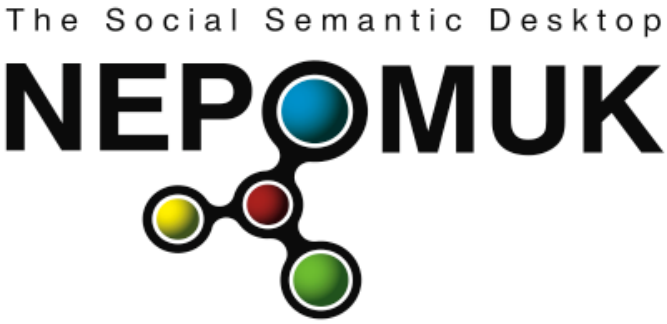
Integrated Project

Priority 2.4.7

Semantic based knowledge systems



Intermediate NEPOMUK Architecture

Deliverable D6.2.A

Version 1.0

03.09.2007

Dissemination level: PU

Nature	R/P
Due date	31.08.2007
Lead contractor	NUIG
Start date of project	01.01.2006
Duration	36 months

Authors

Gerald Reif, University of Zürich
Tudor Groza, DERI, NUIG
Siegfried Handschuh, DERI, NUIG
Mehdi Jazayeri, USI
Cédric Mesnage, USI

Contributors

Gunnar Grimnes, DFKI
Edith Felix, Thales
Knud Möller, DERI, NUIG
Rosa Gudjonsdottir, KTH
Enrico Minack, L3S
Leo Sauermann, DFKI

Mentors

Roman Schmidt, EPFL
Ernie Ong, SAP
Harald Gall, University of Zürich

Reviewers

Paul Chirita, L3S
Stéphane Laurière, Mandriva

Project Co-ordinator

Dr. Ansgar Bernardi
German Research Center for Artificial Intelligence (DFKI) GmbH
Erwin-Schroedinger-Strasse (Building 57)
D 67663 Kaiserslautern
Germany
Email: bernardi@dfki.uni-kl.de, phone: +49 631 205 3582, fax: +49 631 205 4910

Partners

DEUTSCHES FORSCHUNGSZENTRUM F. KUENSTLICHE INTELLIGENZ GMBH (DFKI)
IBM IRELAND PRODUCT DISTRIBUTION LIMITED (IBM)
SAP AG (SAP)
HEWLETT PACKARD GALWAY LTD (HPGL)
THALES S.A. (TRT)
PRC GROUP - THE MANAGEMENT HOUSE S.A. (PRC)
EDGE-IT S.A.R.L (EDG)
COGNIUM SYSTEMS S.A. (COG)
NATIONAL UNIVERSITY OF IRELAND, GALWAY (NUIG)
ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE (EPFL)
FORSCHUNGSZENTRUM INFORMATIK AN DER UNIVERSITAET KARLSRUHE (FZI)
GOTTFRIED WILHELM LEIBNIZ UNIVERSITAET HANNOVER (L3S)
INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS (ICCS)
KUNGLIGA TEKNISKA HOEGSKOLAN (KTH)
UNIVERSITA DELLA SVIZZERA ITALIANA (USI)
IRION MANAGEMENT CONSULTING GMBH (IMC)

Versions

Version	Date	Reason
0.1	25.05.2007	First draft by Tudor Groza
0.8	23.08.2007	version by Gerald Reif
0.9	30.08.2007	Version by Mehdi Jazayeri and Cédric Mesnage
1.0	03.09.2007	Final version by Siegfried Handschuh, review/finalisation by Ansgar Bernardi/Markus Junker

Explanations of abbreviations on front page

Nature
R: Report
P: Prototype
R/P: Report and Prototype
O: Other

Dissemination level
PU: Public
PP: Restricted to other FP6 participants
RE: Restricted to specified group
CO: Confidential, only for NEPOMUK partners

Executive summary

This deliverable presents the current progress in the architectural committments of the NEPOMUK project. It builds upon the outlines, approaches, and goals documented in deliverable D6.1 and serves as basis for further discussion and as an intermediary step towards the final NEPOMUK architecture.

The NEPOMUK architecture assembles the different understandings in the project to reach a common description of the structure, abstractions and relations of the social semantic desktop main aspects.

After introducing the terminology used in this deliverable, we present the methodology used to merge the main knowledge flows of the project. The NEPOMUK engineering cycle combines the activities required to achieve the project goals. We present the scenarios and methodology used to formalise the case study requirements and the resulting abstracted list of functionalities.

The main results of this document are :

- A middleware-based architecture supporting the integration of components,
- A service-oriented modularization based on standard communication techniques,
- Ontological support for social-semantic-based Middleware.

The security implications and requirements of the social semantic desktop are discussed and explained. We give a possible scenario and architectural solutions.

We combine and discuss the relations between the current implemented components, the architecture services and the studied functionalities. We analyse the missing parts of the architecture and implementation and give some explanations and possible solutions.

We conclude by looking at the current issues, evaluation and next steps of the architecture.

Table of contents

1 Introduction

The aim of the NEPOMUK project is to provide a standardized description of a Social Semantic Desktop architecture and to realize reference implementations and evaluated use cases.

In traditional desktop architectures, applications are isolated islands of data - each application has its own formats, unaware of related and relevant other applications.

The problem on the desktop is similar to that on the Web. On the Web we face isolated data islands, on the desktop there is no standardized approach to find and interact between applications. The Social Semantic Desktop paradigm adopts the ideas of the Semantic Web paradigm. Formal ontologies capture both a shared conceptualization of desktop data and personal mental models.

The objective of this document is the presentation of the current NEPOMUK Social Semantic Desktop architecture. It serves as a communication bridge between the case study user requirements and the NEPOMUK technical partners. The scenarios developed in the project are abstracted in a list of functionalities. The developed components are formalised as a set of services. We analyse, compare and relate these two approaches to create the NEPOMUK architecture.

This deliverable follows from deliverable D6.1 and presents a more detailed and comprehensive view of the NEPOMUK architecture as currently shared by the consortium. We follow a systematic process to ensure the integration of all partners visions.

The deliverable is structured as follows. We start by recalling key terminology (Sect. 2) and the followed methodology (Sect. 3). We describe the developed case study scenarios (Sect. 4) and the functionalities abstracted from them (Sect. 5). We then present the architecture(Sect. 6) in terms of ontologies, communication, services and components. Security requirements and the architectural implications (Sect. 7) are discussed next. We finish the document by analysing the current status (Sect. 8) of the architecture and offer some conclusions and next steps.

2 Terminology

We try to use as much as possible common terms to facilitate the understanding of the document by the reader. The following definitions are taken from the Oxford Dictionary.

Middleware – software that occupies a position in a hierarchy between the operating system and the applications, whose task is to ensure that software from a variety of sources will work together correctly.

Service – a set of unified related functionalities published under one interface. A service can be implemented by a composition of components or by a single component.

Component – a part or element of a larger whole, e.g. a piece of software.

API – interface description of a service.

Registry – software that provides the means for registering, unregistering and discovery of services.

Architecture – the conceptual structure and logical organization of a computer or computer-based system.

Application – a program or piece of software designed and written to fulfill a particular purpose of the user.

Extension – a part that is added to something to enlarge or prolong it; a continuation.

Functionality – the purpose that something is designed or expected to fulfill.

Scenario – a postulated sequence or development of events.

Persona – the aspect of someone's character that is presented to or perceived by others.

Usage – the action of using something or the fact of being used.

3 Methodology

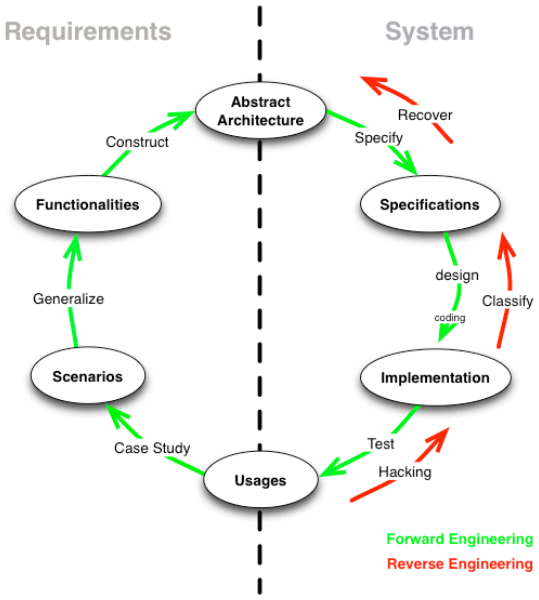


Figure 1: NEPOMUK Engineering Cycle

In a classical engineering project, engineers start with a set of requirements and build a system to satisfy those requirements. In a project aimed at developing innovative research-oriented artifacts, one can hardly start with a well-defined set of requirements. On the other hand, on the implementation side one usually uses an exploratory development method. In the case of building the NEPOMUK architecture we considered two important aspects: (i) existing software developed by the partners, such as P-Grid (the peer-to-peer system) [1], Beagle++ search engine [6] or Aperture [3] — the metadata extraction system, and (ii) the usability research held with the case studies partners, performed by interviewing potential users of the Social Semantic Desktop (SSD). To reconcile these two aspects, we need a combination of top-down and bottom-up approaches. We have developed a methodology based on an engineering cycle (Figure 1). This cycle represents the way in which we intend to integrate the existing technologies (a bottom-up approach) with the needs coming from the users (a top-down approach).

