Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Förderagentur für Innovation KTI

SciMantic

Deliverable D2.1

Functional Requirements and Analysis of Mechanisms for a Semantic Content Infrastructure

The SciMantic Consortium

University of Zurich
Trialox AG

© 2010 the Members of the SciMantic Consortium

For more information on this document or the SciMantic project, please contact:

Prof. Dr. Burkhard Stiller
University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14
CH-8050 Zürich
Switzerland

Phone: +41 44 635 67 10
Fax: +41 44 635 68 09
E-mail: stiller@ifi.uzh.ch

Document Control

Title:	Functional Requirements and Analysis of Mechanisms for a Semantic Content Infrastructure
Type:	R&D
Editor:	Hasan
E-mail:	hasan@ifi.uzh.ch
Author:	Hasan
Contributors:	Burkhard Stiller, Tsuyoshi Ito, Reto Bachmann Gmür, Manuel Innerhofer
Delivery Date:	15. 03. 2010

Legal Notices

The information in this document is subject to change without notice.

The Members of the SciMantic Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the SciMantic Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Executive Summary

The SciMantic project aims at designing and developing a modular software infrastructure termed Semantic Content Infrastructure (SCI) which enables secure sharing of distributed contents. This SCI consists of independent nodes which interact with each other through Web services to share their contents. To develop a modular SCI, system requirements must be known and available technologies must be studied. This deliverable has the purpose to derive functional requirements of an SCI based on two real-world scenarios which make use of this software infrastructure. This deliverable also describes key technologies needed to build an SCI.

The two scenarios presented in this deliverable are a distributed Web Content Management System (WCMS) and a Knowledge Sharing System (KSS). In a distributed WCMS, instances of WCMS are controlled by independent but co-operating organizations which want to share their Web contents. An example is the various (national) WWF organizations supporting each other in conducting their projects. Each of this organization deploys its own WCMS instance, and raw information on activities of a WWF organization can be accessed by a WCMS instance of another WWF organization to be presented in a specific form. While the set of instances of distributed WCMS can be considered static, the situation is different in KSS where its instances can join and leave at any time. Furthermore, knowledge shared by KSS instances can be stored in a distributed manner into the system database in order to achieve higher availability.

Driven by these two scenarios, a set of key functional requirements is derived. Additionally, performance and security requirements are discussed. The key functional requirements include user and content management, keyword-based content tagging and search, and Peer-to-Peer (P2P) mechanisms. Finally, respective mechanisms and technologies are analyzed which can be used to develop the SCI. They comprise technologies for building modular software architecture as well as technologies for data modeling, for implementing functionalities required, and to provide security.

© Copyright 2010, the Members of the SciMantic Consortium

Table of Contents

1 Introduction

The Semantic Content Infrastructure (SCI) defined by the SciMantic project will leverage the Semantic Web technology by providing applications, which are built on top of the SCI, the capability to efficiently relate and search for semantic contents distributed over the Internet and shared among independent instances of these applications. The key technologies behind this capability are an integration of Peer-to-Peer (P2P) mechanisms with Semantic Web.

One of the Semantic Web technologies is Resource Description Framework (RDF) developed by the World Wide Web Consortium (W3C). RDF is a framework for representing information in the Web using a graph data model. This framework allows for unique identification of Web resources, and through the use of a common ontology it is possible to relate distributed resources. Furthermore, W3C also specifies a Web Ontology Language (OWL) to author ontologies. With OWL semantics, inferences about individuals can be made.

P2P mechanisms are able to deliver services which are scalable, highly reliable, and self-organizing. For example, Distributed Hash Table (DHT) is one of the main mechanisms in P2P networking which allows for the development of a scalable and efficient lookup service where resources are distributed among participating nodes and the set of nodes is dynamic. The advantages of P2P mechanisms combined with the flexibility of graph data model and standardized Web services constitute the strength of SCI-based Internet applications.

The remaining sections in this deliverable are organized as follows: In Section 2, two real-world scenarios are described to show how applications can benefit from Semantic Web technologies. Based on these scenarios, Section 3 lists a set of requirements that must be met by those systems defined in the scenarios which are supposed to be SCI-based applications. The discussion on these requirements is followed in Section 4 by a study of various mechanisms which can be applied to develop an SCI. Finally, Section 5 summarizes this deliverable.

2 Application Scenarios

This section describes two scenarios based on which key requirements are derived. These two scenarios represent important application areas of semantic web technologies.

2.1 Distributed Web Content Management System (WCMS)

This scenario defines a group of independent WWF organizations which co-operate and share their Web contents. The WWF consists of national WWF organizations and their umbrella organization, the WWF International. Each national WWF organization manages their own Web contents, and their web pages may be linked one to another. However, in this scenario, through the use of semantic web technologies, content interconnection goes beyond linked documents and organization boundaries. Each WWF organization deploys its own Web Content Management System (WCMS) which communicates with each other to share their Web contents as depicted in Figure 1. Together they build a distributed WCMS, where contents are not located within a single site, and a Web page can be dynamically composed out of contents from different sources.

User U visits the Web site of the WWF International and wants to get information about projects running by national WWF organizations. The WCMS of the WWF International creates dynamically a Web page as a response to the Web request of the user U. In order to do this, it contacts the WCMS of all national WWF organizations to request project data and through the knowledge of the underlying common ontologies it is able to compile the required Web page. Note that, this is not the same as content syndication, where contents are prepared by the owner in a certain format to be subscribed by others.

Furthermore, user U is interested in activities organized by WWF Switzerland against the global warming. Thus, she also visits the WWF Switzerland's Web page describing those activities. In order to provide user U with other information she might be interested in, the WCMS of WWF Switzerland looks for related activities organized by other national WWF organizations. Since WWF Canada is doing similar activities, its WCMS provides the required data to the WCMS of WWF Switzerland, which then presents to user U a short description of those related activities at a special place on the requested Web page.

