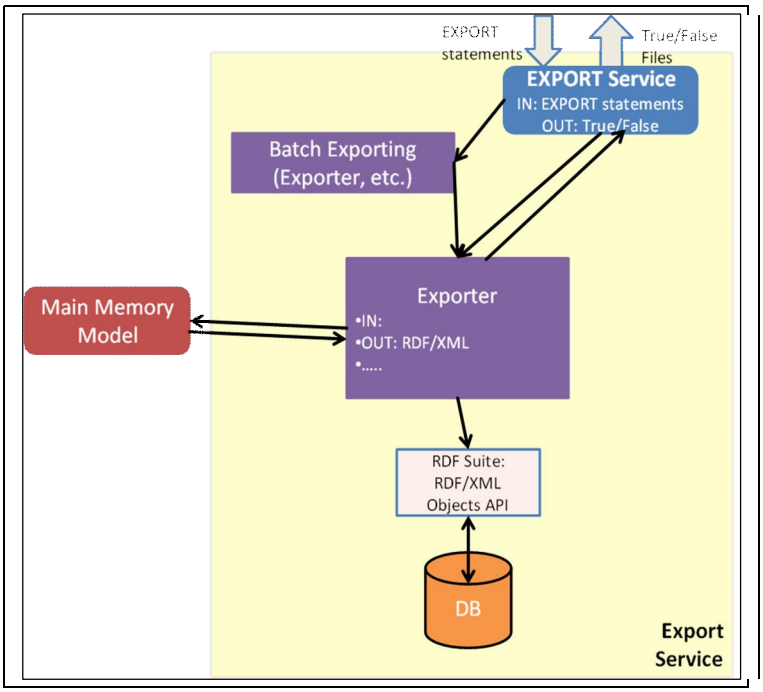

Figure 5.5: The Export Service

The service receives requests either through the Object Exchange Service, where the application is responsible to decide if the requested name or graph space should additionally be loaded into the main memory or directly from the user application where there is no option of loading the requested name or graph space into the main memory.

5.2.4 Import Service

The *Import Service* (Figure 5.6) is responsible for loading the contents of a valid and well formed name or graph space (along with their version ID). This service is responsible for creating the necessary database constructs (tables, relationships, indices) that allow for efficient retrieval and manipulation by the RQL/RUL interpreter. The service can be invoked by providing as input either a file stream (in RDF/XML serialization or TRIG triple-based formats). The output of the service is a TRUE or FALSE statement accounting for successful or unsuccessful completion. The service uses the main memory representation described earlier to firstly load the schema into the main memory before committing it to the repository in order to be able to choose the best possible representation for its storage and perform a series of operations like validation. The schema is afterwards unloaded from the main memory unless it is kept by the service for caching reasons or explicitly kept by the application for easy and fast later usage. The corresponding programming interface will be:
boolean import(String name_or_graphspaceURI, String versionID);

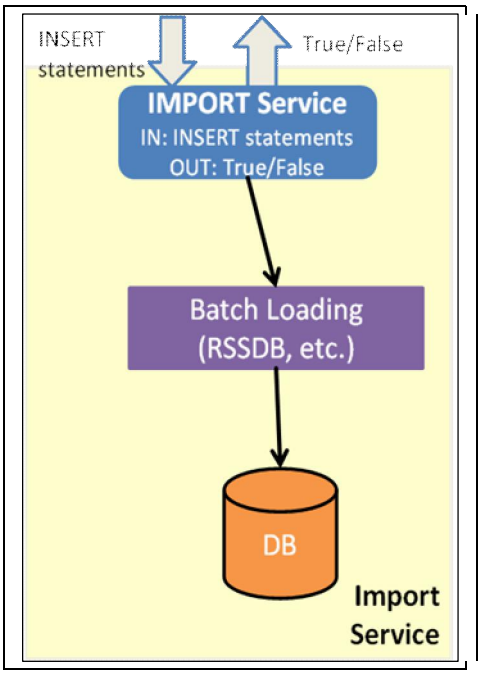

Figure 5.6: The Import Service

In the case of an application tries to import an already loaded name or graph space (identified by the same URI and version ID) the service returns a corresponding error message. The user or the application can accordingly change the namespace URI, invoke the versioning service to create a new version or decide to use the already loaded name or graph space.

5.3 Knowledge Manager Services

5.3.1 Object Exchange Service

The *Object Exchange Service* (see Figure 5.7) enables SWKM to a directly exchange objects of the RDF Model (see section 5.1.1) already fetched in main memory that belongs to a specific name or graph space. By using this service a KP-Lab application can avoid the execution of costly queries in the repository and the reconstruction of the objects in the main memory. Instead it can request the specific objects that it needs and received them in Java serialized binary format, i.e. bytes that can be directly reconstructed as Java objects by the client for immediate consumption. In case where the involved name or graph spaces are not loaded in the main memory the service will contact the Export Service in order to retrieve them from the repository. There will also be a garbage collection process that will unload form the main memory the name or graph spaces that are not used by anyone anymore. Also the service will have the ability to return on request the RDF triples (in TRIG format) of the requested name or graph space. Finally, the service will also provide an interface for storing a changed name or graph space. This will invoke the Versioning Service but it allows the application to directly provide binary objects to be stored in the repository.
The programming interface of the service will be:

Stream fetch(List[String] name_or_graphspaceURIs, boolean BinaryOrTriples);
boolean store(String name_or_graphspaceURI,
Stream changedName_Or_GraphSpace,