Clockwise, Figure 1 shows a forward engineering cycle. We analyzed the end-user's intended **usage** of the SSD, studied the different use cases and

formulated them into **scenarios**. We generalized the individual scenarios and extracted the common **functionalities** that make up the SSD. These functionalities formed the basis to define the reference **architecture** which in turn lead to the service **specification** and **implementation** that is evaluated with the end-users. Working in parallel, partners already started the implementation of certain components that are likely to be needed by the SSD. Therefore, we reverse engineered these components to get their specifications and used the gained experience and knowledge when defining the architecture. Reverse engineering follows a counter-clockwise path through the cycle.

The resulted NEPOMUK architecture represents a confluence of three different building blocks: (i) the requirements and objectives present in the vision of the SSD, (ii) the functionalities extracted from user studies (forward engineering), and (iii) the service specifications of existing implementations (reverse engineering). This confluence provides a verification of the architecture, built on a shared understanding of all partners involved in the project. We see it as a perfect roadmap towards the realization of the SSD.

In the following, we go into details in some of the parts of the engineering cycle, by presenting real-world scenarios that we considered as being particularly representative for the SSD paradigm and show the list of functionalities abstracted from the user study material.

4 Scenarios

Before we move on to the specific functionalities that a Social Semantic Desktop supports and discuss how they are implemented, we first present scenarios which illustrate what an SSD is, how it is used, and how it changes the way we do knowledge work. We present the methodology used to create these scenarios (Sect. 4.1). We chose the scenarios which illustrate the different dimensions of an SSD: Sect. 4.2 describes example usage showing the use of semantics on the desktop, and Sect. 4.3 shows the social dimension of an SSD, i.e. the interaction between desktops of different users. The scenarios give an overview of what is possible and how the SSD presents itself to the user.

4.1 Methodology

Figure 2: NEPOMUK Personas

The study of user needs regarding collaboration on the SSD is a major goal of the NEPOMUK project. User studies were carried out in the project at the case study partner sites, companies and research labs working in the area of business software, biomedical research, Linux development, and management consulting. The type of work performed varies between the case study partners. In each of them, employees are knowledge workers, receiving, interpreting and structuring information on a daily basis. The purpose of the user studies is to understand the work environment in order to develop a SSD that meets the knowledge workers' needs and requirements. Forty contextual interviews [5] and seven video brainstorming workshops [14] were performed with employees at the different partner sites. To document the resulting user requirements 14 personas and 40 usage scenarios [7, 8] were created, illustrating the user needs, desires and expectations on the SSD. Personas are fictitious persons that represent different user groups and are always based on data collected in user studies. A persona and a related scenario where the persona uses the SSD represents an effective way to illustrate how the users want the SSD to operate. Fig. 2 depicts all the personas created in NEPOMUK.

4.2 The Semantic Dimension

A typical use of a single Semantic Desktop is to organize ones data: files, emails, pictures, etc. Users are able to tag those information resources with concepts from a network of ontologies. The ontologies also contain relations (or properties, to use RDF terminology), which can be used to link information resources on the desktop. Organizing information resources helps to find information faster and aid users in their daily work.

When a user begins to use the Semantic Desktop, often-used concepts and properties are already present. E.g., there are basic concepts such as `Person`, `Meeting` or `Place`, and properties such as `knows` or `located-in`. We assume useful tools, such as a geographical ontology, are already in place.

As the need arises, users extend the existing ontologies—e.g., they add concepts for a particular meeting they attend or people they know, such as `Executive-Committee-Meeting-07/06/07`, `Jane-Doe` or `Hans-Schmidt`. The following two scenarios give examples of Semantic Desktop usage. We use two imaginary users (*personas*[1]): Dirk, who works in a research group for some company in Germany, and Claudia, who is the group leader and his boss. Both Dirk and Claudia work on a project called *Torque*.

Organizing Pictures *(Annotation)*

Dirk just got back from his holidays in Norway, where he took a lot of pictures. Using his Semantic Desktop, he now wants to organize them, so that he can later find the pictures he wants to easier, generate photo albums for particular topics, etc. A lot of the important concepts probably already exist on his desktop, such as `Norway`, or the cities he has visited: `Oslo`, `Bergen` and `Trondheim`. Other concepts will be added by Dirk himself, such as `Holidays-in-Norway-2007` and tourist sights like `Preikestolen` or `Holmenkollen`. Since these concepts are more than just tags, Dirk can also say things *about* them, e.g., that `Holidays-in-Norway-2007` was a `Trip` and took place in `2007`, and that `Preikestolen` is a `Location` in `Norway`. Dirk managed to take a picture of Prince Håkon and Princess Mette-Marit, he creates two more concepts `Håkon` and `Mette-Marit`. There are many ways in which Dirk can link (or *tag*) his pictures to the relevant concepts—however, part of the Semantic Desktop are intuitive user interfaces, which hide most of the intricacies that go on under

[1] Within the NEPOMUK project, these personas were created by distilling typical users from a series of interviews and evaluations with our use-case partners.

the hood from the user. E.g., Dirk might have an application that shows all the concepts that he is interested in in the form of a tag cloud. Linking the pictures would then simply require him to drag them onto the desired concept in the cloud.

Planning a Trip *(Context)*

Later, Dirk finds out that he has to go on a business trip: a conference in Oslo. The Semantic Desktop assists him in planning and organizing this trip, through the notion of *context*. Dirk creates a new `Trip` object `Trip-to-OOC2007-Oslo` and tells his desktop that he is now in the context of this trip. This means that everything he does from this moment on will be interpreted as happening in that context, until he quits the context again. He books a flight in his Web browser, the destination field is automatically filled with "Oslo", similarly the departure field. When he books a hotel room, he is assisted similarly. Dirk receives a number of confirmation emails, such as the flight itinerary and booking confirmation for his hotel. These emails and their attachments are automatically associated as belonging to the `Trip-to-OOC2007-Oslo` context, so that Dirk can easily find them again later. Once he knows his exact flight dates and where his hotel will be, he enters this information into his calendar, which is also context-aware and remembers that these entries belong to Dirk's trip.

4.3 The Social Dimension

Users will have a lot of benefit from just using the Semantic Desktop on their own. However, by connecting to others, a number of additional possibilities arise.

Assigning Tasks in a Group (*Social Interaction*)

In the previous scenario, Dirk found out he had to go on a business trip. In fact, he found out about this because he was notified by his boss Claudia, who also uses a Semantic Desktop. Claudia plans to travel to the OOC2007 conference in Oslo to present a research prototype her group has developed as part of the *Torque* project. She doesn't want to travel alone, so she first needs to find out which of her group members are free when the conference is on. Through the network of Social Semantic Desktops, her calendar application has access to the calendars (or parts of them) of all her contacts. She can ask the calendar to give her a list of all people in her group (`My-Research-Group`) who are working on the *Torque* project (`Torque-Project`) and are free when OOC2007 is on. She finds out that Dirk is free at the desired time. Just like Dirk in the previous scenario, she creates a `Trip-to-OOC2007-Oslo` object and makes it her current context. She also links the trip to the `Torque-Project`. Now, she creates a new `Task` object `Dirk-Prepare-Trip-To-OOC2007`, with a subtask `Dirk-Prepare-Presentation-Slides` and sends an email to Dirk, asking him to accompany her to the conference, book flights and hotel rooms, and prepare slides for the conference presentation. Her email and the task will of course be automatically linked to the proper context. Also, in this version of the scenario, Dirk no longer has to create the `Trip-to-OOC2007-Oslo` object himself—instead, it will be added to his Semantic Desktop automatically when he gets Claudia's mail.

5 Functionalities

In order to integrate the requirements expressed in the scenarios and other materials produced in the case studies we need to use a more formal approach. All the material was processed by a group of members of the project

Desktop	Annotation, Offline access, Desktop sharing, Resource management, Application integration, Notification management
Search	Search, Find related items
Social	Social interaction, Resource sharing, Access rights management, Publish/Subscribe, User group management
Profiling	Training, Tailor, Trust, Logging
Data Analysis	Reasoning, Keyword extraction, Sorting and grouping

Table 1: Functionalities of the Social Semantic Desktop.

coming from different areas: developers, case study partners, architects and usability designers. The results of the Lugano Functionality Workshop is an homogeneous list of functionalities required to satisfy the scenarios. For each functionality, we provide a name, a short textual description, inputs, outputs and the relevant material in which this functionality was discovered. We grouped these in five categories as shown in Table 1.