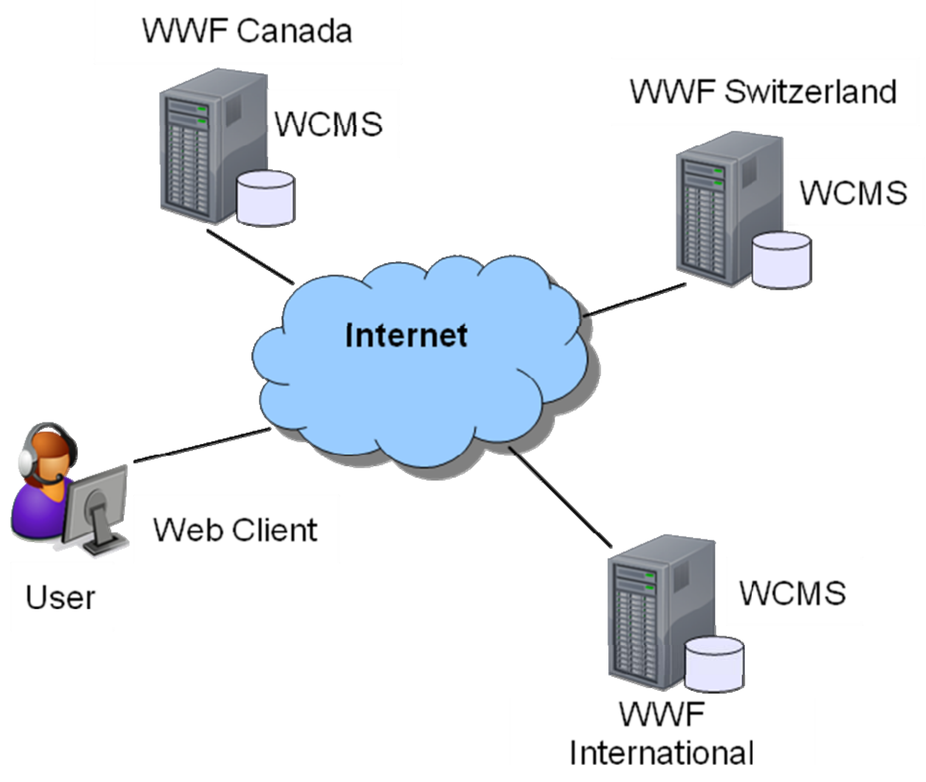

Figure 1: Distributed WCMS

2.2 *Knowledge Sharing System (KSS)*

The second scenario deals with sharing of knowledge in a scientific community. Basically, knowledge sharing is not restricted to a particular community, thus mechanisms described in this scenario should be applicable to knowledge sharing in other communities as well.

Professor X and the PhD Student S are working in related research area, namely Semantic Web. Professor X is carrying out a research project on "High Performance Triple Store" and gives a lecture on Semantic Web to graduate students. The PhD student S is writing his thesis on "Efficient RDF Query Evaluation" and is looking for material for his Related Work chapter. Both Professor X and the PhD Student S are participating in a "Knowledge Sharing Network for Information Technology", since they are interested in knowledge acquired by other scientists working in the same or related area of research.

Each participant of this network is expected to share his/her knowledge by entering it into the system's database using a software tool called Knowledge Sharing System (KSS) (cf. Figure 2). The whole knowledge in the database is divided into (knowledge) units. The participant who enters a particular unit into the database is the unit owner and has the right to manipulate it. Once a unit is stored into the database, all other participants have read access to it, until it is removed from the database. To allow participants to search knowledge of a specific topic, units are tagged with keywords by their owner. Therefore, Professor X enters a unit about her research into the database and tags it with the following keywords: RDF, TripleStore, and SemanticWeb. With the help of the KSS, the PhD Student S can search for work performed in the area of Semantic Web and find the unit provided by Professor X.

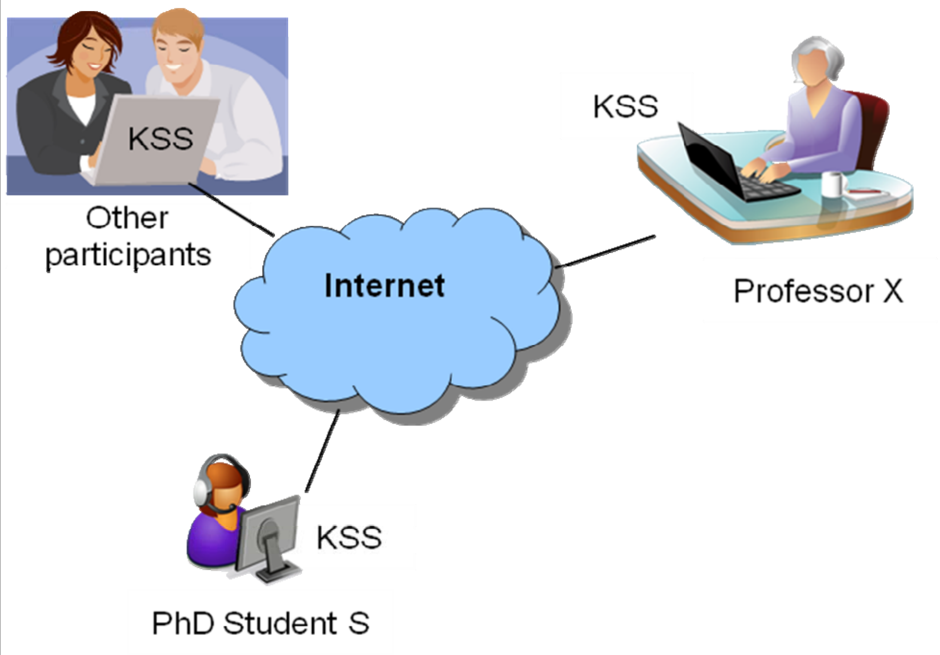

Figure 2: Knowledge Sharing System

Since Professor X wants to update her lecture notes on Semantic Web according to recent research results, she uses the KSS to request for notifications if units tagged with the SemanticWeb keyword are modified or if a new unit is created and tagged with this keyword. This way, she is able to keep her lecture notes up-to-date.

