WeKnowIt

Emerging, Collective Intelligence for Personal, Organisational and Social Use

FP7-215453

D6.4.1

Integration of WP 1-5 tools and common components version 1

Dissemination level	Public
Contractual date of delivery	Month 18, 2009-09-30
Actual date of delivery	Month 19, 2009-10-23
Workpackage	WP6 Architecture and integration
Task	T6.3 Integration of tools and development of common components
Type	Prototype
Approval Status	Approved
Version	03
Number of pages	81
Filename	D641-man_2009-10-23_v03_smind_deliverable-integration_wp1_wp5_version_1

Abstract

This document accompanies the WeKnowIt (WKI) System prototype and presents the integration of services developed by WPs 1-5.

Both technical and non-technical aspects of integration are discussed. The internal parts of the prototype are described, with special attention to integrated services and the WKI DS API. The existing cooperation paths of services are presented. Also the installation and running of the prototype is explained.

Finally, initial integration of the user interface (UI) components with the WKI System is presented, and further extensions of D6.4.1 prototype, leading to M19 WP7 deliverables, are explained.

SEVENTH FRAMEWORK PROGRAMME

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1		Initial version	
0.2	21.10.2009	Final version.	Symeon Papadopoulos (internal reviewer)
0.3	23.10.2009	Final, revised version.	

Author list

Organization	Name	Contact Information
SMIND	Andrzej Boruch	andrzej.boruch@softwaremind.pl
SMIND	Rafał Janik	rafal.janik@softwaremind.pl
SMIND	Tomasz Kaczanowski	tomasz.kaczanowski@softwaremind.pl
SMIND	Jowita Kwiecińska	jowita.kwiecinska@softwaremind.pl
SMIND	Paweł Wesołowski	pawel.wesolowski@softwaremind.pl
USFD	Neil Ireson	N.Ireson@dcs.shef.ac.uk

Executive Summary

The "Integration of WP 1-5 tools and common components version 1" deliverable presents the integration of components in the WKI System. This prototype is a basis, on which the further work towards the WP7 1st prototype deliverable (D7.3.1 and D7.4.1 in M19), will be conducted.

D6.4.1 deliverable is based on the WKI System architecture (described in D6.1.2 "Identification of architecture elements and relations" deliverable) and on the services provided by all WPs (WP1-WP6). Integration is possible thanks to the existence of:

- the Service Oriented Architecture (SOA) and the OSGi[1] environment, which constitute the basis of technical integration (detailed explanation can be found in D6.1.2),
- the Common Model (Section 3), which specifies the main entities used through the system and makes it possible to exchange data between services,
- the WKI Data Storage (WKI DS), responsible for storing all data of the WKI System (Section 4).

This report describes the integrated WKI services and the scenarios that present the interactions between them. It explains the role of the Common Model and of the WKI DS in the integration. The initial integration of the WKI System with the UI components provided by use case applications is discussed. Also technical details of the installation and usage of the prototype are provided.

[1] http://www.osgi.org

Abbreviations and Acronyms

API	Application Programming Interface
CAP	Community Administration Platform
CDL	Community Design Language
CI	Continuous Integration
COMM	Core Ontology for Multimedia
CSG	Consumer Study Group use case
DFS	Distributed File System
DTO	Data Transfer Object
ER	Emergency Response use case
ESB	Enterprise Service Bus
FOAF	Friend-of-a-Friend
FSCR	File System Content Repository
GML	Geography Markup Language
GNU LGPL	GNU Lesser General Public License
HTTP	Hypertext Transfer Protocol
ID	Identifier
JAR	Java Archive
JCR	Java Content Repository
JSON	JavaScript Object Notation
M3O	Multimedia Metadata Ontology
MPEG-7	Multimedia Content Description Interface
NLP	Natural Language Processing
OGC	Open Geospatial Consortium
ORM	Object-relational mapping
OSGi	Open Services Gateway initiative
POI	Point of Interest
POJO	Plain Old Java Object
QA	Question and Answer
RDBMS	Relational Data Base Management System