Desktop

At the *desktop* level, the semantic functionality common to most applications is the ability to add information about any resource. **Annotation** comprises the facilities to store and retrieve semantic relations about anything on the desktop. When Dirk annotates his photos from his trip, he does it from his most favorite photo application (such as Picasa or iPhoto), the annotations are then stored by the SSD. We name this functionality **Application integration**; applications interact with the SSD by means of different services. When Dirk got notified about the trip to Oslo, this was an example of **Notification management**. The SSD handles different kinds of mechanisms such as emails, syndication, or text messaging. When Dirk creates a new concept or even a new file on the SSD, the application he uses interacts with the **Resource management** facilities of the SSD, creating the needed semantics according to the current context and setup. Some of the information Dirk needs when booking his trip are stored on Claudia's desktop. If she is not connected to the network, the **Offline access** facility exports the relevant information to another desktop. **Desktop sharing** is the ability for different users of the SSD to work on the same resources. Claudia might write a report of the trip together with Dirk: the resource management is done on Dirk's desktop, but Claudia can access and edit it remotely.

Search

The semantic network created on the desktop enables a whole new way of searching on the SSD. **Search** uses the semantic relations as well as social relations to retrieve relevant items. Once an item is found a user can also **find related items**. For instance, when Dirk searches for a flight to Oslo, he can also search for related items and may find documents related to the upcoming meeting.

Social

The SSD provides different means of **Social interaction**, e.g., the embedding of semantic information in emails or text messaging, or the ability to annotate another user's resources. Some desktop level functionalities such as desktop sharing and offline access require the SSD to enable **Resource sharing**. When Dirk and Claudia collaborate on the trip's report, Dirk might make it accessible to the whole group by adding it to a shared information space. When sharing resources or information on the network, the **Access rights manage-**

ment of the SSD provides ways to define specific rights relations between users, groups and resources. The SSD's **User group management** system makes it easy for the rapid creation of new groups from a list of users. These groups can then be used to modify access rights or for resource sharing in a shared information space. E.g., some of Dirk's friends may have subscribed to get notifications of new pictures that Dirk annotates and makes available. The **Publish/Subscribe** mechanism of the SSD facilitates the creation of feeds of relevant information.

Profiling

If enabled, the **Logging** functionality of SSD logs user activity, which helps to detect the current user's context. The *profiling* of the SSD can be done automatically by **Training**: the SSD learns to predict the user's behavior. The user can still **Tailor** the SSD's intelligent behaviors: some learned contexts can become irrelevant and may need to be re-adapted or removed. The notion of **Trust** on the SSD between people or information sources is also a result of the profiling of the desktop. Dirk might define that he trusts Claudia's information, or Claudia's SSD might learn that Dirk is a trustworthy source of information regarding the *Torque* project.

Data Analysis

To support the training behaviors of the SSD or querying related items, the SSD provides different *data analysis* mechanisms such as **Reasoning**. For instance, when Dirk tags a picture with `Preikestolen` and `Norway`, the SSD may infer that Preikestolen is in Norway. This information can later be reused for search. **Sorting and grouping** supports applications that perform search. The SSD returns items from many sources and people; sorts and groups these items regarding different criteria, using the semantics defined on these resources. The **Keyword extraction** from resources such as text resources is useful for automatically tagging or summarizing.

6 Architecture

In this section we introduce the reference architecture for a SSD. Some key aspects of the architecture are:

- The architecture is middleware-based as a middleware approach supports integration of components;
- The architecture is service-oriented to support the integration of independent new components based on standard communication techniques;
- The architecture is based on ontologies to support the use of semantic web technologies.

The architecture reflects the two aspects of the scenarios introduced in Sect. 4, i.e., the semantic (which can operate on a single desktop) and the social aspect (which is relevant in a network of desktops). To cover these requirements and the functionalities discussed in Sect. 5, the SSD is organized as a Service Oriented Architecture (SOA). Each service has a well defined WSDL (Web Service Definition Language) interface and is registered by the *Service Registry*. The social aspect of sharing resources over the network is enabled by the peer-to-peer (P2P) infrastructure of the architecture. Fig. 3 depicts the architecture that was developed within the NEPOMUK project and shows the main services that contribute to the SSD. In the following we present the services of the SSD.

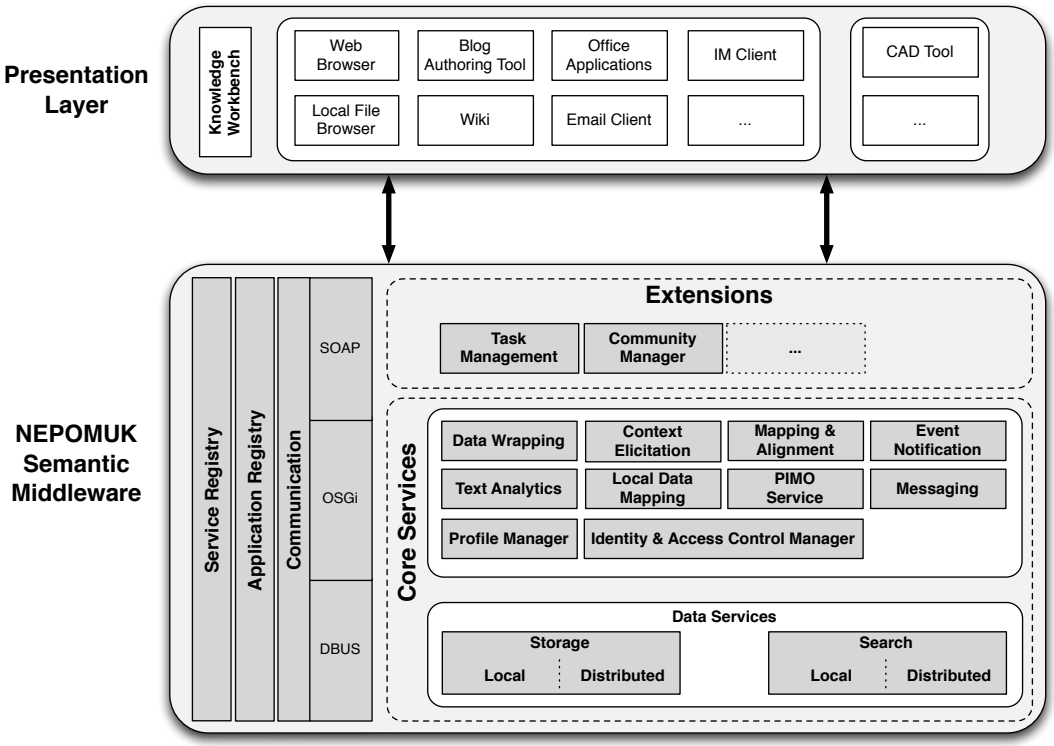


Figure 3: NEPOMUK Architecture

6.1 Overview

The architecture, as shown in Fig. 3, is organized on two layers. Although not present in the figure, like current desktop systems, the desktop environment builds on top of the *Operating System* core, such as the file system, kernel, and network environment. On the SSD the desktop environment is pooled in the *Social Semantic Desktop Middleware* Layer (SSD Middleware). The SSD Middleware groups the services of the SSD to be used by the *Presentation Layer*, which provides the user with SSD enabled applications that take advantages of the functionalities of the SSD.

The SSD is made up f individual desktops, which are organized in a P2P fashion. To support the communication between the peers, the SSD Middleware provides P2P network communication services. To enable information sharing between individual desktops, the RDF metadata of shared resources is stored in the *distributed index* of the P2P system[2].

The Middleware is split into three parts: (i) the Core Services, (ii) the Extensions, and (iii) the Middleware Registries (for Service and Application).

The *Core Services* of the SSD Middleware comprise the services providing basic SSD functionalities. These services are accessed via the SSD Application Programming Interface (API). If a developer wants to exploit the SSD Core Services to build his domain-specific application, he creates an *extension* of the SSD Middleware. An example of such an extension is the *Task Management* which provides functionalities such as creating, delegating, and manipulating tasks. Both core services and extensions are detailed in Sect. 6.4.1.

The *Service Registry* provides the means for registering, unregistering and

[2]In NEPOMUK, the P2P system is based on GridVine [2], which in turn is built on top of P-Grid [1] and provides a distributed index query supports.

discovery of the services part of the SSD Middleware. Finally, the *Application registry* allows applications from the Presentation Layer to register call back methods at the SSD Middleware if they need to be notified by SSD services, e.g., when a message arrives and has to be displayed to the user in an Instant Messaging Client.

The top layer of the architecture is the presentation layer. It provides a user interface to the services provided by the SSD, and is built using the SSD API. Many desktop applications are possible sources for resources that should be managed by the SSD. Therefore, a desktop application can be SSD-aware if it integrates support for the SSD Middleware. Since this assumption does not hold for most of the current off-the-shelf applications, Nepomuk plug-ins and add-ons may be used to enable a seamless integration with existing applications. These plugins extract email, calendar or document data and add them as resources of the SSD. Within NEPOMUK we develop dedicated applications that make use of the SSD API directly, such as a semantic *Wiki* or *Blogging Tools* [15].