String versionID, boolean BinaryOrTriples);

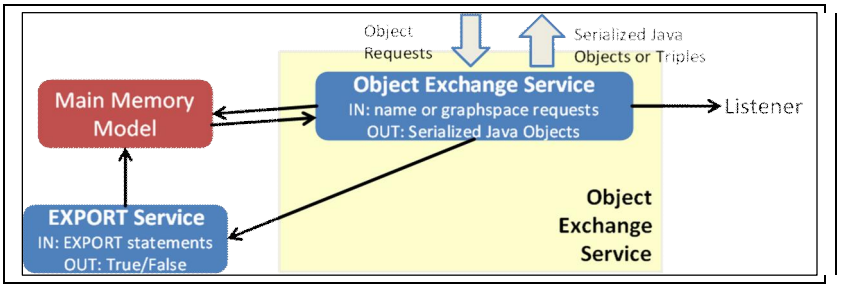

Figure 5.7: The Object Exchange Service

This service to be used for the binary retrieval of the requested objects requires that the client is also implemented in Java so that the binary objects can be directly consumed. This means that it can be implemented (see also paragraph 5.1 for a more general discussion) directly as a tightly coupled Java service and only indirectly as a web service (since transporting binary objects through SOAP messages is currently infeasible).

5.4 Knowledge Mediator Services

5.4.1 Registry Service

This service (see Figure 5.8) is responsible for recording the RDFS schema namespaces (i.e. ontologies) stored in the knowledge repository along with their versions as well as descriptions about their subject, scope and purpose than can be created by employing other RDFS schemas. When invoking this service by passing as input the URI of a namespace it replies with the corresponding list of version IDs that are available in the repository. To record a new RDFS schema the URI of its namespace and the graphspace containing its description is required. Only registered namespaces can afterwards be imported into the knowledge repository. When an inserted namespace is already stored in the repository, a new version will be created by invoking the *versioning service* and the registry will be updated automatically. In both cases the new *VersionID* is returned. Finally, the registry could return the graphspace URI (or a list of graphspace URIs) describing a namespace as long as its URI and version ID are given as input. So the service's programming interface is:
List[String] lookup(String namespaceURI);
String insert(String namespaceURI, String versionID, String graphspaceURI);
List[String]retrieve(String namespaceURI, String versionID, String RQL_Query);

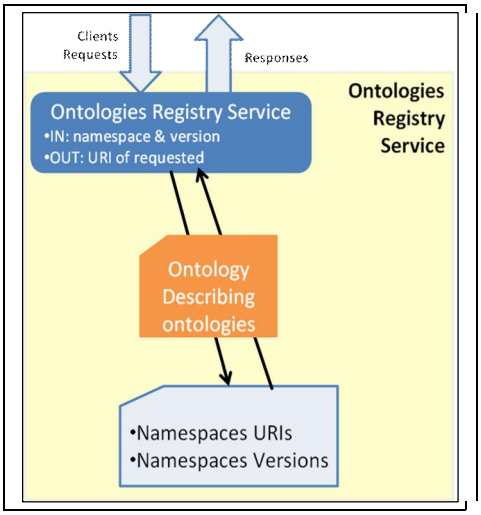

Figure 5.8: The Registry Service

The underlying implementation of this service relies on a dedicated instance of the knowledge repository that will be responsible for retrieving and updating the descriptions of RDFS schema namespaces.

5.4.2 Comparison Service

The *Comparison Service* (see Figure 5.9) is responsible for comparing two name or graph spaces (obviously is meaningless to compare a schema with a description graph) already stored in the repository and compute their delta in an appropriate form (e.g. as triples serialized in TRIG or as a RUL statement). The programming interface will be:
String diff(String name_or_graphspaceURI, String name_or_graphspaceURI);

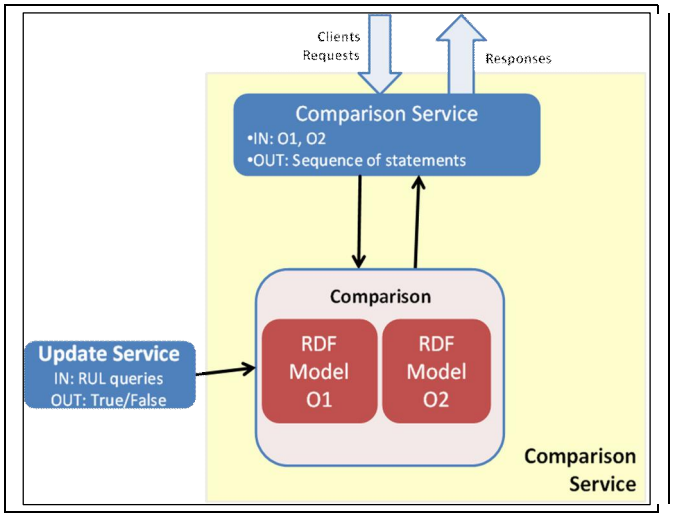

Figure 5.9: The Comparison Service

5.4.3 Versioning Service

The *Versioning Service* (see Figure 5.10) is responsible for constructing a new persistent version of a namespace (or graphspace) already stored in the repository. To

this end it takes as input the original namespace (or graphspace) as well the delta specified in an appropriate form (e.g. as triples serialized in TRIG or as a RUL statement) for constructing a new version. The version ID is returned when the service is successfully executed while all changes are logged and, this change log is made available to the *Registry Service* (which could be used for describing the new version). It is under the responsibility of the service to determine the best possible way to render persistent the new version according to the established version policies. When the versioned name or graph space is already loaded into the main memory and caching is used, then changes could be additionally applied to the corresponding objects of the RDF Data Model. The programming interface will be:
String version(String name_or_graphspaceURI, String Delta);