RDF	Resource Description Framework
RDFa	Resource Description Framework - in - attributes
REST	Representational State Transfer
RSS	Really Simple Syndication
SIOC	Semantically-Interlinked Online Communities Project
SOA	Software Oriented Architecture
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SSL	Secure Sockets Layer
SVN	Subversion
UI	User Interface
URI	Unique Resource Identifier
URL	Unique Resource Locator
W3C	World Wide Web Consortium
WKI	The WeKnowIt project
WKI DS	The WeKnowIt Data Storage
WP	Work package
WS	Web Services
XML	Extensible Markup Language

Table of Contents

List of Figures

List of Tables

List of Code Samples

1. Introduction

The integration of the WKI services has two main aspects. The first of them is related to the architectural model of the WKI System (presented in D6.1.2 "Identification of architecture elements and relations" deliverable) and refers to the technical means, by which the services of the WKI System exchange messages and access shared resources. This integration is based on the set of underlying technologies that were presented in D6.1.2.

The other side of integration is related to the position of the integrated services within the WKI System and the role of the integration for the both use cases (Emergency Response and Consumer Study Group).

This document addresses both issues, and intends to present the full picture of services integration.

First, it provides a description of the available services along with the functionality offered by them. Then, the Common Model – which serves as a basis for data exchange between the services – is presented. This section is followed by the description of the WKI Data Storage (service responsible for storing all the data of the WKI System). Next, possible integration scenarios, based on the requirements of the WKI use case applications, are presented. Finally, some technical information on the installation of the prototype software is included.

2. Integrated services

This section describes the services that are integrated in the WKI System.

2.1. Integration models

D6.1.2 provides a detailed plan of the WKI service development. The resulting services are OSGi-compatible, and are deployed into the OSGi environment provided by the WKI System. The communication between services is handled internally by the WKI System via the OSGi registry.

At the same time, the architecture of the WKI System is not limited to the OSGi services integration. Web Services (WS) are also a valid way of integration of services by:

- enabling them to access the system via WS calls,
- making them accessible through WS calls.

Such services fit perfectly in the integration plan of the WKI System. In fact, implementation of WS calls and WS endpoints is straightforward, as such an option is directly supported by the underlying Fuse ESB[2] framework.

The downside of using WS is that there is an inevitable performance penalty caused by the overhead related to the message exchange protocol (encoded in XML) over the net.

The majority of services provided so far follows the original development plan and relies on the OSGi integration. There are also services, that adopted a different development model, and are accessible solely as WS (in particular using REST calls).

For the OSGi-based services, a wiki page was prepared that gathers information required to use each service. The aim is to facilitate the usage of services by the project partners by providing snippets of Maven's[3] pom.xml (dependencies) and Spring[4] configuration files (beans declarations). The content of this wiki page is included in Appendix C.

[2] http://fusesource.com/products/enterprise-servicemix/

[3] http://maven.apache.org

[4] http://www.springsource.com

2.2. List of services

Service name	WP responsible	Partner(s) responsible
WP1_AccountManager	WP1	USFD
WP1_LogIn	WP1	USFD
WP1_ManageItem	WP1	USFD
WP1_Tag	WP1	USFD
WP1_Comment	WP1	USFD
WP1_Rate	WP1	USFD
WP2_Text_Classification	WP2	USFD
WP2_Text_Clustering	WP2	USFD
WP2_Text_Annotation	WP2	USFD
WP2_VisualAnalysis	WP2	CERTH-ITI
WP2_TagProcessing	WP2	CERTH-ITI
WP2_TagNormalization	WP2	CERTH-ITI
WP2_SpeechIndexing	WP2	BUT
WP2_SearchInSpeech	WP2	BUT
WP3_LocalTagCommunityDetector	WP3	CERTH-ITI
WP3_AnswerSpamDetector	WP3	CERTH-ITI
WP3_DetectLatentTopics	WP3	UOKOB
WP3_SearchPlaces	WP3	Yahoo!
WP3_GetPOIs	WP3	Yahoo!
WP3_InformationPlacePOIsEtc	WP3	Yahoo!
WP4_CommunityDesignLanguage	WP4	EMKA
WP4_Cat_Algorithms	WP4	EMKA
WP5_LogMerger	WP5	CERTH-ITI
WP5_GroupManagement	WP5	UOKOB
WP6_DataStorage	WP6	SMIND

Table 1 Integrated services

Significant effort was put into making the services, interoperable by choosing a "common model" that they would use. This Common Model is a set of entities (Java POJOs) that are exchanged between services. A more detailed explanation of the common model and its role can be found in Section 3. The Common Model does not apply to all services. Also, not all services that are meant to use it are already doing so. In the future, more services will use the Common Model entities for exchanging information.