The *Knowledge Workbench* is the central place to browse, query, view, and edit resources and their metadata. In this way the Knowledge Workbench aims to replace current file management tools such as the MS File Explorer. If the SSD is extended by usage extensions, the application programmer also has to provide the corresponding user interface in the Presentation Layer (*e.g.,* for Task Management, Community Management, *etc.*).

6.2 Communication

An advantage of using a service oriented architecture is the availability of standard tools. We use the industry standard SOAP (Simple Object Access Protocol) for exchanging messages between our components. For traditional applications the names and structure of SOAP messages is specified using the Web Service Description Language (WSDL), which in turn uses XML schema data-types to specify the form of the objects being exchanged. However, since the formal modeling of the target domain using ontologies is the core idea of a Semantic Desktop application, the best-practices for SOAs are slightly different. In this section we will discuss some important differences from a traditional SOA system.[3] Basing a system architecture on underlying domain ontologies is similar in nature to Model Driven Architectures (MDA)[4]. However, on the SSD, ontologies take the place of UML models.

6.2.1 Working with RDF

As described in [20], we invested substantial effort into the modeling of our domains as ontologies in a formal language. These ontologies give us a very powerful and flexible modeling language, although the structure of instances of such ontologies at first sight seem much more constrained than complex XML schema data-types, the flexibility of RDF introduces some additional requirements for developers of components that should handle RDF instances:

- The structure of the RDF instances received may not be fully known at design time. This means one must take care that the code does not

[3] Here, we make the assumption that a modern object-oriented programming language like Java will be used for implementation, but observations and solutions are equally valid for most other languages.

[4] MDA: `http://www.omg.org/mda/`

break when encountering unknown properties in the data, and these unknown properties must be preserved. In general, programming services for the Semantic Desktop is like programming services for the web, rather than for traditional desktop applications, and one should follow the general rule of web-programming: “Be strict in what you send and tolerant in what you receive.”

- Conversely, other services might not be aware of all the properties the local service uses. Therefore each service must be programmed to be tolerant of missing data and do their best with the data that was provided.

6.2.2 Passing Instances in Messages

Normally, when using SOAP in connection with WSDL and XML schema for data modeling, some mapping is used that will convert the XML schema definition to class definitions in the programming language of choice. Stubs and skeletons are generated for the service themselves, so that the details of communication are hidden. Programming against remote services is then indistinguishable from programming against a local object. However, when using services that pass instances for which the structure is defined by ontologies, the mapping is not so straight forward. We identify three alternatives for programming web services where parameters are instances of ontology:

1. Starting with the ontologies, a number of tools[5] can be used to create a set of Java classes from the ontologies. The service interface is written using parameters of these types, and another tool is used to generate the WSDL and associated XML schema types from these. By sharing the URIs of the concepts in the ontologies with the URIs of the XML schema types, the semantics of messages is retained. The benefit of this approach is that much of the SOAP technology is retained, existing tools may be reused. Developers not familiar with Semantic Web technology, find that developing and using these services is unchanged from a normal Java environment. The main problem with this approach comes from the fact that ontologies are in general more dynamic than Java class definitions. In particular, we expect the personal information models to change frequently. This approach requires a complete re-run of the whole tool chain and a recompile of the system when an ontology changes, as well as introducing some constraints on the ontologies.

2. On the other end of the spectrum it is possible to bypass the parameter passing of SOAP all together, and rely more on the Semantic Web technology. Each method offered by a service will take a single RDF document (possibly including several named-graphs), and all the details about the message are given in these RDF graphs. An additional ontology for messages and parameters must be constructed, and some named-graph aware serialization (e.g., TriG or TriX[6]) of RDF is used to construct the XML SOAP messages. This approach was, for instance, used in the SmartWeb project.[7] The benefit of this approach is that the effort that has gone into modeling the ontologies is not duplicated for modeling objects. Also, the full expressivity of RDF may be used when modeling, as it is not required that the instances fit into another representation. The backside to this flexibility is that it is significantly harder to

[5] RDFReactor: `http://wiki.ontoworld.org/wiki/RDFReactor`; RDF2Java: `http://rdf2java.opendfki.de`; Elmo: `http://openrdf.org`, etc.

[6] TriG/TriX: `http://www.w3.org/2004/03/trix/`

[7] SmartWeb: `http://www.smartweb-project.de/`

program with RDF graphs than with simple Java objects, and both service developers and consumers need good knowledge about RDF. One can of course envisage new tools that facilitate programming with such RDF messages, but since all the interesting details are hidden inside the RDF parameter, existing SOAP tools for development or debugging are no longer useful.

3. Finally, a hybrid approach of the two methods is possible. Here each method retains multiple arguments, but each argument is represented by an RDF resource. We envisage two possibilities to realize this: either each parameter is given as a (*named-graph-uri*, *uri*) tuple pointing into an RDF document given as a special parameter; or, alternatively, each parameter is in itself an RDF graph plus the URI of the actual parameter (each RDF graph may contain several resources). The benefit of this method is that the changes in the ontology do no longer require a recompile of the system, while at the same time allowing slightly more compatibility with existing SOAP tools. The problem with this method remains that both client and server programmers need in-depth knowledge of RDF and the ontologies being used.

Regardless of which of the three alternatives one chooses, it remains an important issue to make sure that the formal description of the services (i.e., the WSDL+XML Schema definitions) remains semantically correct and retains the pointers to the ontology concepts which the parameters represent. As mentioned, the first approach is handled by well chosen URIs for the XMLSchema types. The second and third approaches the parameters have the form of simple *string* objects in both the WSDL definition and the SOAP messages, since the RDF serialization is represented as a string. However, both versions of WSDL available at the time of writing allow extensions to the WSDL format itself,[8] and additional constraints on the type or form of the RDF instances contained inside the string parameters may be specified here. This is the approach taken by the *Semantic Annotation for WSDL and XML Schema* (SAWSDL) working group[9], and the NEPOMUK project uses this standard.

6.3 NEPOMUK Ontologies

In Section 6 we introduced the service oriented nature of the NEPOMUK SSD. In the next sections we discuss in detail the two layers we grouped the services in the NEPOMUK Semantic Middleware depicted in Fig. 3: (1) the core services and (2) the extensions. Before we detail on these two layers, we recapitulate the layered structure of the NEPOMUK ontology pyramid introduced by the NEPOMUK Ontology task force and discuss in which way the ontology pyramid is related to the layered NEPOMUK architecture.

Ontologies form a central pillar in Semantic Desktop systems, as they are used to model the environment and domain of the applications. The common definition of an ontology is “a formal, explicit specification of a shared conceptualization” [11].

We differentiate between four levels of ontologies for the SSD: *Representational*, *Upper-Level*, *Mid-Level* and *Domain*. The main motivation for this layering is that ontologies at the foundational levels are more stable, reducing the maintenance effort for systems committed to use them. A core principle of the Semantic Desktop is that ontologies are used for personal knowledge man-

[8] Language Extensibility in WSDL1: `http://www.w3.org/TR/wsdl\#_language` and in WSDL2: `http://www.w3.org/TR/wsdl20/\#language-extensibility`

[9] SAWSDL: `http://www.w3.org/TR/sawsdl/`

agement. Each user is free to create new concepts or modify existing ones for his *Personal Information Model*. This modeling takes place on the domain-ontology level, but the user is of course free to copy concepts from the other layers and modify them to fit his or hers own needs. In order of decreasing generality and stability the four layers are:

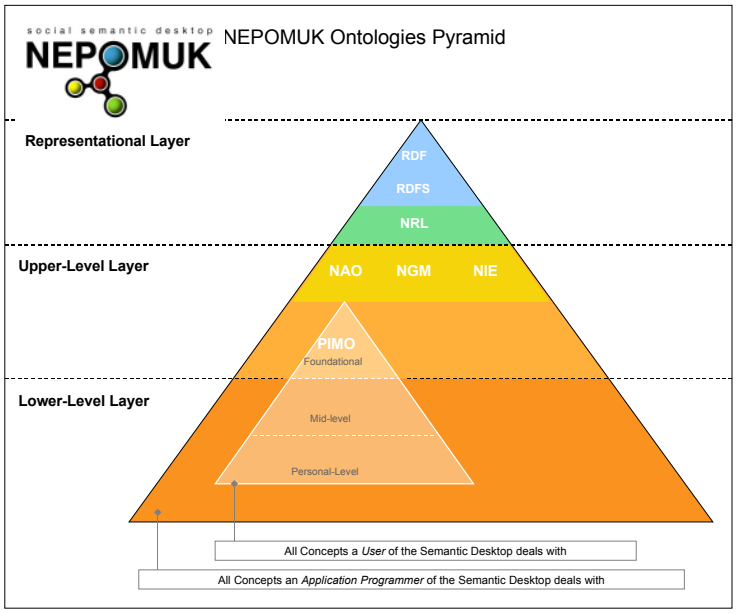


Figure 4: Nepomuk Ontology Pyramid

Representation(al) Ontology

Representational ontologies (i.e., ontology definition languages) define the vocabulary with which the other ontologies are represented; examples are RDFS and OWL. The relationship of a representational ontology to the other ontologies is quite special: while upper-level ontologies generalize mid-level ontologies, which in turn generalize domain ontologies, all these ontologies can be understood as instances of the representational ontology. Concepts that might occur in the Representional Ontology level include: classes, properties, constraints, etc.