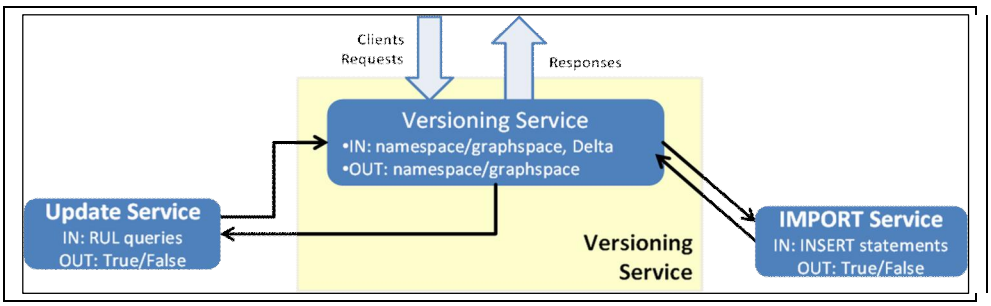

Figure 5.10: The Versioning Service

5.4.4 Change Service

The *Change Service* (Figure 5.11) is responsible for determining the changes that should occur on a name or graph space in response to a change request. The service takes a change request as input and returns series of primitive update operation that capture all the effects (in explicit knowledge) and side-effects (in implied knowledge) of the original change request in a target name or graphspace. As a result, these series of operations have no side-effects and can be easily implemented by the *Update* service (see section 5.2.2). All change requests will describe the necessary changes as a *String Delta* of required actions in an appropriate form (e.g. as triples serialized in TRIG or as a RUL statement). The service will ensure that the requested change will be realized in the target namespace (or graphspace) with the minimal possible impact (changes) upon the original namespace or graphspace, without negating its consistency. If this is not possible, the empty string will be returned (no changes performed). The minimality of a change is determined using some kind of "impact preference ordering" that allows us to compare the impact of different sorts of update operations (e.g. on schema classes and properties as well as their instances). This preference ordering is given as a parameter of the service and plays a critical role in the determination of the actual change meaning. The programming interface of the service will be:
String change(name_or_graphspace URI, String Delta, Preference_Order PO);

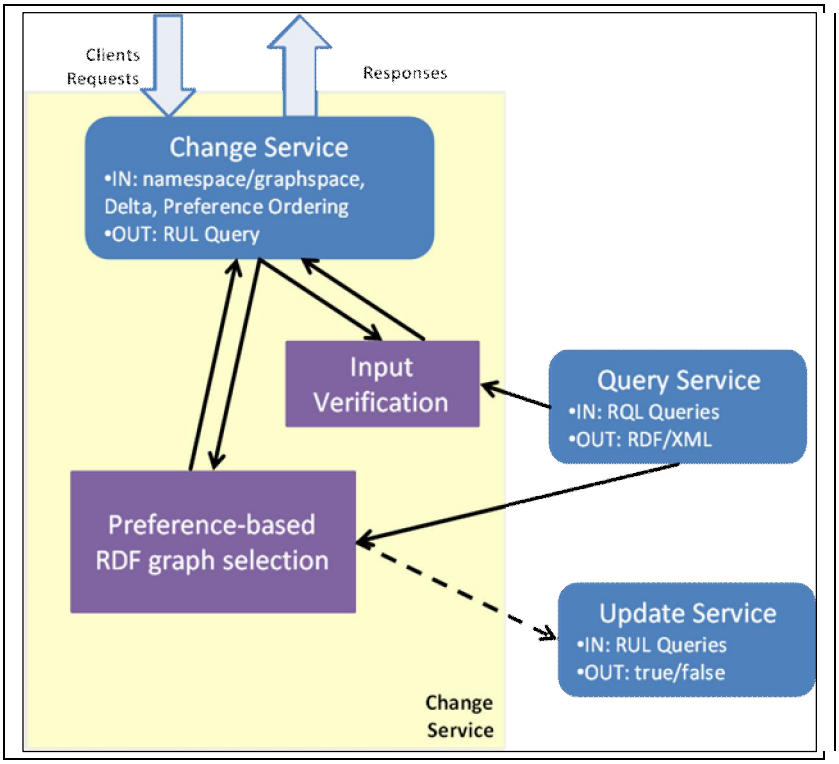

Figure 5.11: The Change Service

5.5 Knowledge Matchmaker Services

5.5.1 Preference Services

This service supports users accessing the ontology registry by taking into account their preferences in order to provide customized results. This will be done by designing and implementing an *Ontology Recommendation Service* and a *Ranking Service*.