Below, important concepts used in the descriptions of services are listed. Note that these definitions describe their meaning in terms of the services but not the meaning in general English.

- **Place:** is a broad location such as continent, country, state, or city.
- **POI:** is a point of interest such as a landmark, tourist attraction, museum, etc.
- **Event:** is an event taking place at a particular location at a particular time.

2.3. WP1 services

This subsection lists services provided by "Personal intelligence" (WP1).

2.3.1. WP1_AccountManager

This service allows a new user to register in the WKI System. Once the user has an account, he is able to access the other services provided by the WKI System. This is centered on the OpenId[5] standard and the concept of a federated identity. The federated identity is one which spans multiple information systems.

2.3.2. WP1_LogIn

The login service is the main entry point to the rest of the services in the WKI System, provided a user has already created an account. It allows users to authenticate via a username and password or, preferably, through the use of the OpenId standard.

2.3.3. WP1_ManageItem

Deals with the initial upload of information into the WKI System. It is concerned with uploading and deleting items like photographs, videos, and documents in the WKI DS. Users may also state the permission level of the item to be uploaded: public, private, or shared amongst signified communities.

[5] http://en.wikipedia.org/wiki/OpenID

2.3.4. WP1_Tag

The tag service allows users to add, change and delete tags that have been assigned to the uploaded items. This leverages WP2 services to make tag suggestions.

2.3.5. WP1_Comment

This is another major data entry point in the WKI System. To support this user action, this service allows to add a textual description of an uploaded item, and also to allow other users to add more comments in order to generate a discussion about items.

2.3.6. WP1_Rate

The goal of this service is to allow users to give a numeric rating from a predefined set of values to an item they will upload or they have previously uploaded.

2.4. WP2 services

This subsection lists services provided by "Media intelligence" (WP2).

2.4.1. WP2_Text_Classification

The goal of this service is to determine the similarity between a given text (or set of texts) and some pre-defined language models relating to different categories.

The pre-defined language model is created as part of the offline text classification process, using a sample of classified training data provided by the (WP7) application providers. A category model is constructed from a collection of example texts relating to that category (e.g. reviews and not-reviews). The modelling tool creates a collection of category models which can be used by the classification service to classify a text.

2.4.2. WP2_Text_Clustering

To cluster a set of texts into related clusters, this can be used to help the user focus on similar documents.

Texts are clustered according to their content. The clustering process applies NLP[6] techniques to normalise the text and then group texts containing similar terms into clusters. Therefore, a cluster can be seen as the set of (weighed) terms, derived form the texts being members of the cluster, which discriminate that cluster from the others.

[6] http://en.wikipedia.org/wiki/Natural_language_processing

2.4.3. WP2_Text_Annotation

This service automatically annotates entities in a text document, using some pre-defined annotation model.

For each entity type (e.g. geospatial, temporal) an annotation model is developed (offline). These entity models are then applied to the document in order to recognise and annotate entities, adding this information to the document metadata.

2.4.4. WP2_VisualAnalysis

The aim of this service is to exploit the visual content within the WKI system in terms of visual similarity matching, retrieval and localization.

Users of the WKI System will be able to search and retrieve information about specific landmarks or POIs efficiently during their trip or during an emergency event. Furthermore, the service facilitates the way they share personal multimedia content acquired during their trips or emergency situations. More specifically, raw multimedia content (e.g. still images acquired by digital cameras or mobile phones) uploaded directly from emergency event sites will be analyzed and its information exploited towards efficient, automated and quick identification of objects, landmarks or events of interest.

2.4.5. WP2_TagProcessing

This service supports the exploitation of textual tags derived from visual content within the WKI System in terms of tag matching and similarity. By utilizing the additional textual information, WKI users will be able to enhance the visual retrieval and the localization of specific landmarks or POIs efficiently during their trips or emergency events. More specifically, the list of suggested tags for the specific image are paired with known landmarks in the proximity of the location the image was captured (as listed in the Geonames[7] list), or discarded. The list of tags that persist through the pairing process, are replaced with a more formal form (capitalized where appropriate, etc.) and returned to the system/user.