Upper-Level Ontology

"An upper ontology [...] is a high-level, domain-independent ontology, providing a framework by which disparate systems may utilize a common knowledge base and from which more domain-specific ontologies may be derived. The concepts expressed in such an ontology are intended to be basic and universal concepts to ensure generality and expressivity for a wide area of domains. An upper ontology is often characterized as representing common sense concepts, i.e., those that are basic for human understanding of the world. Thus, an upper ontology is limited to concepts that are meta, generic, abstract and philosophical. Standard upper ontologies are also sometimes referred to as foundational ontologies or universal ontologies." [19] In the upper-level ontology you will find concepts like: `Person`, `Organization`, `Event`, `Time`, `Location`, `Collection`, etc.

Mid-Level Ontology

"A mid-level ontology serves as a bridge between abstract concepts defined in the upper ontology and low-level domain specific concepts specified in a domain ontology. While ontologies may be mapped to one another at any level, the mid-level and upper ontologies are intended to provide a mechanism to make this mapping of concepts across domains easier. Mid-level ontologies

may provide more concrete representations of abstract concepts found in the upper ontology. These commonly used ontologies are sometimes referred to as utility ontologies." [19] The mid-level ontologies may include concepts such as: `Company`, `Employer`, `Employee`, `Meeting`, etc.

Domain Ontology

"A domain ontology specifies concepts particular to a domain of interest and represents those concepts and their relationships from a domain specific perspective. While the same concept may exist in multiple domains, the representations may widely vary due to the differing domain contexts and assumptions. Domain ontologies may be composed by importing mid-level ontologies. They may also extend concepts defined in mid-level or upper ontologies. Reusing well established ontologies in the development of a domain ontology allows one to take advantage of the semantic richness of the relevant concepts and logic already built into the reused ontology. The intended use of upper ontologies is for key concepts expressed in a domain ontology to be derived from, or mapped to, concepts in an upper-level ontology. Mid-level ontologies may be used in the mapping as well. In this way ontologies may provide a web of meaning with semantic decomposition of concepts. Using common mid-level and upper ontologies is intended to ease the process of integrating or mapping domain ontologies." [19] Domain ontologies consist of concepts like: `Group Leader`, `Software Engineer`, `Executive Committee Meeting`, `Business Trip`, `Conference`, etc.

Fig. 4 shows how these four layers relate to the four ontologies created and used in the NEPOMUK Project. We were hesitant to make use of OWL for the representational ontology level in NEPOMUK. OWL assumes an open-world view of all data, and this seems inappropriate for the local Semantic Desktop. Instead, we would like to be able to combine a closed-world view of the local desktop with the open-world of the network of Semantic Desktops or even with the Web itself. To achieve these goals we developed the NEPOMUK Representational Language [20] (NRL), which defines an extension to the semantics offered by RDF and RDFS; the main contribution of NRL is the formalization of the semantics of named graphs[10]. NRL allows multiple semantics (such as open and closed world) to coexist in the same application, by allowing each named graph to have separate semantics. The NEPOMUK Annotation Ontology (NAO) is a basic schema for describing annotations of resources, this is essentially a formalization of the tagging paradigm of Web 2.0 applications. A specialized part of NAO is the NEPOMUK Graph Metadata schema (NGM) which allows the description of named graphs, defining meta-data properties such as the author, modification dates and version data.

Finally, the NEPOMUK Information Elements ontology (NIE) contains classes and properties for describing objects found on the traditional desktop, such as files (Word documents, images, PDFs), address book entries, emails, etc. NIE is based on existing formats for file meta-data such as EXIF for image meta-data, MPEG7 for multimedia annotations, ID3 for music files, iCal, and others.

6.4 NEPOMUK Middleware

Similar to the ontology pyramid, the NEPOMUK Semantic Middleware is organized in layers. However, opposite to ontology engineering, where the levels are organized top down, in Software Engineering layered architectures are built bottom up. Lower layers provide generic services to the layers above. The higher layers rely on these services and combine them to more complex and problem specific services, which in turn are offered to layers above or via the user interface of the applications to the user.

[10] `http://www.semanticdesktop.org/ontologies/nrl/`

In Fig. 3 we already introduced the service oriented NEPOMUK architecture. Within the NEPOMUK Semantic Middleware, the services are organized into the Core Services and the Extensions. The core services make up the basic functionalities of the SSD. The extensions take advantage of these core services and combine them into complex and user specific services such as task management or community management.

Since the core services are dealing with the basic functionality of the SSD, these services are operating on the more general concepts defined in the Upper-Level and Midd-Level Ontology. On the other hand, the Extensions deal with domain specific concepts, which are defined in a Domain Ontology. This results in a situation where lower layers of the architectures deal with the upper-level concepts of the ontology pyramid and vice-versa.

6.4.1 Services

A basic feature of the SSD is the possibility to store semantic meta-data and to query the stored data. Since we are talking about the *Social* Semantic Desktop, users are able to share meta-data over the network. The meta-data we handle are distributed over the computers of individual users. To enable this social aspect the *Data Services* are based on a peer-to-peer network.

The *Data Services* are responsible to control the insertion, modification, deletion, and retrieval of resources on the SSD. A resource can be a user, a document, a calendar entry, an email, and so on. It provides a service to store the RDF meta-data in the *Local Storage*. Resources and their RDF descriptions can either be manually added to the SSD, or the *Data Wrapper* or *Text Analysis* service extract the information from desktop applications such as email clients or calendars. Data Wrappers are used to extract metadata form structured data sources (*e.g.,* email headers, calendar entries, *etc.*). In NEPOMUK, data wrappers are implemented based on Aperture [3]. The Text Analysis service is used to extract metadata from unformatted text (*e.g.,* email bodies, word processing documents, *etc.*). For local queries and for offline working the RDF metadata is stored in the *Local Storage*. If a resource is shared with other users in an information space, the meta-data is also uploaded to the distributed index of the underlying peer-to-peer system. The *Search* service can either issue a *local* search in the local storage or a *distributed* search in the underlying peer-to-peer system.

Before new metadata is added to the repository, we check whether this metadata describes resources that are already instantiated (*i.e.,* a URI has been assigned) in the RDF repository. In this case, the URI of the resource is reused, rather then creating a new one. This process is known as information integration [4]. The *Local Data Mapper* service takes over this responsibility in the SSD Middleware. E.g., the Data Wrapping service extracts contact information from the address book and stores the metadata in the repository. Since this is the first time information about the contacts is added to the RDF repository, a new URI is generated for each person. If later the Data Wrapping service extracts information from an email header, the Local Data Mapping service is responsible to lookup whether information about the sender of the email is already in the repository and reuse the corresponding URI instead of creating a new one [18].

Ideally only one ontology exists for a domain of interest such as contact data, calendar events. In reality, however, we are faced with many ontologies of (partly) overlapping domains (*e.g.,* FOAF and vCard for contact data, or different personal information models). Since individual users share information over the SSD, it is likely they use different ontologies for their annotations even

when talking about similar domains. The SSD Middleware provides a *Mapping & Alignment* service that is used by other Middleware services and services in higher layers to translate RDF graphs from a source ontology to a target ontology.

The SSD Middleware logs the actions a user performs on the resources on his desktop. The logged data is stored in the Local Storage and analyzed by the *Context Elicitation* service to capture the current working context of the user. The context is used to adapt the user interface or to suggest meaningful annotations to the user, depending on the task they are currently working on.

As discussed in Sect. 6, the services on the SSD use RDF to exchange data. Therefore, services need the capability to generate and process RDF graphs. However, the handling of RDF graphs is more difficult than the instantiation and manipulation of objects in an object-oriented programming language. To simplify the handling of the RDF graphs, the *PIMO Service*[11] provides an easy way to create and manipulate concepts in RDF graphs.

The *Event Notification* service enables other services or human users to register their interest in a change of the shared information space. For example a user wants to be notified when the status of a certain report changes from "draft" to "final" or several actions have to be triggered in the local task management extension when a new task has been assigned to the user. Ideally such a service would be based on a fully fledged publish/subscribe system that is build on top of a distributed event based system. In NEPOMUK we currently do not plan to implement a RDF enabled event based system. Instead we are following a simpler polling strategy. The Event Notification service allows for registering SPARQL[12] queries that describe the kind of change in the shared information space that should be observed. In addition a call-back method is registered that should be triggered when the change has been detected. The Event Notification service checks the registered notification requests (subscriptions) in regular time intervals or on request.