5.5.1.1 Ontology Recommendation Service

The recommendation service uses a publish/subscribe mode, whereby the users subscribe their preferences at the publish/subscribe service in the form of terms (keywords) describing the ontologies of interest (see Figure 5.12). When a new ontology enters the registry repository the system passes its URI together with its description over to the matching module. The matching module compares then the description to the subscriptions of the subscription repository and notifies (automatically) those users whose subscription matches the description of the new ontology (and only those users). The programming interfaces of the service will be:
String subscribe(String UID, List[String] terms);
String unsubscribe(String UID, List[String] terms);
String recommend(String event, String namespaceURI, List[String] terms);

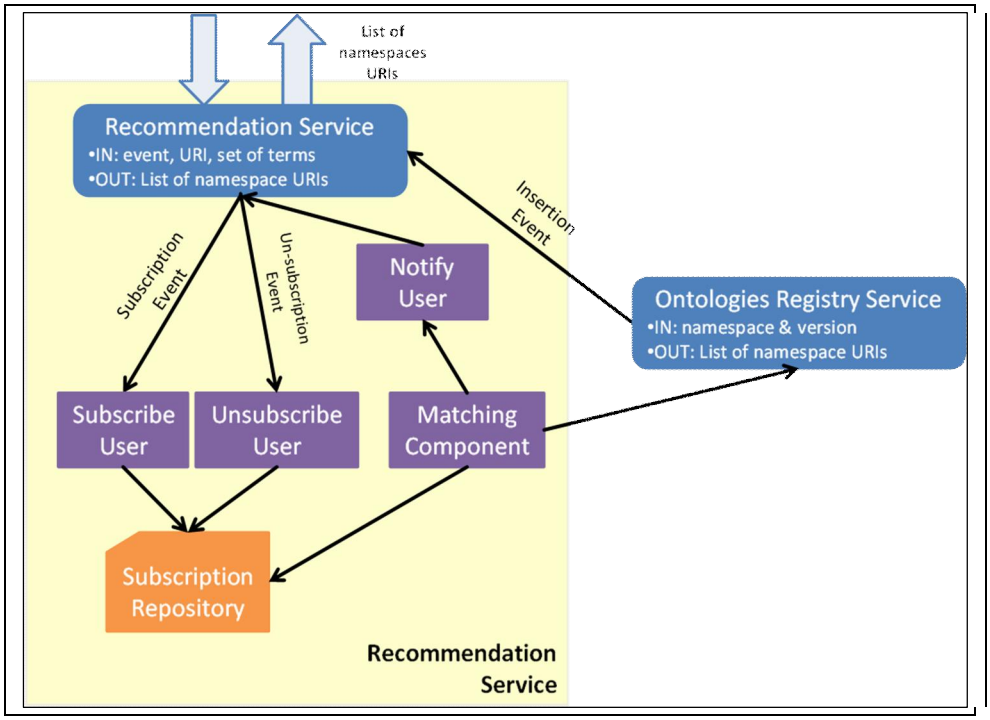

Figure 5.12: The Recommendation Service

5.5.1.2 Ranking Service

The ranking service uses preferences provided online (by the user), together with a query against the registry. A second RQL query is built based on the combination of the provided one and the user preferences and this query is executed against the registry. The answer to the query is a ranked list of ontology URIs that respects the user's preferences. The programming interface of the service will be:
String rank(String RQL_Query, Preference_Order PO);

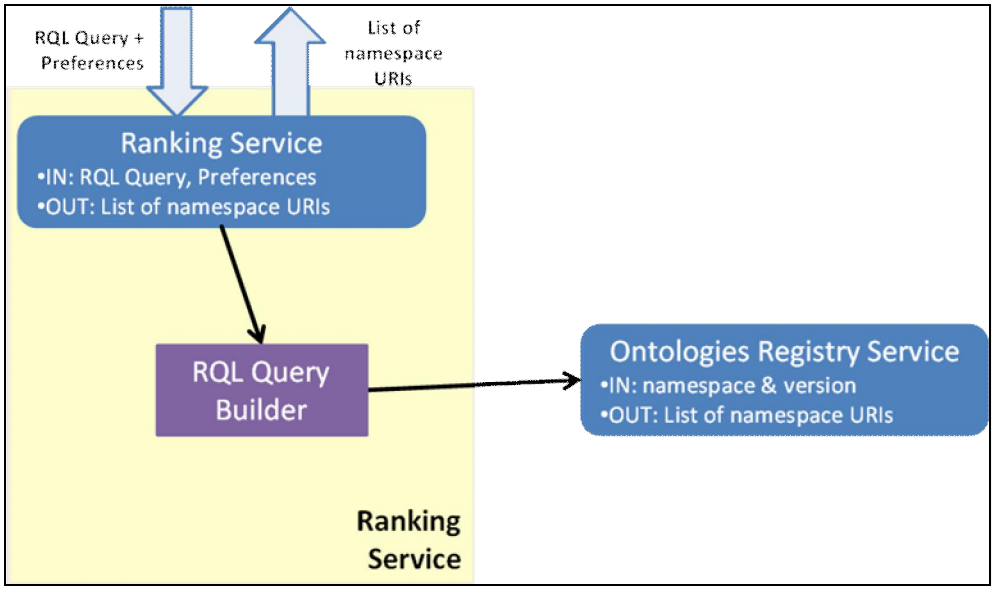

Figure 5.13: The Ranking Service

5.5.2 Text Mining Services

Text Mining (TM) can be defined as a (semi-) automatic extraction of knowledge components (concepts) from plain texts. In the KP-Lab project, the Text Mining Services (as a part of Knowledge MatchMaker) will inspect textual descriptions or annotations of knowledge artefacts to suggest relevant concepts from KP-Lab ontologies, and extend the underlying KP-Lab ontologies (see also in section 0).

The following two basic tasks can be identified for core functionality of the Text Mining Services:
Classification – supervised method. Can be used for classification of artefacts to some pre-defined categories (i.e. ontology concepts).
Clustering – unsupervised method. Can be used to group similar artefacts together.