3 Requirements

The two scenarios presented in the previous section are analyzed to derive key functional requirements which the distributed WCMS and KSS have to fulfill. In addition to functional requirements, this section also discusses performance and security requirements since information searching and processing normally take some time, and rights of a user are generally restricted.

3.1 Functional Requirements

Table 1 lists functional requirements for both systems: distributed WCMS and KSS. A cross (X) in a table cell of column "WCMS" or "KSS" means that the specified requirement is valid for the respective system. If the cross is placed within a square bracket, then the requirement for the system is optional.

Table 1: Functional requirements for WCMS and KSS

No.	Requirement	WCMS	KSS
1	Data management (creation, storage, retrieval, modification, and removal) Remark: Data are not Web pages or knowledge units as presented to	X	X

	users, but are the underlying raw data which a Web page or a knowledge unit is composed of. In a distributed WCMS, data owned by an organization is stored within this organization. In a KSS, data owned by a user are stored locally and additionally can be stored remotely in other KSS instances to allow for a higher availability.		
2	Page management (creation, storage, retrieval, modification, and removal) Remark: A Web page is created through composition of data from various sources depending on what information the page should present.	X	-
3	Knowledge unit management (creation, storage, retrieval, modification, and removal) Remark: A knowledge unit is normally composed of data from a single source, since it is created by one participant. However, data of a unit can also be stored on remote instances in a distributed manner, in which case various sources must be consulted to construct the unit.	-	X
4	User management (creation, storage, retrieval, modification, and removal) Remark: In a distributed WCMS, users who consume information (visitors of a Web site) are only allowed to view data, whereas users who produce information (editors of Web pages) are allowed to perform all data management functions. In a KSS, unit owners are allowed to modify their units, whereas other users may only have read access. Taken this into account, the respective system must be able to distinguish between different roles of users.	X	X
5	Keyword management (creation, storage, retrieval, modification, and removal)	X	X
6	Tagging Remark: This function has the purpose to enable searching of data (Web resources or knowledge units) based on keywords.	X	X
7	Keyword-based search Remark: This function enhances standard data retrieval function.	X	X
8	Web client interface Remark: This interface allows Web clients to visit Web pages provided by the distributed WCMS.	X	-
9	Service discovery Remark: An instance of a KSS needs to connect to other instances in order to share knowledge units. This requires a mechanism to discover other instances currently online.	-	X
10	P2P communications and mechanisms Remark: The set of interacting KSS instances is dynamic, thus the P2P communication paradigm is very well suitable for this system. In a distributed WCMS where the set of instances is known, P2P mechanisms are not needed.	-	X
11	Event subscription	[X]	X

© Copyright 2010, the Members of the SciMantic Consortium

	Remark: The KSS must support subscriptions to get notifications if a knowledge unit is modified. In case of WCMS, notification is needed only if a WCMS instance caches data from other WCMS instances.		
12	Notification of data modifications Remark: A KSS instance must send notifications to subscribers if a unit is modified through this KSS instance.	[X]	X
13	Subscription management (creation, storage, retrieval, and removal)	[X]	X

3.2 Performance Requirements

Distributed WCMS as well as KSS is an interactive system, thus the key performance parameter is the response time. Several factors determine the response time of an interactive system with distributed data that communicates over the Internet:

- Access bandwidth: the bandwidth available for accessing the Internet
- Network delay: delay within the network
- Search delay: the time needed to find the data requested
- Processing delay: the time needed to construct a Web page or a knowledge unit.

To allow working conveniently, the response time must be within a tolerable range. According to [4] a Web page should load in under 8.6 seconds (non-incremental display) or in under 20 to 30 seconds (incremental display) with useful content within 2 seconds.

3.3 Security Requirements

Although data in distributed WCMS and KSS are accessible to public, there are some security concerns that must be solved. The following table lists security requirements for both systems.

No.	Requirement	WCMS	KSS
1	Permission management Remark: Since not all users are allowed to do modifications to data, specific permissions must be defined and granted to certain users.	X	X
2	Authentication Remark: In order to know who is currently interacting with the system, users need to be authenticated. However, this should not preclude unauthenticated (anonymous) users to have access to the system.	X	X
3	Authorization Remark: Authorization is necessary to only allow certain users to do certain actions. In a distributed WCMS, only editors are allowed to do changes to Web pages, whereas in a KSS, only unit owners are allowed to make changes to their units.	X	X

4 Mechanisms

In this section, main mechanisms potentially needed to meet requirements discussed in Section 3 are analyzed.

4.1 Modular Architecture

To create a software infrastructure with extensible functionality, a modular architecture is required. Following subsections describe two service oriented designs.

4.1.1 OSGi

OSGi stands formerly for Open Services Gateway initiative, but today this acronym is not used, since it does not reflect current work of the OSGi Alliance, which is the open standard organization founded in March 1999, that specifies and maintains OSGi standard. The initial goal was to provide an open standard that enables multiple value-added services to be dynamically (also remotely) loaded and run on a single services gateway such as a set top box, cable modem, DSL modem, PC or dedicated residential gateway. This gives application service providers, network operators, and device and appliance manufacturers both application and platform independence. Today, an OSGi framework [13] is a service platform that enables dynamic installation and removal of components which may interact with each other within an execution environment through services implemented by those components.