2.4.6. WP2_TagNormalization

Tags that have been assigned to each resource are matched to one or more domain ontology concepts. Each matching is accompanied by a weight coefficient that shows the degree of the tag-concept relation. For the appropriate mapping between tags and formal descriptions, external sources of knowledge (such as WordNet, Wikipedia, etc) are exploited.

[7] http://www.geonames.org/

Furthermore, string processing, matching and comparison functionalities are employed. The direct benefit of this process is that it is going to yield interoperability to our dataset and allow for performing reasoning operations. Moreover, it is expected to enhance the results of the provided visual analysis in terms of landmarks or POIs identification.

2.4.7. WP2_SpeechIndexing

The service prepares an index for the search in a given set of speech recordings and provides a link (URI) to this index for subsequent searches.

The set of audio file IDs is processed in the given order. The service also checks whether the recording file type is supported. The settings parameter is taken into account next. The values of specified parameters are set according to the provided data. If a particular parameter is not specified, the pre-defined default value is applied. Then, the service reads the content of the provided speech files. Based on the given settings, the service can retrieve a set of relevant metadata to improve the indexing process. The speech data is transformed and finally indexed. The service generates and returns a unique URI for the index.

2.4.8. WP2_SearchInSpeech

The service evaluates a query on the identified speech resources and returns the most probable hits.

The user query is analysed first. The service checks the format of the given query string, parses it and prepares the actual search. The given list of speechindexURIs is compared to the content of the internal repository and the links to non-existing indices are reported. If the list of speechindexURIs is empty, the service performs the search on the entire collection of the processed recordings. Service-specific settings are parsed and the search-related parameters are set accordingly. Based on the given settings and the query, the service retrieves relevant metadata and prepares filtering of the search results. Finally, the set of keywords is searched in a given list of previously indexed speech recordings. The service returns a list of the parts from the relevant files that most probably correspond to the given query. The list is sorted according to the confidence estimation.

2.5. WP3 services

This subsection lists services provided by “Mass intelligence” (WP3).

2.5.1. WP3_LocalTagCommunityDetector

Given some input tag, the goal of this service is to identify a collection of tags that form a community around it. A community of objects is defined with respect to a graph of objects. In our scenario, we consider a tagging system where users tag resources (e.g. questions, answers, pictures or other web objects). Based on the tag co-occurrences, a network of tags is created (e.g. if a user tags an object with tags "cars" and "BMW", then a link is created between these tags in the network). By processing this network, the service extracts groups of tags that are closely connected with each other and less connected with the rest of the network. In most cases, this process corresponds to identifying topics in the tagging system. The special thing about the service is that it operates at a local level, i.e. it processes only a small part of the graph in order to output the identified community, thus it is suitable for use in interactive applications.

2.5.2. WP3_LexicalSpamDetector

This service flags a piece of text (e.g. comment, post) as lexically spam.

The service is intended for use in the context of filtering user contributed text content before presenting them to the interface or before another service requests them for processing. The service operates on generic text fragments by checking their conformance to statistical models generated from large corpora of well-formed documents.

2.5.3. WP3_DetectLatentTopics

The service finds relevant latent topics for a given document: question or answer. Each latent topic has a list of related questions and answers so, indirectly, the service allows to find documents in the system that are related to the provided input.

Given the input (text in question, question and tags, or answer), the service classifies it into previously calculated latent topics.

2.5.4. WP3_SearchPlaces

This service supports searching for a particular place. If a user indicates that she wants to get information about London, this service will give information about different places called London: London, England and London, Canada. The service provides both geo-coordinates and identifiers that can be used to retrieve further information from other services.

Currently, only Places (see definition above) are supported. In the future the service may be extended to constitute a more general search function.

2.5.5. WP3_GetPOIs

This service returns a ranked list of POIs for a particular Place. Say you are travelling to London, England and you want to get a list of POIs in the city. The service returns a ranked list of POIs. For each POI, the service returns geo-coordinates and identifiers that can be used to get further information about the particular POI.

2.5.6. WP3_InformationPlacePOIsEtc

This service is used for retrieving detailed information about a Place, POI, etc. The service returns geo-coordinates and a short description of the Place, POI, etc. It also provides identifiers of resources from external services - Yahoo! Geo Planet and Wikipedia - that can be used to collect further information on the particular Place/POI.