Derived TM tasks are:
Keyword extraction / summarisation. Can be used to extract keywords from textual descriptions of artefacts. This task can be used to create an initial dictionary for ontology.
Information extraction. Can extract the values of various metadata properties.

The classification task is supervised by a *model*, which is created from a training set. The model contains a set of parameters (weights, rules, etc. – based on the used algorithm) created in the process of training and used in the classification of unknown examples.
Algorithms for Classification:
- simple term matching
- kNN
- SVM
- Winnow
- Perceptron
- Naive Bayes (multinomial and binomial)
- boosting
- decision rules
- decision trees (various combination of growing and prunning methods)

Algorithms for Clustering:
- kMeans
- SOM
- GHSOM

The following services correspond to the Classification and Clustering TM tasks:
1a) *Classification service - Learning* (i.e. creation of mining model):
The service is provided by the *Mining Engine* component and is accessible through the *Mining Service Interface* (see Figure 5.12).
INPUT:
a) training examples - annotated artefacts [plain text + RDF]
b) ontology [RDF, taken from the *Object Exchange Service*]
c) text mining settings [Serialized Java Objects, taken from the *Mining Engine Console*]

OUTPUT: mining model [Serialized Java Objects, passed to the *Mining Object Repository*]

1b) *Classification service - Classification*:

The service is provided by the *Annotation Engine* component and is accessible through the *Recommendation for Annotation Service* interface.

INPUT:

a) description/annotation of an artefact [plain text]
b) mining model + settings [Serialized Java Objects, taken from the *Mining Object Repository*]

OUTPUT: recommendation of metadata/ontology concepts [RDF format]

2) *Clustering service - Clustering*:

The service is provided by the *Annotation Engine* component and is accessible through the *Recommendation for Annotation Service* interface.

INPUT: description/annotation of artefacts [plain text]

OUTPUT: structure of clustered artefacts [RDF format]

The architectural design of the Text Mining Services (TMS), as an integral part of Knowledge MatchMaker functional component of the SWKM, follows the principles of the SOA (Service Oriented Architecture) principles. The structure of particular high-level services and its interconnections are presented on the Figure 5.14.

Functionality of the TM services and components is described in more details in the following paragraphs.

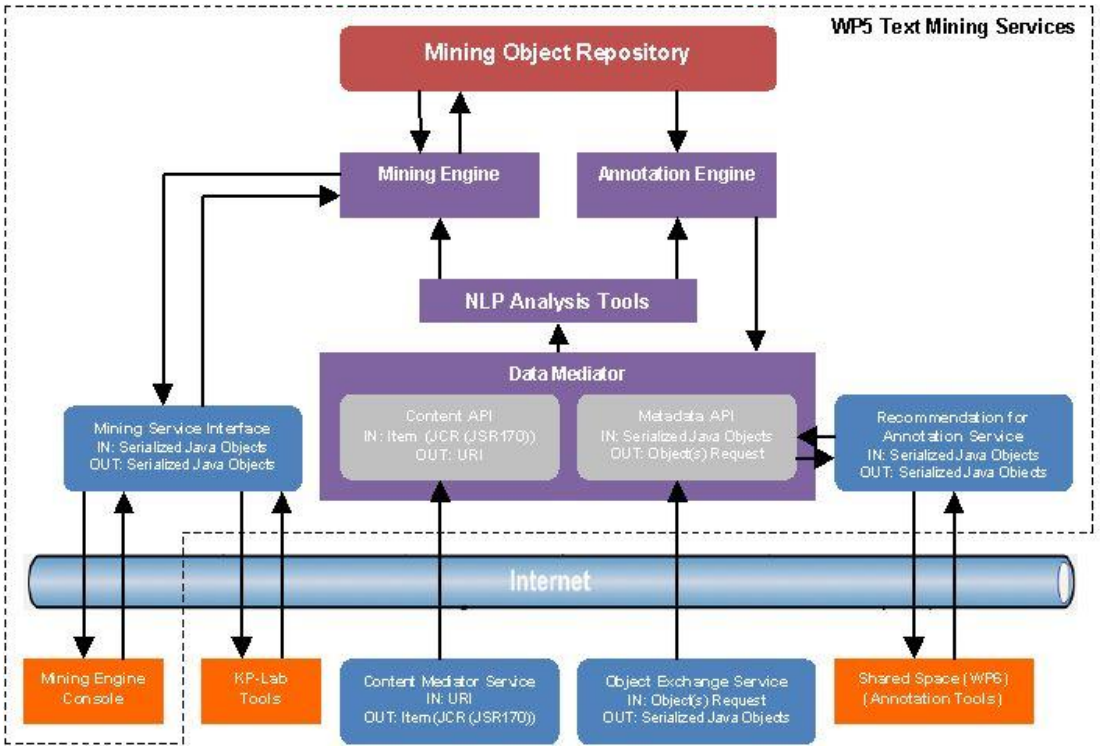

Figure 5.14: Overall Schema of the Text Mining Services

Mining Engine Console is a specialized tool for setting up proper (global) parameters of a mining algorithm in the *Mining Engine* and to inspect and visualize the mining models saved in the *Mining Object Repository*. The *Mining Engine Console* tool should be used by an expert only.

Shared Space (i.e. its Annotation tools, e.g. *Knowledge Artefact Annotation* and *Knowledge Process Annotation*) will use the *Recommendation for Annotation Service* as a supporting tool in the process of annotation, i.e. in the description of textual knowledge artefacts by proper metadata.
On the output, the *Recommendation for Annotation Service* will provide a set of ontology concepts – recommendations for describing an artefact with metadata.