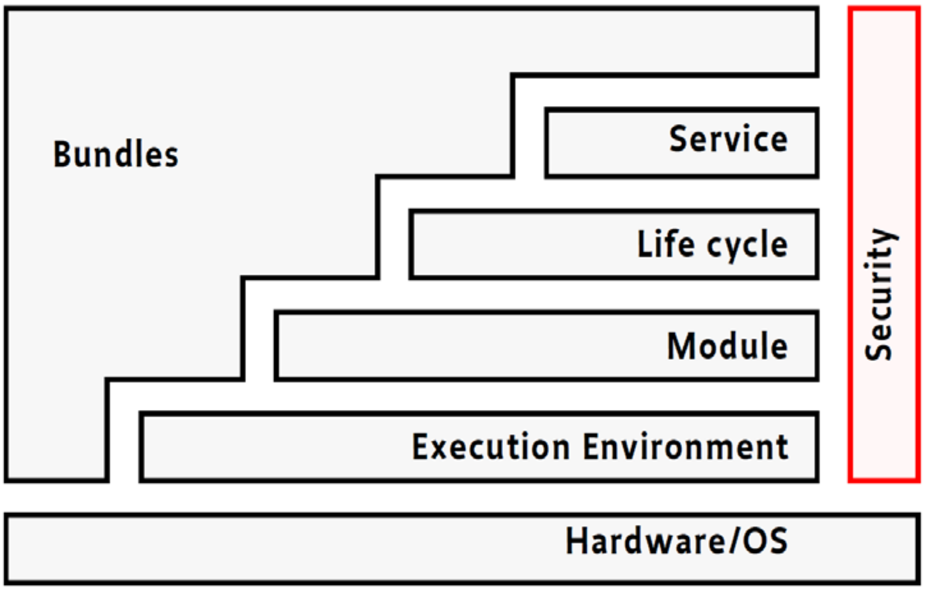

Figure 3: OSGi framework [4]

In an OSGi world, an application is composed out of a set of interacting components which come in the form of bundles for deployment. Bundles can be remotely installed, started, stopped, updated and uninstalled without requiring a system reboot. Therefore, extending the functionality of an application is easily done by just adding new components into a running system. The OSGi technology is developed to create a collaborative software environment. Initially, it does not provide for component interactions across a Virtual Machine (VM), however, in the new version 4.2 of the specification, new services and capabilities are added. This includes “Remote Services” [14] that allows the exporting of services to remote VMs (formerly known as Distributed OSGi).

The OSGi has a layered model as depicted in Figure 3. Each bundle is a normal JAR component containing a collection of classes, JARs, and extra manifest headers (configuration files) that contain information on the bundle’s external dependencies if any. The bundle concept embodies modularity. In OSGi, everything in a JAR is hidden unless explicitly exported, and a bundle that wants to use another JAR must explicitly import the parts it needs.

The Services layer connects bundles in a dynamic way by offering a publish-find-bind model for Plain Old Java Objects (POJO). A bundle can provide a service and register it with the OSGi service registry under one or more interfaces. Other bundles can request from the registry a list of all services registered under a specific interface or class, and invoke those services. The service registry also allows bundles to detect the registration of new services, or the removal of services, through listening to the respective events, and adapt accordingly. To allow uninstalling bundles at any time, services must be dynamic. Therefore, bundles using services of other bundles must consider this dynamics.

The Life-Cycle layer provides an API for managing the life-cycle (install, start, stop, update, and uninstall) of bundles, whereas the Module layer defines encapsulation and dependencies, *i.e.* mechanisms for a bundle to import and export code. The optional Security layer is based on the Java 2 security architecture and provides the infrastructure a controlled environment to deploy and manage bundles. The release 4.2 of the specification supports definitions of negative permissions to forbid specific actions instead of allowing them. Finally, the Execution Environment defines classes which are available in a specific platform. At the time of writing this report, there are 4 open source implementations of the OSGi framework known to the authors: Apache Felix [2], Eclipse Equinox [5], FUSE ESB 4 [6], and Knopflerfish [11].

4.1.2 Service Component Architecture (SCA)

Service Component Architecture (SCA) is a set of specifications which describe a model for building applications using a Service-Oriented Architecture (SOA). SCA specifications are maintained by the Organization for the Advancement of Structured Information Standards (OASIS), a global consortium that drives the development, convergence and adoption of e-business and Web service standards.

The smallest building blocks of an application in SCA are components which may be built using different technologies but based on SCA-defined programming models. SCA defines a common assembly mechanism to combine components into an application. Components can be distributed over several processes in a single machine or in multiple machines. Interactions among components are modeled as services, thus, cleanly separating implementation technology from the function provided. A detailed definition of component in [9] reads: “One basic artifact of SCA is the component, which is the unit of construction for SCA. A component consists of a configured instance of an implementation, where an implementation is the piece of program code providing business functions. The business function is offered for use by other components as services. Implementations may depend on services provided by other components – these dependencies are called references. Implementations can have settable properties, which are data values which influence the operation of the business function. The component configures the implementation by providing values for the properties and by wiring the references to services provided by other components.”

Services, references, and properties are the configurable aspects of a component, and are collectively referred to as the component type. One problem in SCA is the declaration of a component type which is implementation dependent. Java implementations of a component allow the use of code annotations for this declaration, while other implementations have to use other mechanisms. Thus, the determination of component types becomes awkward as can be read in [9]: “The component type is calculated in two steps where the second step adds to the information found in the first step. Step one is introspecting the implementation (if possible), including the inspection of implementation annotations (if available). Step two covers the cases where introspection of the implementation is not possible or where it does not provide complete information and it involves looking for an SCA component type file. Component type information found in the component type file must be compatible with the equivalent information found from inspection of the implementation. The component type file can specify partial information, with the remainder being derived from the implementation. In the ideal case, the component type information is determined by inspecting the implementation, for example as code annotations. The component type file provides a mechanism for the provision of component type information for implementation types where the information cannot be determined by inspecting the implementation."

Components can be logically combined to build a larger and more complex construct termed composite (cf. Figure 4) which can further act as a component in another composite. This means composites can be nested. A complete application might be constructed from just one composite or a combination of several different composites. For each composite, there is a configuration file that describes how components are connected to build the composite. This file uses an XML-based format called the Service Component Definition Language (SCDL, pronounced "skiddle"). As shown in Figure 4, a composite can promote services and references of its components. It can contain property declarations for setting its component properties.