The *Messaging* service provides means to exchange messages between NEPOMUK Desktops over the network. Other services can register a call-back method and the type of message they are interested in. There will not be a fixed set of allowed message types. We think of message types for extensions to enable task management services to directly communicate with each other. A different message type can be used to build an instant messaging service. The Messaging service will be implemented on top of the Jabber Instant Messaging and Presence technology [16].

The *Application Registry* allows applications from the Presentation Layer to register call-back methods at the SSD Middleware if they need to be notified by SSD services, e.g., when a message arrives and has to be displayed to the user in an Instant Messaging Client.

The realization of the *Profile Management* and the *Identity and Access Control Manager* Services is still under discussion. Sect. 7 summarizes the current state of the discussion and give an outlook on possible solutions.

6.4.2 Extensions

The Core Services of the SSD Middleware provide basic functionalities of the SSD. These services are accessed via WSDL interfaces of the individual services. The set of all WSDL interfaces of the Core Services is the NEPOMUK Middleware API. If a developer wants to exploit the SSD Core Services to build

[11] PIMO: Personal Information Model - Ontology for personal information management
[12] SPARQL: SPARQL Protocol and RDF Query Language - an RDF query language

his domain-specific application, he creates an *Extension* of the Middleware.

Within the NEPOMUK project, the case study partners provide us with use cases for Extensions. SAP builds an Extension to implement the Task Management System. This Extension provides functionalities such as creating, delegating, and manipulating of tasks. The domain ontology for this case study is the Task Management Ontology (TMO) introduced in D3.1. Mandriva builds an Extension to develop the Help Desk Community introduced in D11.1 and TMI bases their knowledge management business cases as Extension, as discussed in D9.1 on the Core Service.

6.5 Implemented components

The NEPOMUK architecture integrates with existing components developed by the NEPOMUK partners. In this section we briefly present the components and map them to services of the NEPOMUK Middleware. We further give references to further information about each component.

Distributed Storage and Search

The Distributed Search and Storage component allows users to share and locate content in the NEPOMUK community. It is based on P-Grid [1], a structured P2P overlay network for search and storage utilities, which are further enhanced with semantics via GridVine [2], to offer a distributed index with RDQL query support.

The component provides four services that are used by other services in the middleware: (1) The distributed index is used to advertise metadata that is shared between NEPOMUK desktops, (2) the distributed storage that replicates local resources to remote desktops in order to increase their availability even when the owner is unavailable, (3) the distributed search to query the distributed index for a range of results; and (4) the lookup service can be used to find out the IP address when the peer-ID is known. Since every NEPOMUK desktop has its own ID this service can be used to retrieve the IP address for a direct file transfer or communication session.

Detailed information:

- D4.1 Distributed Search System - Basic Infrastructure: Dec 2006 (EPFL, L3S) - submitted and accepted
- `http://nepomuk.semanticdesktop.org/xwiki/bin/view/Main/Comp%2DDistributedIndex`
- `http://dev.nepomuk.semanticdesktop.org/wiki/sp/DistributedSearch`

Local Repository

The RDFRepository is the central metadata and structured data store in Nepomuk. It contains all information from the filesystem from the DataWrapper, ontologies, and the Personal Information Model (PIMO) of the user. All Data that Nepomuk handles locally should be kept here, to benefit most from data integration.

All Desktop applications can use the RdfRepository database to locally share metadata about resources within a single NEPOMUK desktop, and through this sharing it is possible to integrate them on the level of data and ontologies. Applications can query the repository and provide novel interlinked views on the user's information that are not possible on current desktop systems.

Detailed information:

- `http://dev.nepomuk.semanticdesktop.org/wiki/RdfRepository`

Data Wrapper

The DataWrapper component wraps data from native applications in the presentation layer, such as files in the filesystem or emails in an email client. Datawrappers extracts the data from various applications, converts the data to RDF and stores it in the RdfRepository. Currently the data-wrapper does not offer the functionality of writing data back into the applications. The DataWrapper can be configured to crawl (i.e. access all datasources) at a given interval, thus ensuring that the data in the RdfRepostiry remains up to date with the existing desktop data.

In the NEPOMUK there are two data wrapper components under development: Beagle++ and gnowsis/Aperture.

Detailed information:

- `http://dev.nepomuk.semanticdesktop.org/wiki/DataWrapper`
- `http://dev.nepomuk.semanticdesktop.org/wiki/ApertureDataWrapper`
- `http://dev.nepomuk.semanticdesktop.org/wiki/BeagleDataWrapper`

Text Analytics

The text analytics component is performing a process of information extraction whose goal is to automatically extract structured or semistructured information from documents on the desktop (e.g. emails, research papers, meeting notes). The text analytics component under development within NEPOMUK supports at the moment the following features: i) named entity recognition, ii) controlled language processing, iii) keyword extraction and iv) speech act detection.

Detailed information:

- `http://dev.nepomuk.semanticdesktop.org/wiki/TextAnalytics`

PIMO Service

The PimoService implements commonly used methods to access personal data structures in the RdfRepository. The PimoService allows easy creation of new classes, new instances, and ensures that the user's PIMO conforms to the NEPOMUK ontologies and that it remains consistent with regard to NRL.

Detailed information:

- `http://dev.nepomuk.semanticdesktop.org/wiki/PimoService`

Local Data Mapping

The LocalDataAlignment analyzes data crawled from resources (from your e-mails or files) to create and annotate instances in the PIMO ontology. It aligns locally crawled data with data stored in the local PIMO of the user, for example it carries out tasks such as adding Dirk as author of the PDF document on your harddrive or Claudia as the person who sent you e-mail X. Also, it cares that only one Person named Dirk exists in the PIMO at one time and that any occurrence of this topic in your documents is matched to this Dirk. The suggestions are confirmed or rejected in a user interface. As a result, the context of persons, places, projects is enriched.

Detailed information:

- `http://dev.nepomuk.semanticdesktop.org/wiki/LocalDataAlignment`

Context Elicitation

This component consists of a set of interfaces and services to allow the observation of and reasoning about a user's current work context. This includes plugins for detecting the current workflow of the user by observing the operations the user does. When integrated into the semantic desktop, these plugins will allow to elicitate knowledge about the current goals of the user which in turn is useful for tuning information structuring, storage, and retrieval services.

Detailed information:

- http://dev.nepomuk.semanticdesktop.org/wiki/UserWorkContext

6.6 Usage Examples of Core Services by Extensions and Applications

So far we introduced the services and extension of the NEPOMUK Middleware in general. In this section we use two examples to illustrate the use of the services. The Task Management example describes the implementation of an extension, the application to application communication example presents an application using NEPOMUK services.

6.6.1 Example: Task Management

In this example we discuss the services involved in the creation, delegation, and completion of a task. We present how the Task Management Extension uses the Core Services and the TMO domain ontology to provide the domain specific functionality. The Extension is presented to the user via a user interface.

A user, Claudia, creates a new task. The Task Management extension is responsible to create the metadata describing the task in the local repository of the NEPOMUK Middleware. The task specific metadata uses the vocabulary defined by the TMO ontology and contains information such as the name, the current state of the task, the creation date, the scheduled end date, the consumed time so far, the role of the involved co-workers, whether the task is a subtask of another task, and so on. Since the extension frequently has to instantiate and modify instances of the TMO ontology the extension provides helper classes to ease the manipulation of the TMO concepts.

When Claudia created the task, it is assigned to her co-worker Dirk. This step involves several services. The metadata of the task is modified in the local repository to reflect the new role distribution. Since the task is no longer a private issue, the metadata is made available in the distributed index of the peer-to-peer system and the according access rights are set. Dirk's NEPOMUK desktop is informed that there is a new task he is supposed to take care of. This is done via the Messaging System. For the exchange of messages between the Task Management Extensions of different NEPOMUK desktops the task management message type is used. Each extension defines its own message types. To inform the Messaging system that the extension is responsible to handle messages of the type task management, the extension registers the message type and the call-back method that should be called when a message of this type arrives. When the message of the new task assignment arrives at Dirk's desktop, the metadata of the task is stored in Dirk's local repository and a message about the new tasked is displayed to him.

Since Claudia is interested in the progress of the task, she uses the Event Notification service to stay up-to-date. She places a subscription to be notified about every change of the metadata that is related to this task. For example, a more detailed subscription informs her when a task related document state changes from "draft" to "final". The subscriptions are internally represented as SPARQL queries. The subscriptions are checked in regular time intervals or on user or service request. If a subscription matches, the registered call-back method is called.

Subscription can be used for both, the user notification and the coordination of actions within an extension. An example of a service coordination action is the handling of the task end. The extension registers a subscription to the change

of the task state “finished”. The registered call-back method is then responsible to issue the required actions. A message sent via the Messaging service informs Claudia that the task is finished. A subscription which observes the scheduled task end date and task progress issues actions when the task is delayed.