Other external ***KP Lab Tools***, which can use the text mining services, as e.g.:
- a service to share internal model of mining algorithm,
- a service to set up the parameters of algorithm in the *Mining Engine*,
- a service to update model of algorithm in the *Mining Engine*.

These services could be used by the *KP Lab Tools*, which will be designed to inspect and visualize the mining models saved in the *Mining Object Repository*.

Object Exchange Service returns the requested metadata as Serialized Java Object (see section 5.3.1).

Content Mediator Service returns the content as Item (JCR (JSR170)), based on the requested URI[6].

Mining Service Interface will be used by the KP Lab components, which will inspect and visualize mining models, and which will be responsible for setting up parameters of mining algorithms to build new mining models (e.g. by the ***Mining Engine Console***). IN and OUT will be specified according to the Java Data mining API (JSR 73) extended for text mining methods.

Recommendation for Annotation Service is an interface to the *Annotation Engine*. It provides the metadata, which will be proposed to user to annotate given artefact (by specialized *Annotation Tools* in the *Shared Space*).

Content API for querying the *Content Mediator Service.*

Metadata API for querying the *Knowledge Repository* by RQL. The *Object Exchange Service* will be used to query the *Knowledge Repository.*

Data Mediator is an internal component, used by *Mining Engine* and *Annotation Engine* for accessing data by the *Metadata API* and the *Content API* (a synchronization of metadata with the content by URI). This module expects metadata in an uniform format, e.g. as Serialized Java Object in the RDF model from the *Object exchange service*.

Mining Engine (see Figure 5.15) will provide following text mining tasks:
- computing a vector representation of the (textual description of) artefacts,
- building the text mining model,
- testing the built model,
- import and export of the mining model.

---

[6] Using of JSR170 is still open question in WP4. This service could be provided by Transaction / Synchronization services by WP4, which is still a matter of discussion in the WP4.

Input to the *Mining engine* is a set of parameters (settings) specifying the input for building a mining model. The Build Settings may be global (high level), specified for a task (i.e. input data), or specific for particular mining algorithm.
The *Mining engine* will contain an implementation of various algorithms (i.e. NLP algorithms based on the matching of analyzed text, machine learning algorithms for text clustering or classification, etc.). These algorithms can be selected on input (by the input settings) to produce various mining models. The mining model will then contain the essential knowledge extracted from the textual data as determined by the algorithm.

A single type of the model (i.e. linear classifier, or clustering centroid model) can be produced by different algorithms (i.e. SVM, Perceptron or k-Means and Self Organizing Maps). A model can be descriptive or predictive. A descriptive model helps in understanding the underlying data or model behavior. For example, a clustering model that will describe each cluster with the set of characteristic terms can be used to describe a set of artefacts. A predictive model can be an equation or a set of rules that makes it possible to select the set of concepts from the domain ontologies related to the artefact. Mining models are stored in the *Mining object repository* together with all data required to build or evaluate models (i.e. vector representation of the artefact content, statistics about the term/concept co-occurrences, dictionaries, etc.). Stored mining models can be inspected by others KP-lab components by the task of importing the mining model, which will load and retrieve selected model from the *Mining object repository*.

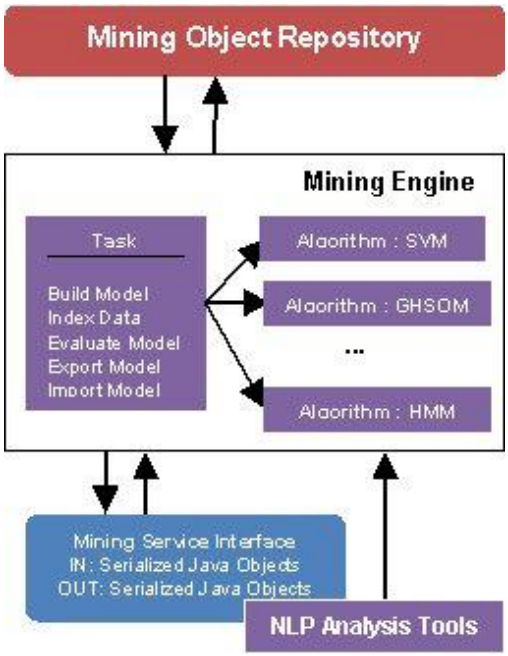

Figure 5.15: Mining Engine

Annotation engine (see Figure 5.16) loads a pre-configured set of mining models and then applies these models to the (already annotated) artefact. Mining models will provide two types of metadata:

- List of selected concepts from the existing domain ontologies, which will be proposed to the user annotating the artefact in a *KP-Lab annotation tool*.

- List of terms (words or phrases) extracted from the artefact content, which will be proposed as candidates for new concepts in a domain ontology. A list of extracted terms can be extended with various types of relations (i.e. broader/narrower, is related to, etc.) between extracted terms mutually, or between extracted terms and the concepts already existing in the domain ontologies.

All the entries in these lists can be weighted. The weight value will indicate a confidence level, estimated by the mining model. All extracted metadata will be provided for the annotation tools as RDF objects defined by *Object exchange service*.