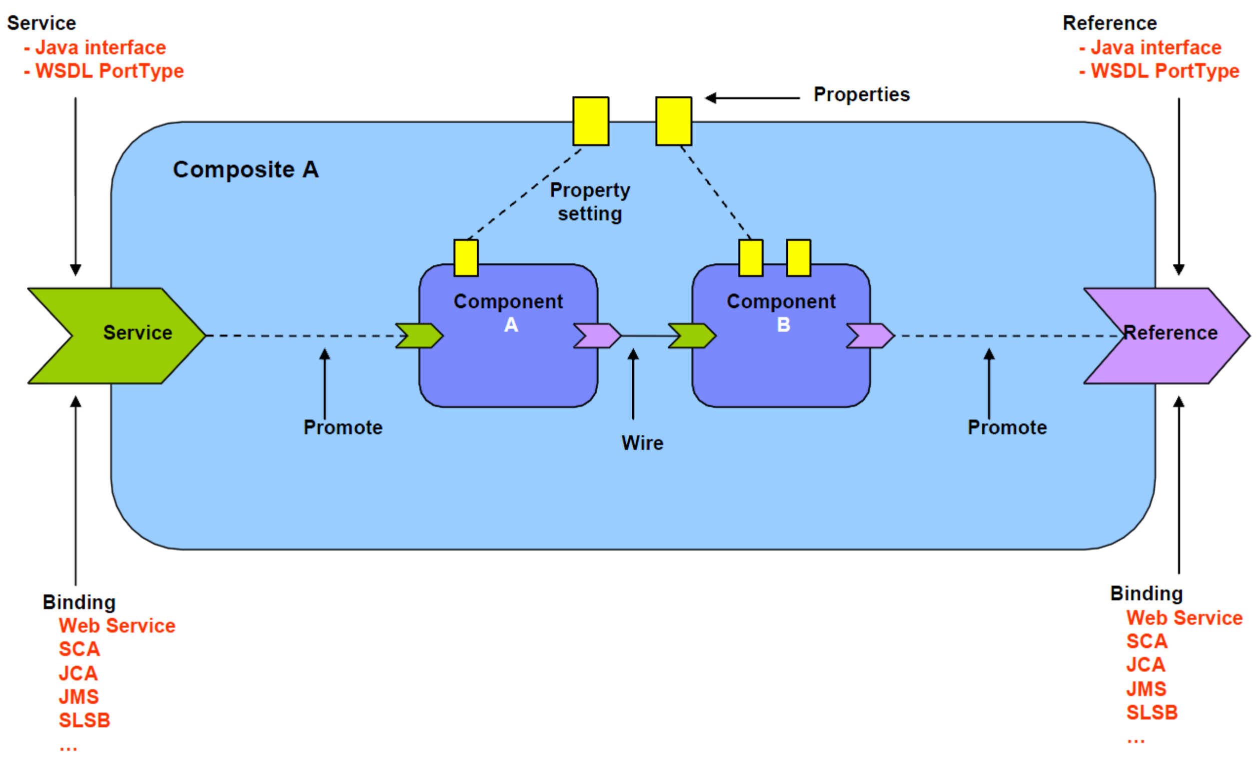

Figure 4: SCA Composite [3]

In SCA, composites are deployed within an SCA Domain which represents a complete SCA runtime configuration, potentially distributed over a series of interconnected runtime nodes. Typically, the functionality provided by an SCA Domain is controlled by a single organization. An SCA Domain hides all SCA mechanisms from other entities interacting with services provided by this SCA Domain. This means connections to services outside an SCA Domain cannot use SCA wires, but must use binding specific mechanisms for addressing services. Furthermore, implementations of SCA wires are SCA runtime (vendor) specific.

Currently, three SCA implementations are known to the authors: Apache Tuscany [2] developed by the Apache Software Foundation, Newton Framework [22] by Paremus Limited, and Fabric3 [9] by Metaform Systems.

4.2 Contents Modeling

Contents can be modeled using relational model, either traditionally as a set of tuples contained in tables or alternatively as a set of triples contained in a graph. This section briefly describes the graph data model as specified in Resource Description Framework (RDF) [23], one of the recommendations of the World Wide Web Consortium (W3C).

RDF is a framework for representing information in the Web. Information is modeled as a set of triples comprising a subject, a predicate, and an object. This set of triples build a directed graph, in which subjects and objects are nodes, and predicates are directed arcs from subject to object. An RDF triple is a triple with the following special characteristics:

- A subject is an RDF Uniform Resource Identifier (URI) reference or a blank node
- A predicate is an RDF URI reference
- An object is an RDF URI reference, a literal, or a blank node.

Thus, an RDF graph is a set of RDF triples. A blank node is described in [23] as “a node that is not a URI reference or a literal. In the RDF abstract syntax, a blank node is just a unique node that can be used in one or more RDF statements, but has no intrinsic name.” The key advantage of this graph data model lies in its flexibility to deal with structural changes of relations. New predicates can be easily added and existing ones easily removed, since there is no construct that fixes the structure as in tables.

There are various triple store implementations available today, *e.g.*, Jena [5], Sesame [1], Mulgara [11], Bigdata [23], and Virtuoso [14]. To allow applications to manipulate triples, these implementations offer different APIs or support popular ones, such as Jena API, or Storage and Inference Layer (SAIL) API developed by OpenRDF. Triple store implementations generally provide SPARQL [26] query support to allow retrieving data in a standard way and a Web Ontology Language [24] (OWL)-based inference mechanism.

4.3 Authentication

Authentication is necessary in a system which needs to know, whom the system is interacting with. This will allow authorization (cf. Section 4.4) to be based on user authentication.

4.3.1 Java Authentication and Authorization Service (JAAS)

Java provides Java Authentication and Authorization Service (JAAS) [11], a security framework, that can be used to determine who is currently executing the Java code. In JAAS, a source of an access request is called a Subject, which may be any entity, such as a person or a service. A Subject can have many Principals which are used to identify the Subject, for example a name or an identification number. A Subject also has credentials to be used for authentication.

Authentication based on JAAS is generic, which means that it can be used with different underlying authentication technologies. A technology specific authentication module implements the LoginModule interface. A LoginModule implementation performs the authentication according to a specific technology, *e.g.*, a finger print authentication or a simple password-based authentication. A JAAS login configuration file is used to associate a name in each entry with a LoginModule implementation to be applied. This name will be referred to by an application if it wants to use the corresponding LoginModule implementation.