6.6.2 Example: Application to Application Communication

In this example show how applications take advantages of the uses of Semantic Web technologies. A user writes a letter in his word processor (e.g., MS Word) and wants to add the receiver’s address. The addresses are stored in the user’s address book application. If both applications support the processing of RDF metadata, this data transfer is done with the help of the operating system clipboard.

The user selects the receiver’s address in the address book and copies it to the clipboard. The RDF representation of the address is stored in the temporary storage to the operating system clipboard. Dirk switches back to the word processor and pastes the clipboard content. The word processor recognizes that the data in the the clipboard is an address and formats the data to fit the letter document template.

Over the years, the Semantic Web community defined numerous ontologies for various domains. In many cases more than one ontology is found for similar or overlapping domains. There are three different ontologies [13, 12, 22] aiming to model the vCard RFC2426 text format [9] as an ontology. Other ontologies such as FOAF [10] or the Semantic Web Research Community ontology (SWRC) [21] provide concepts to represent contact data.

If the target application (in our example the word processor) is able to process domain metadata (the addresses) and does not support the ontology vocabulary provided by the temporal storage of the clipboard, it uses the Mapping and Alignment service from the NEPOMUK Semantic Middleware to transform the the data into an ontology the target application is able to understand. A detailed discussion of the operation of such a *Semantic Clipboard* can be found in [17].

7 Security

Although the NEPOMUK architecture has an advanced status based on strong foundations, as presented until this point, the security is still in an early stage. We performed major steps towards modeling the security needs on the SSD and mapping these needs to possible solutions. Thus, in the following we present a summary of the results achieved so far, results which will be part of the Roman Candle milestone.

7.1 Security requirements

The first step performed was to collect the functional requirements in terms of user identity and access rights management in NEPOMUK. These basically address the users’ needs regarding:

User identity – Identities might be used in different contexts, like: defining nominative access rights to a resource, delegating a task, or managing

the reputation of a person in a social network. E.g.: defining nominative access rights to a resource – when a user decides to share a document, he might choose to share it with only a list of users; he should be able to pick the exact identity of the user in an address book, or any facility where he manages his contacts.

Groups – Groups should be pre-defined in order to rationalize access rights management, mainly in the context of enterprise projects and/or tasks management.

Access rights – Access rights might be used to share a resource, or delegate a task. E.g.: sharing a resource – access rights are managed in order to precise with whom the user wants to share a resource. According to the requirements, access rights should handle the following operations: read, read meta-data only, write, comment, search. The granularity should be on the level of a semantic tag. This encourages sharing as much as possible without sharing private information. Also, the default access rights should propagate according to the knowledge hierarchy (e.g., account > folder > document > tag).

Roles – Roles are used in order to facilitate recurrent access rights management. They provide access rights template on pre-defined resource. In other terms, this corresponds to access rights patterns.

As it can be observed, various contexts were envisaged, some of them within an organization (e.g. knowledge management organisation in a consulting company, or task management in a company). Nevertheless, NEPOMUK personal knowledge management capabilities lead to consider the personal semantic desktop usage in the open community of the Web itself. A rapid review of the needs in this context leads to the following classification regarding data and metadata: (i) Private: I don't want to share the data; (ii) Shared with named users: I want to share this data with particular persons; (iii) Shared with a group: I need to share the data with a team; (iv) Shared with a community: All the P2P community has access to the information; (v) Shared anonymously: I want to share the data with users, groups, a community, but I want to keep privacy on my identity; (vi) Shared with named users with confidential metadata: I want to share a data with named users, but I do not want the meta-data of it to be accessible by anybody, any index or any semantical value-added profiling functionnality.

7.2 Achieved terminology

As a direct result of the requirements analysis, we derived the following terminology which we use in the security scenarios and the possible solutions.

Community. Community is a set of peers joining the same P2P network sharing and accessing the same index. It connects identities by means of peers and identities need a community key to participate.

User A user is an individual human being or an agent acting on behalf of a human being. In our context, he is connected to several communities by means of a peer. He uses several identities in order to be identified by other users of the communities he belongs too. A user can define lists of access rights local to a peer to customize the access to user identities or groups.

Peer A peer is a machine program accessible through an IP address and a port. It can be used by several users, usually by only one user simultaneously.

User identity A user identity is defined and available centrally, and independently from the communities. A user chooses one or several identities to connect to a community by means of a peer. Choosing at least one user identity is mandatory when joining a community. A user identity can participate in several communities.

Group A group is set of user identities. Groups are used in the context of access rights (read, write, annotate, delete) management on resources. Members of the group need to know the group key in order to access the remote resources.

Role A role is a template of access rights on pre-defined resources.

7.3 Security scenario example

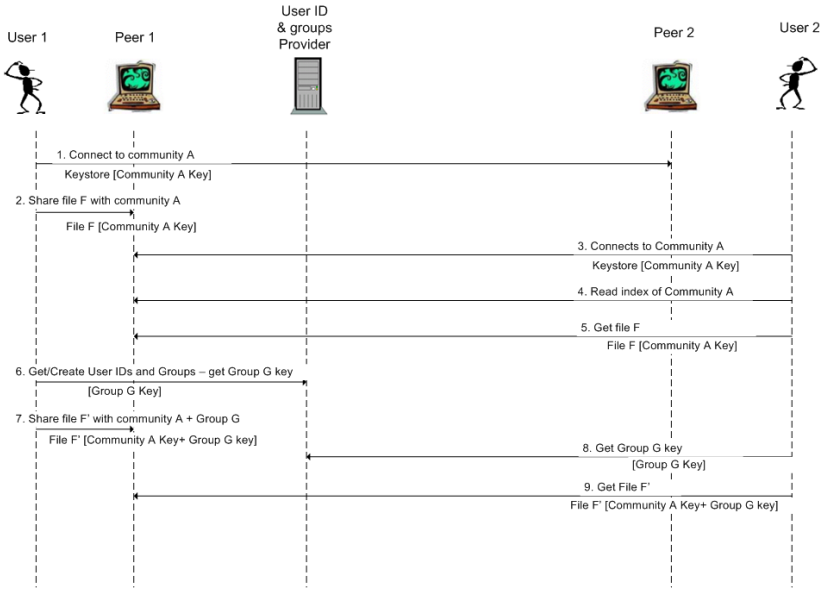


Figure 5: NEPOMUK Security: Example of information flow

Fig. 5 presents an example of how the information flow could look like in the case of sharing a file within a particular group.

- User 1 authenticates with community A using one of his identities. As a result he receives the community key, which is then stored into his local keystore. The community can have a known or empty key. By receiving this key, a user is able to access the community's indexes and share public information.
- User 1 shares a file F with community A. Since he belongs to community A, he tags file F with the Community A key.
- User 2 authenticates with the community A using one of his identities. Again, as a result of the authentication, he can access the community's index and even get file F, shared by User 1.

- User 1 retrieves the user IDs and groups from the central server (note: for authentication and group management reasons we identify the need for a central server[13]). User 2 can create IDs or groups. The central server keeps a key associated to each ID or group. Only the user ID or the group members have access to the key.
- User 1 restricts the access to his file F1 to group G. He tags file F1 with the Community A key + the Group G key.
- User 2 belonging to group G authenticates with group G, and thus receives the group's key. Then he can access file F1 shared by User 1 only inside this group.

7.4 Architectural implications

The security requirements influenced the NEPOMUK Architecture at two levels: (i) component level, and (ii) data workflow level.

The component architecture was directly influenced by the introduction of two additional core services (see Fig. 3), i.e. the Profile Manager and the Identity and Access Control Manager. Also, at the general level we observed the need for a centralized point having the role of managing the overall list of users and groups. The Identity and Access Control Manager represents an enriched local proxy of the this central server. Its main functionality is to store locally (and when necessary synchronize) user identities, group memberships, access keys (for both communities and groups) and manage the access control for the shared resources. The Profile Manager represents an enriched wrapper of the afore-mentioned component. It provides a level of functionality closer to the user, acting in the same time also as a meeting point for some of the social and security aspects of the Social Semantic Desktop. Examples of functionalities would be: editing of the personal profile, management of other profiles (based on roles, e.g. friends), assigning trust policies or defining the shared resources. It can be observed that we're heading towards an interleaved social and security direction, where access rights will be introduced based on the defines social networks.

The second influenced level mentioned above was the data workflow. If until this point the communication, sharing and access of resources (for the distributed environment, i.e. the distributed index) was done in an open and un-protected way, by implementing the security requirements, we introduced the control of the information flow. Based on the direction of the flow, towards the community or from the community, the P2P responsible components will use in a transparent way the security features for enforcing restrictions on the published resources. As an example, considering the the scenario presented in the previous section, when User 2 will want to access file F1 shared by User 1, before releasing the file, the P2P components of User 1 will check what are the access rights of User 2 (on User 1's peer): group membership or personal access rights.