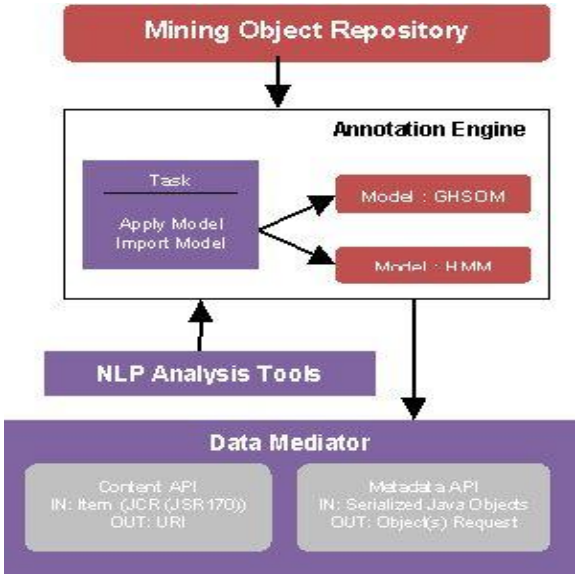

Figure 5.16: Annotation Engine

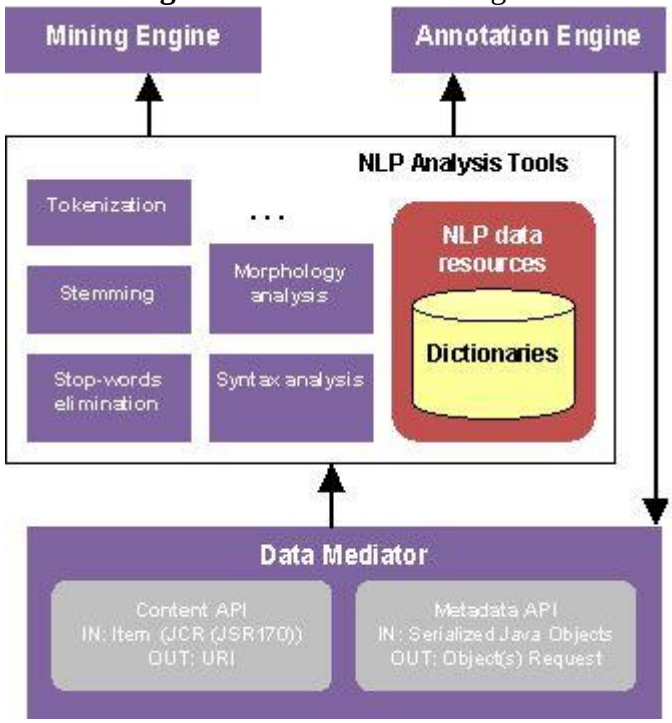

Figure 5.17: NLP Analysis Tools

A set of ***NLP Analysis Tools*** (see Figure 5.17) will be used for text pre-processing. These tools are composed from several modules, which are used by particular NLP techniques and algorithms.
Basic NLP Analysis techniques are:

- tokenization (split of input text to individual tokens),
- stemming,
- elimination of stop words,
- morphological analysis (part-of-speech tagging),
- syntactical analysis,
- semantic analysis, etc.

Modules of the NLP Analysis Tools may vary for different languages (not all modules could be implemented for all languages).

Mining Object Repository is an internal component that contains mining models created by the text mining methods.

Bibliography

[Aberer et al 2004] Aberer, K., Cudre-Mauroux, P., Ouksel, A.M., Catarci, T., Hacid, M.S., Illarramendi, A., Kashyap, V., Mecella, M., Mena, E., Neuhold, E.J., Troyer, O.D., Risse, T., Scannapieco, M., Saltor, F., de Santis, L., Spaccapietra, S., Staab, S., Studer, R.: Emergent Semantics Principles and Issues. In: Database Systems for Advanced Applications 9th International Conference, DASFAA 2004. Volume 2973 of LNCS. pages 14–43, (2004)

[Anderson 1975] Anderson J. R., Cognitive Psychology and Its Implications, 2nd Edition, (New York: W. H. Freeman and Company, 1985); Schank, R. C., "The Structure of Episodes in Memory", in D. G. Bobrow and A. Collins (eds.), Representation and Understanding: Studies in Cognitive Science, (New York: Academic Press, 1975), pp. 237-272

[Athanasis et al 2004] N. Athanasis, V. Christophides and D. Kotzinos: Generating On the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL), Third International Semantic Web Conference (ISWC'04), Hiroshima, Japan, November, 2004.

[Bhatt 2000] Bhatt, G.D. Organizing knowledge in the knowledge development cycle, Journal of Knowledge Management, Vol. 4, No. 1, 2000.

[Bardram 2000] Bardram J. E. Temporal Coordination: On Time and Coordination of Collaborative Activities at a Surgical Department. Computer Supported Cooperative Work 9: 157–187, 2000.

[Chistophides et al 2004] V. Christophides, G. Karvounarakis, D. Plexousakis, Michel Scholl and S. Tourtounis: Optimizing Taxonomic Semantic Web Queries Using Labeling Schemes. Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 1(2), 2004, pp. 207-228;

[Engeström 1999] Engeström, Y. Activity theory and individual and social transformation. In: Engeström, Y., Miettinen, R., Punamäki R.-L. (eds.): Perspectives on Activity Theory. Cambridge Univer-sity Press: Cambridge 19-38, 1999.

[Ernst&Young 1997] Knowledge Management Case Study: Knowledge Management at Ernst&Young, 1997, Available at http://www.bus.utexas.edu/kman/e_y.htm.

[Frej et al 2006] Hanen Belhaj Frej, Philippe Rigaux, Nicolas Spyratos, Matching Algorithms for User Notification in Digital Libraries, Proceedings of BDA 2006, Lille, France, 2006.