To authenticate a user, the JAAS framework requires a LoginContext. It is given information about which entry in the login configuration file is applicable and a CallbackHandler to be used for interacting with a user to obtain the authentication information. After a successful authentication the Subject is populated with the respective Principals.

Thus, to use JAAS for authentication, an application must provide an implementation of the LoginModule and the CallbackHandler.

4.3.2 HTTP Authentication

To authenticate users accessing a Web site, HTTP-based authentications are suitable. Two authentication mechanisms based on HTTP have been studied:

- HTTP Basic Authentication
- Cookie-based Authentication

HTTP Basic Authentication provides a simple mechanism for the transfer of authentication information from a client to a web server. This information consists of a user name and a password. If a web request to access a protected resource does not contain the required authentication information, the web server sends a 401 response (Unauthorized) with a WWW-Authenticate header specifying the authentication scheme(s) and parameters. Upon receiving this response, a web browser will pop up a dialog box to request the user name and password if it does not already have this information. In subsequent requests the authentication information in the form of username:password will be Base64 encoded and sent in an Authorization-Header.

The advantage of this mechanism lies in its simple implementation and is therefore the most frequently used HTTP-based authentication mechanism. However, it has two disadvantages:

- Passwords are transmitted in plain text.
- The dialog box provided by existing web browsers for requesting user credentials cannot be customized and integrated into a user defined login page.

With a Cookie-based Authentication, an application can use its own user interface to request user credentials. After a successful authentication, the client is sent a cookie containing credentials to be stored and used for further interactions, in which the client must send back the cookie. Cookies are sent in a specific HTTP header, basically without using encryption. Therefore, it is as insecure as the HTTP Basic Authentication. In order to have a secure authentication, HTTP can be used in combination with Secure Sockets Layer (SSL) or Transport Layer Security (TLS) which is known as Hypertext Transfer Protocol Secure (HTTPS).

4.4 Authorization

Authorization is required to control access to protected resources. Only users who have the required access control rights may gain access to those resources.

4.4.1 JAAS

The JAAS framework also provides functionality for authorization, i.e. to check whether a code source executed by a Principal (thus a Subject) has the required permissions (access control rights) to perform an action. Java provides an abstract class java.security.Permission to represent access to system resources and a set of its subclasses for each specific permission type, *e.g.*, permissions for manipulating files. New permission classes are derived from this abstract class or one of its subclasses. Typically, a permission is specified by defining actions that are allowed to be performed on certain targets in a particular permission type. For example, one can define a permission to read a file named test located in the directory /tmp. The action here is “read”, whereas the target is “/tmp/test”. However, there are also other types of permissions where actions need not be specified. In this case, an entity either has the permission or not.

In order to grant permissions to a code source or Principals a system wide Policy object is used. If needed, an application can change currently installed Policy object with another instance. Thus,

© Copyright 2010, the Members of the SciMantic Consortium

location of policy information and its structure depends on the Policy implementation. JAAS provides for a Policy reference implementation which consults policy information from one or more policy configuration files. Policy information consists of a set of policy entries, each of which specifies a set of permissions to be granted to a specific code source and/or to a set of Principals. A specific code source is identified by its location and/or its signer, whereas a Principal is identified by a pair of class name and Principal name. For example, one can grant a permission to read and write into the directory "/tmp/games" to codes downloaded from "www.games.com", signed by "Duke" and executed by the Principal of class javax.security.auth.x500.X500Principal whose name is "cn=Alice".

Authorization (permission check) is performed by invoking a method called checkPermission provided by the AccessController class on an instance of the respective permission class. If an application does not want to use the built-in access control algorithm of the AccessController class, it can install its own SecurityManager which provides for a checkPermission method implementing own access control algorithm. Thus, access control can also be enforced by a SecurityManager which is typically activated when the system is brought up.

JAAS provides the methods doAs and doAsPrivileged in order to allow an action to be performed as a particular Subject. These two methods differ in the AccessControlContext associated with the Subject. The method doAs uses the current thread's AccessControlContext, whereas the method doAsPrivileged uses the provided AccessControlContext specified in its parameter. An AccessControlContext contains amongst others information about the location of all codes executed so far and the permissions granted to the codes. If a Subject does not have the required permission, the AccessController respectively the SecurityManager raises the corresponding AccessControlException.

4.4.2 OAuth

In mid 2009 a new Working Group at IETF called OAuth Working Group was established to standardize a protocol which enables a client on behalf of a resource owner to access her protected resources stored on a server without requiring the resource owner to reveal her credentials to the client. The client can be an application or a third-party Web site. In a traditional client-server authentication model, the client knows the credentials of the resource owner, which is normally the client itself. OAuth introduces a third role: the resource owner which needs not to be the role that wants to access the protected resources.

An example given in the Internet Draft entitled "The OAuth 1.0 Protocol" [2] motivates the need of this protocol: "For example, a web user (resource owner) can grant a printing service (client) access to her private photos stored at a photo sharing service (server), without sharing her username and password with the printing service. Instead, she authenticates directly with the photo sharing service which issues the printing service delegation-specific credentials." The Internet Draft specifies the usage of tokens as delegation credentials and HTTP/1.1 for the communication protocol. Such tokens usually have a restricted scope and limited lifetime. More precisely, OAuth defines a mechanism to obtain token credentials by using HTTP redirections.

This redirection-based authorization method includes three steps, as summarized in [2]:

1. The client obtains a set of temporary credentials from the server (in the form of an identifier and shared-secret). The temporary credentials are used to identify the access request throughout the authorization process.
2. The resource owner authorizes the server to grant the client's access request (identified by the temporary credentials).
3. The client uses the temporary credentials to request a set of token credentials from the server, which will enable it to access the resource owner's protected resources.

Note that after step 1, the client redirects the resource owner (her user-agent) to the server in order to carry out step 2. Following step 2, the server redirects the resource owner to the client again.