8 Current Architecture Status

In this section we discuss the current state of the architecture by analysing the coverage of the abstracted functionalities as well as the current implemented components.

[13] Central but not unique, distributed as OpenID

Figure 6: Relations between functionalities, services and components.

In Sect. 5 we described a list of 20 functionalities which were abstracted from the scenarios and other materials developed at the case study partners. In Sect. 6 we designed an architecture based on a shared understanding from both the current implementation and the requirements. Figure 7 shows the current coverage of the functionalities by the architecture. The functionalities are specifically oriented towards the NEPOMUK middleware, we do not consider application nor extension functionalities which would be quite numerous. Underlined functionalities are currently covered by services defined in the architecture. Some are a direct one to one relationship, such as the *keyword extraction* functionality with the *text analytics* service. Some are more complex, such as the *resource management* functionality which we consider being covered by the *data wrapping*, *local storage* and *PIMO* services. *Reasoning* seems present in the *context elicitation*, *mapping and alignment* and *local data mapping*; in this case, it might be interesting to assemble the reasoning algorithms of these services into a reasoning service that these three services would rely on.

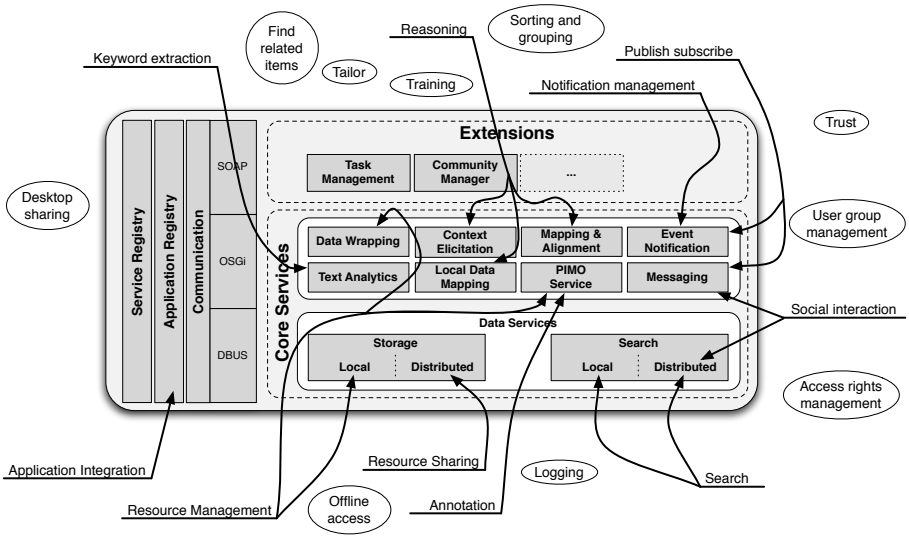


Figure 7: Current functionalities covered by NEPOMUK middleware architecture. Circled functionalities are currently uncovered.

10 out of the 20 functionalities abstracted from the case study material are covered by services of the current architecture. The services are implemented by different components detailed in the previous section. Some services do not yet have an implementation namely :

- Event Notification
- Messaging

We are currently looking at Jabber as a possible messaging implementation

and protocol. Regarding the uncovered functionalities, *User Group Management*, *Access Rights Management* will lead to services and are currently discussed and studied by the NEPOMUK security task force. *Offline Access* is achievable by implementing a routine service[14] which would fetch relevant information when available, relying on the distributed storage and search. *Logging* might be already part of the *context elicitation* service. *Desktop Sharing* and *Trust* are problems larger than the focus of the NEPOMUK project and we do not have a unifying solution. *Find related items*, *Tailor*, *Training* and *Sorting and grouping* are intelligent services which can be built on top of the Middleware Core Services and have to be investigated after the basic infrastructure is stabile.

9 Conclusion

We presented the current state of the NEPOMUK architecture. We gave a common terminology and described our methodology. We detailed the different services, ontologies and technical requirements. We analysed the security implications of the social semantic desktop. We discussed the status of the architecture regarding the current implementation and functionalities coverage.

The main current issue is the integration of the security requirements and the elicitation of the affected services. The development of the extensions and case study applications must be based on the current architecture understanding. The communication of this understanding as well as a critical discussion must take place with all involved project partners.

The case study prototypes have been recently evaluated. The results of this evaluation is valuable for the validation of certain aspects of the architecture, such as the terminology and main structure. The architecture activity will look at this evaluation and analyse the resulting verification of the architecture.

The next step in the definition of the architecture is to document the NEPOMUK application programming interface. The architecture task force will work at describing formally the NEPOMUK Middleware services.

The communication of the architecture will take place in an upcoming architecture meeting involving the main technical partners as well as in the evaluation result workshop. The dissemination and discussion of the architecture will help in the expected convergence towards the social semantic desktop.

[14] a service which runs at regular intervals

References

[1] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Record*, 32(3):29–33, 2003.

[2] Karl Aberer, Philippe Cudré-Mauroux, Manfred Hauswirth, and Tim Van Pelt. Gridvine: Building internet-scale semantic overlay networks. In *3th International Semantic Web Conference ISWC 2004*, pages 107–121. Springer Verlag, 2004.

[3] Aperture a java framework for getting data and metadata, Last visited March 2007. `http://aperture.sourceforge.net/`.

[4] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano. Semantic integration and query of heterogeneous information sources. *Data & Knowledge Engineering*, 36(3):215–249, 2001.

[5] H Beyer and K Holtzblatt. *Contextual Design ? Defining Customer-Centered Systems*. Academic Press, San Diego.

[6] I. Brunkhorst, P. A. Chirita, S. Costache, J. Gaugaz, E. Ioannou, T. Iofciu, E. Minack, W. Nejdl, and R. Paiu. The beagle++ toolbox: Towards an extendable desktop search architecture. Technical report, L3S Research Centre, Hannover, Germany, 2006.

[7] Alan Cooper. *The Inmates are Running the Asylum: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity*. SAMS, Indianapolis, 1999.

[8] Alan Cooper and Robert Reinman. *About Face 2.0: The Essentials of Interaction Design*. John Wiley & Sons, 2003.

[9] F. Dawson and T. Howes. RFC 2426 - vcard mime directory profile. IETF RFC, September 1998. `http://www.ietf.org/rfc/rfc2426.txt`.

[10] The friend of a friend (foaf) project homepage, Last visited February 2005. `http://www.foaf-project.org/`.

[11] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. In *International Journal of Human-Computer Studies*, volume 43, pages 907–928, 1995.

[12] Harry Halpin, Brian Suda, and Norman Walsh. An ontology for vcards. W3C Note, November 14 2006. `http://www.w3.org/2006/vcard/ns`.

[13] Renato Iannella. Representing vCard objects in RDF/XML. W3C Note 22 February 2001, 2001. `http://www.w3.org/TR/vcard-rdf`.

[14] E Mackay, A.V. Ratzer, and P Janecek. Video artifacts for design: bridging the gap between abstraction and detail. In *Designing interactive systems: processes, practices, methods, and techniques, DIS '00*. ACM Pres, 2000.

[15] Knud Möller, Uldis Bojārs, and John G. Breslin. Using Semantics to Enhance the Blogging Experience. In *The third European Semantic Web Conference*, pages 679–696, Budva, Montenegro, June 2006.

[16] Ed. P. Saint-Andre. RFC 3920 - extensible messaging and presence protocol (XMPP): Core. IETF RFC, October 2004. `http://www.ietf.org/rfc/rfc3920.txt`.

[17] Gerald Reif, Gian Marco Laube, Knud Möller, and Harald Gall. SemClip - overcoming the semantic gap between desktop applications. In *5th Semantic Web Challenge at the 6th International Semantic Web Conference*, Busan, South Korea, November 11-15 2007. Springer-Verlag.

[18] L. Sauermann, G. AA. Grimnes, M. Kiesel, C. Fluit, H. Maus, D. Heim, D. Nadeem, B. Horak, and A. Dengel. Semantic desktop 2.0: The gnowsis experience. In *Proc. of the ISWC 2006 Conference*, Nov 2006.

[19] S.K. Semy, M.K. Pulvermacher, and L.J. Obrst. Toward the use of an upper ontology for U.S. government and U.S. military domains: An evaluation. Technical report, MITRE, September 2004. `http://colab.cim3.net/file/work/SICoP/ontac/reference/SemyObrstPulvermacher04.pdf`.

[20] Michael Sintek, Ludger van Elst, Simon Scerri, and Siegfried Handschuh. Distributed knowledge representation on the social semantic desktop: Named graphs, views and roles in NRL. In *Proceedings of the 4th European Semantic Web Conference (ESWC)*, 2007.

[21] Semantic web research community ontology (SWRC), Last visited August 2007. `http://ontoware.org/projects/swrc/`.

[22] Norman Walsh. Extracting vcards from hcard markup, December 12 2005. `http://norman.walsh.name/2005/12/12/vcard`.