[Gruber 1995] Gruber T.R.. Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies, 43(5-6):907–928, 1995.

[Haase et al 2004] Haase P, Broekstra J, Eberhart A, and Volz R. A Comparison of RDF Query Languages. In Proceedings of the 3rd International Semantic Web Conference, Japan, 2004.

[Hutchins 1995] Hutchins, E. Cognition in the Wild. Cambridge MIT Press 1995.

[Karvounarakis et al 2003] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, K. Tolle: Querying the Semantic Web with RQL. Computer Networks and ISDN Systems Journal, Vol. 42(5), August 2003, pp. 617-640. Elsevier Science

[Keenoy et al 2004] K. Keenoy, A. Poulovassilis, V. Christophides, P. Rigaux, G. Papamarkos, A. Magkanaraki, M. Stratakis, N. Spyratos and P. Wood: Personalization Services for Self e-Learning Networks. Fourth International Conference on Web Engineering (ICWE'04), Munich, July, 2004.

[Kotzinos et al 2005] D. Kotzinos, S. Pediaditaki, A. Apostolidis, N. Athanasis and V. Christophides: Online Curriculum on the Semantic Web: The CSD-UoC Portal for Peer-to-peer e-learning. 14th International World Wide Web Conference (WWW'05), Chiba, Japan, May 10-14, 2005

[Kukkonen 2001] Oinas Kukkonen The 7C model for organizational knowledge creation and management 2001.

[Leontiev 1978] Leontiev, A.N. Activity, Consciousness, and Personality. Prentice Hall: Englewood Cliffs, New Jersey, 1978.

[Magiridou et al 2005] M. Magiridou, S. Saxtouris, V. Christophides, M. Koubarakis.RUL: A Declarative Update Language for RDF. In Proc. Of the 4th International Conf. On the Semantic Web, Galway, Ireland, November 2005.

[Magkanaraki et al 2004] A. Magkanaraki, V. Tannen, V. Christophides and D. Plexousakis: Viewing the Semantic Web Through RVL Lenses. Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 1(4), 2004, pp. 359-375

[McGrath et al 1986] McGrath, J. E. & Kelly, J. R. Time and Human Interaction: Toward a Social Psychology of Time. New York: Guilford Publications, Inc., 1986.

[Mika 2004] Mika P. "Ontologies Are Us: A Unified Model of Social Networks and Semantics". In Proc. of the International. Semantic Web Conference (ISWC), 2005, pages 522–536, 2005.

[Nonaka & Takeuchi, 1995] Nonaka, I., Takeuchi, H. The knowledge-creating company, How Japanese companies create the dynamics of Innovation, Oxford University Press, New York, 284 pp. 1995

[Norman 1991] Norman, D.A. Cognitive artefacts , In Designing interaction: Psychology at the human-computer interface, Cambridge University Press, Cambridge, England, John M. Carroll, (Ed.), pp. 17-38, 1991.

[Paavola et al 2004] Paavola S., Lipponen L., and Hakkarainen K. "Models of Innovative Knowledge Communities and Three Metaphors of Learning". Review of Educational Research, 74(4):557–576, 2004.

[Polanyi 1966] Polanyi, M. The tacit dimension. London: Routledge & Kegan, Paul 1966.

[Solomon 2000] It's not just the tool, but the educational rationale that counts. Invited keynote address at the Ed-Media Meeting. Montreal, June 28, 2000. Available at http://www.aace.org/conf/edmedia/00/salomonkeynote.htm.

[Sørensen et al 2002] Sørensen C. & Kakihara M. Knowledge Discourses and Interaction Technology Proceedings of the 35th Hawaii International Conference on System Sciences – 2002.

[Spillers et al 2003] Spillers F. & Loewus-Deitch D. Temporal attributes of shared artefacts in collaborative task environments. Workshop on the Temporal Aspects of Tasks (HCI 2003)

[Spyratos et al 2005] Spyratos N. and Christophides V., Querying with Preferences in a Digital Library, Dagstuhl Seminar (Nº 05182) Federation over the Web, May 2005.

[Stacey 2000] Stacey P. E-Learning & Knowledge Management. E-Learning for the BC Tech Industry October 2000.

[Stahl 2003] Stahl, G. Building collaborative knowing: Elements of a social theory of learning, In What We Know about CSCL in Higher Education, Kluwer, Amsterdam, NL, J.-W. Strijbos, P. Kirschner, & R. Martens (Eds.) 2003.

[Stoyanova et al 2002] Stoyanova N., Kommers P. Learning Effectiveness of Concept Mapping in a Computer Supported Collaborative Problem Solving Design. Journal of Interactive Learning Research, 13(1/2), 111-133, 2002.

[Theoharis et al 2005] Theoharis Y., Christophides V., Karvounarakis G. Benchmarking Database Representations of RDF Stores. In Proc. Of the 4th International Conf. On the Semantic Web, Galway, Ireland, November 2005.

[Vygotsky 1978] Vygotsky, L. Mind and Society. Cambridge, MA: Harvard University Press, 1978.

[Uschold et al 1996] M. Uschold and M. Gruninger, "Ontologies: Principles, Methods and Applications," The Knowledge Eng. Rev., vol. 11, no. 2, pp. 93–136, 1996.

[Zhang et al 1994] Zhang, J., and Norman D.A. Representations in Distributed Cognitive Tasks, Cognitive Science, 18: 87-122, 1994.