In a distributed WCMS or a KSS, read access to resources is always allowed. Therefore, there is no need to add OAuth capability, either as a client or a server, to an instance of these systems. However, if the underlying Semantic Content Infrastructure is used to manage protected resources of users, then the support of OAuth is beneficial.

4.5 Tagging

Tagging is a simple mechanism to associate a particular content with a set of concepts (also known as keywords). From a user point of view, tagging provides a functionality to organize data and can ease searching of data by specifying the associated concept. In this respect, W3C has worked out a knowledge organization system called Simple Knowledge Organization System (SKOS) [33]. It is an RDF vocabulary for representing semi-formal knowledge organization systems, such as thesauri, taxonomies, classification schemes and subject heading lists.

A concise description of concepts in SKOS is given in [33] as follows: “concepts can be identified using URIs, labeled with lexical strings in one or more natural languages, assigned notations (lexical codes), documented with various types of note, linked to other concepts and organized into informal hierarchies and association networks, aggregated into concept schemes, grouped into labeled and/or ordered collections, and mapped to concepts in other schemes.”

The meaning of a concept can be considered to be represented by its labels. One or more labels can be defined for a concept, however, there is only one preferred label allowed per language within a Knowledge Organization System (KOS). SKOS defines two other types of labels: alternative labels and hidden labels. Hidden labels are for example useful to represent misspelled variants of other labels.

An important property of concepts is their semantic relationships. Semantic relations enhance the meaning of a concept beyond its labels. SKOS defines three semantics relations: “broader”, “narrower”, and “related”. While “broader” and “narrower” relations enable the representation of hierarchical links, “related” relations enable the representation of associative links. The SKOS model does not determine that these relations are transitive or intransitive. In many KOS, hierarchical relations can be considered transitive, thus in these cases, one can apply inference mechanisms to assert about relationships.

To allow classification of concepts, SKOS introduces concept schemes. Concepts are therefore defined within a particular scheme. In this respect, the Semantic Web can be considered to contain a network of concept schemes. An application that deals with several concept schemes may want to map or relate concepts in different schemes. Therefore, SKOS provides several properties to enable this mapping: exact match, close match, broad match, narrow match, and related match.

The SKOS model is used by several organizations extensively including DBpedia.org and the United States’ Library of Congress for its thesaurus of subject headings.

4.6 Decentralized Indexing

Distribution of data over a dynamic set of communicating nodes in a system leads to the need of an indexing service. Indexing service can be centralized or decentralized. The advantages of a decentralized indexing lie in the better scalability and the fact that it has no single point of failure. Distributed indexing in form of Distributed Hash Tables (DHT) is widely researched in relation to P2P networking. A DHT provides a lookup service for finding values given a key. The mapping from keys to values is distributed among the participating nodes in a way that if nodes join or leave, the service disruption is minimal.

© Copyright 2010, the Members of the SciMantic Consortium

There are various solutions available today for DHT, amongst others Kademlia [5], Pastry [16], and Chord [17]. They differ in their performance with respect to various metrics: number of routing hops during lookup, number of messages transmitted when nodes join or leave, and amount of states to be stored per node. The problem with DHT is its exact mapping of keys in a search. A system which wants to support vague search terms, requires advanced indexing mechanisms. Research has been performed in the area of similarity search [1] as well as search based on keywords, *e.g.*, SHARK [7] [8]. SHARK autonomously arranges nodes into a multidimensional symmetric redundant hierarchy, where each node is assigned a Group of Interest which represents a leaf in the hierarchy. Objects stored in each node are described through meta-data that determine the hierarchical structure for the construction and operation of SHARK. Search for objects is based on routing of queries, which are meta-data descriptions of those objects.

4.7 Event Subscription and Notification

In a system where nodes are interested in changes of states of other nodes, a mechanism is required to propagate these state changes. Basically, a node can ask (poll) other nodes periodically whether there are changes. However, this is inefficient if changes rarely occur. How frequent rare really means is application specific. Instead of polling other nodes, a node which has state changes can push this information to all other nodes. Obviously, this is very inefficient if changes occur frequently. An efficient way to propagate state changes is through event subscription and notification. A node can subscribe to a particular change pattern and let it be notified of those changes that match the pattern.

Today, well-established Relational Database Management Systems (RDBMS) like Oracle [13], Microsoft (MS) SQL Server [8], IBM DB2 [4], mySQL [15], and PostgreSQL [18] provide a mechanism called trigger that allows one to define a routine that will be called before or after changes are made to a database (structural or content). The functions that are supported by triggers are system (implementation) specific, but normally also allow for invocations of external programs. Typical usage includes logging of changes, prevention of changes, enforcement or execution of business rules. Based on this mechanism, event subscription and notification can be implemented. Thus, MS SQL introduces a mechanism called query notification which allows applications to request notification within a specified time-out period when the underlying data of a query changes.

To the knowledge of the author, most of the triple store implementations existing at the time of writing this report, *e.g.*, Jena [5], Sesame [1], Mulgara [11], and Virtuoso’s native triple store [14], do not support the trigger mechanism as described above. Therefore, an application framework which uses triple stores as data storage, and which wants to offer event subscription and notification, has to implement this functionality itself.

Although Virtuoso’s native triple store does not support trigger mechanism, Virtuoso’s relational database support does. Virtuoso [14] is a server solution which allows applications to transparently connect to databases from various vendors (a. o. Oracle, Microsoft SQL Server, DB/2, Informix, Progress, CA-Ingres). All databases are treated as a single logical unit. Virtuoso provides both a native database capability and a virtual database (VDB) that integrates third-party ODBC data sources with its own. The VDB allows transparent unified queries across all linked data sources. Virtuoso supports the creation of stored procedures and triggers in relational databases which are modeled after SQL 3.

The Open Anzo project [13] developed a semantic middleware platform which supports data replication for offline use and near real-time synchronization through a notification service. Triples from the server can be selectively cached on client machines and persisted for offline access and to improve performance. A client receives notification of updates by triple pattern subscription through a Java Message Service (JMS) Notification subsystem about changes to relevant triples and named graphs. The notification service consists of an update results publisher and a notification server. The interaction is described in [13] as follows: “As updates are processed on the server, the update results are passed to an updates publisher. The updates publisher takes the contents of the update results and places them on an update queue within the JMS server. The notification server is responsible for processing messages from that queue and placing them on the queues of any client that has permission to see the data. The notification server decomposes the update message into individual statement messages before it sends the data to the client, which allows the clients to filter their subscriptions based on contents of the statements.” Anzo stores RDF triples in underlying relational databases. Supported RDBMS include IBM DB/2, Oracle, PostgreSQL, MS SQL Server, H2 and HSQLDB.

5 Summary

This deliverable has presented two real-world scenarios, namely a distributed WCMS and a Knowledge Sharing System, in which Semantic Web technologies play an important role. Based on these scenarios, a set of functional, performance and security requirements have been derived. In order to meet those requirements specified, supporting mechanisms needed have been studied and analyzed. These mechanisms include OSGi and SCA as modularization technology, RDF graph data model, distributed hash tables, contents tagging, notification of data changes, as well as user authentication and authorization. This deliverable provides a basis for the design and implementation of a Semantic Content Infrastructure defined by this project.

References

[1] Aduna B.V.: *User Guide for Sesame 2.3*; 2010, URL: http://www.openrdf.org/doc/sesame2/users/.

[2] Apache Software Foundation: *Apache Felix*; URL: http://felix.apache.org/site/index.html.

[3] Apache Software Foundation: *Apache Tuscany*; URL: http://tuscany.apache.org/.

[4] T. Bocek, E. Hunt, D. Hausheer, B. Stiller: Fast Similarity Search in Peer-to-Peer Networks; 11th IEEE/IFIP Network Operations and Management Symposium, April 2008.

[5] Eclipse Foundation: *Eclipse Equinox*; URL: http://www.eclipse.org/equinox/.

[6] FUSE Open Source Community: *FUSE ESB 4*; URL: http://fusesource.com/products/enterprise-servicemix4/#documentation.

[7] E. Hammer-Lahav (Editor): *The OAuth 1.0 Protocol*; Internet-Draft, draft-hammer-oauth-10, February 2010, URL: http://tools.ietf.org/html/draft-hammer-oauth-10.

[8] IBM Corporation: *IBM DB2 Universal Database - Application Development Guide: Programming Server Applications - Version 8*; Reference Manual, 2002.

[9] Jena Development Team: *Jena – A Semantic Web Framework for Java*; 2010, URL: http://jena.sourceforge.net/documentation.html

[10] A. King: *Speed Up Your Site: Web Site Optimization*; New Riders Press, 2003.

[11] Makewave: *The Knopflerfish Project*; URL: http://www.knopflerfish.org/documentation.html.

[12] P. Maymounkov, D. Mazières: *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*; 1st International Workshop on Peer-to-peer Systems, 2002.

[13] Metaform Systems: *Fabric3 Reference Guide Version 1.3*; 2009.

[14] Microsoft Corporation: *SQL Server 2008 R2: SQL Server Database Engine*; 2008.

[15] Jan Mischke: *Scalability of Lookup and Search Overlays in Peer-to-Peer Networks*; ETH Zurich, TIK-Schriftenreihe Nr. 56, Diss. ETH Zurich Nr. 15363, 2004.

[16] Jan Mischke, Burkhard Stiller: *Peer-to-peer Search with SHARK: Symmetric Redundant*

© Copyright 2010, the Members of the SciMantic Consortium

Hierarchy Adaption for Routing of Keywords; TIK-Report Nr. 164, ETH Zurich, February 2003.

[17] Mulgara Development Team: *User Documentation*; 2010, URL: http://www.mulgara.org/trac/wiki/Docs

[18] OASIS: *SCA Service Component Architecture: Assembly Model Specification*; SCA Version 1.00, March 2007.

[19] OpenAnzo.org: *The Open Anzo Project*; 2009, URL: http://www.openanzo.org/projects/openanzo/wiki/

[20] OpenLink Software: *OpenLink Virtuoso Universal Server*; 2009.

[21] Oracle Corporation: *Oracle® Database Concepts 11g Release 2 (11.2) Chapter 8: Server-Side Programming: PL/SQL and Java*; October 2009. URL: http://download.oracle.com/docs/cd/E11882_01/server.112/e10713/srvrside.htm#g23577

[22] Oracle Corporation: *Java^{TM} Authentication and Authorization Service (JAAS) Reference Guide*; 2006, URL: http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html

[23] Oracle Corporation: *MySQL 5.5 Reference Manual*; 2010.

[24] The OSGi Alliance: *OSGi Service Platform Core Specification*; Release 4, Version 4.2, June 2009.

[25] The OSGi Alliance: *OSGi Service Platform Service Compendium*; Release 4, Version 4.2, August 2009.

[26] Paremus Limited: *Newton 1.5-DEV: Developer Guide*; 2008.

[27] The PostgreSQL Global Development Group: *PostgreSQL 8.4.2 Documentation*; 2009.

[28] A. Rowstron, P. Druschel: *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*; IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, Nov 2001.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan: *Chord: A scalable peer-to-peer lookup service for internet applications*; ACM SIGCOMM Conference, August 2001.

[30] SYSTAP, LLC: *Bigdata*; URL: http://www.systap.com/bigdata.htm.

[31] W3C: *OWL Web Ontology Language Guide*; W3C Recommendation, February 2004.

[32] W3C: *Resource Description Framework (RDF): Concepts and Abstract Syntax*; W3C Recommendation, February 2004.

[33] W3C: *SKOS Simple Knowledge Organization System Primer*; W3C Working Group Note, August 2009.

[34] W3C: *SPARQL Query Language for RDF*; W3C Recommendation, 2008.

Acknowledgments

The authors would like to thank all participants of the SciMantic project who have contributed to this deliverable through constructive discussions and feedback. This project is funded by The Innovation Promotion Agency CTI of the Swiss Federal Office for Professional Education and Technology (Das Bundesamt für Berufsbildung und Technologie BBT).