2.6. WP4 services

This subsection lists services provided by "Social intelligence" (WP4).

2.6.1. WP4_CommunityDesignLanguage

The service calls the Community Administration Platform (CAP, described in deliverable D4.2 "Prototype of the community administration platform") via the structured language Community Design Language (CDL). Via CDL expressions the following commands can be executed:

- define policies,
- define access rights,
- check access rights,
- report about policies and access rights,
- get help on CDL.

2.6.2. WP4_Cat_Algorithms

This service provides methods for the analysis of social networks. The service calculates for input social networks (graphs) general graph statistics, centrality measures, eigensystem decompositions and clusters.

2.7. WP5 services

This subsection lists services provided by "Organizational intelligence" (WP5).

2.7.1. WP5_LogMerger

The service merges a set of event logs referring to the same incident to a single log, and allows targeted searching within the merged log. In practice, this service is useful in the post-event stage of the ER use case, when the logs, which have been created by the ER personnel, are collected and controlled (for review and audit purposes). Frequently, there are multiple log files created by different people that refer to the same emergency event. This service enables automatic merging of the logs and ordering of the respective log entries (based on time), as well as indexing of the log entries, so that they can be searched based on keyword, person, location, ER function, and action.

2.7.2. WP5_GroupManagement

The aim of this service is to define, modify, and delete organisations and groups, i.e., structures consisting of people and other groups, within the WKI System. These groups can be represented in a distributed manner. This also means that other groups managed in other systems can be referenced and included. With the group management the role of the individuals and groups such as the position of a user in the organisational structure, his organisational profile (e.g., skills), and others can be specified. The service allows for defining organizations in ER beforehand, i.e., the emergency response entities can be modelled before the incident happens. Once these organisations for ER are defined, this service can be leveraged in the case of a concrete incident to set up a virtual organisation for ER (see requirements stated in deliverable D7.1 "Consumer and emergency response use case initial requirements").

2.8. WP6 services

This subsection lists services provided by "Architecture and Integration" (WP6).

2.8.1. WP6_DataStorage

This service enables the storage/retrieval of data from the WKI Data Storage. The service API covers all components of the WKI Data Storage: namely triple store, database, object and file storage. Service was designed to be able to deal with variety of data types used by all services living inside the WKI System. The major goal of the service is to provide an easy-to-use, high-level API for other services. See Section 4 for more information about the WKI Data Storage.

2.9. Development model

The majority of services were developed according to the development plan presented in D6.1.2 "Identification of architecture elements and relations, version 2". The elements of this plan are listed below:

- projects were created using common Maven pom files:
  - weknowit-bundle-pom - Maven archetype[8] that provides a project starter,
  - weknowit-parent-pom - contains common configuration for build process, reports, plugins and enforces use of certain versions of libraries and components,
- all source code is stored in the common code repository (SVN[9]),
- issues are raised using bug tracker software (Mantis[10]):
  - problems with the WKI System,
  - problems with services (e.g. compatibility issues),
  - infrastructure issues (e.g. server downtimes),
- all artifacts are stored in the common artifacts repository (Nexus[11]),
- artifacts are created on the Continuous Integration Server (Hudson[12]),
- artifacts are tested on their compatibility with the WKI System by developers (on local machines) and by WP6.

For a detailed explanation of the elements listed above, refer to D6.1.2.

During the development of services, some enhancements were made to the development process. Sonar[13], a quality reporting tool, was integrated in the CI server. It is used as a primary source of software metrics and quality checks of the services source code.

2.9.1. Typical development flow

This subsection provides an insight on development process by presenting a typical flow of actions, that occur in case a bug is found. It also presents how various elements of the development infrastructure facilitate the flow

[8] http://maven.apache.org/plugins/maven-archetype-plugin/

[9] http://subversion.tigris.org/

[10] http://www.mantisbt.org/

[11] http://nexus.sonatype.org/

[12] http://hudson-ci.org/

[13] http://sonar.codehaus.org/

of information between project partners, and allow to synchronize the development efforts of the distributed WeKnowIt team.

Let us assume, that the code of some service was written and committed to the SVN repository. This service was build by CI server (Hudson) and used by other partners. One of the partners has found a bug. Figure 1 presents actions, that follow the finding of this bug.

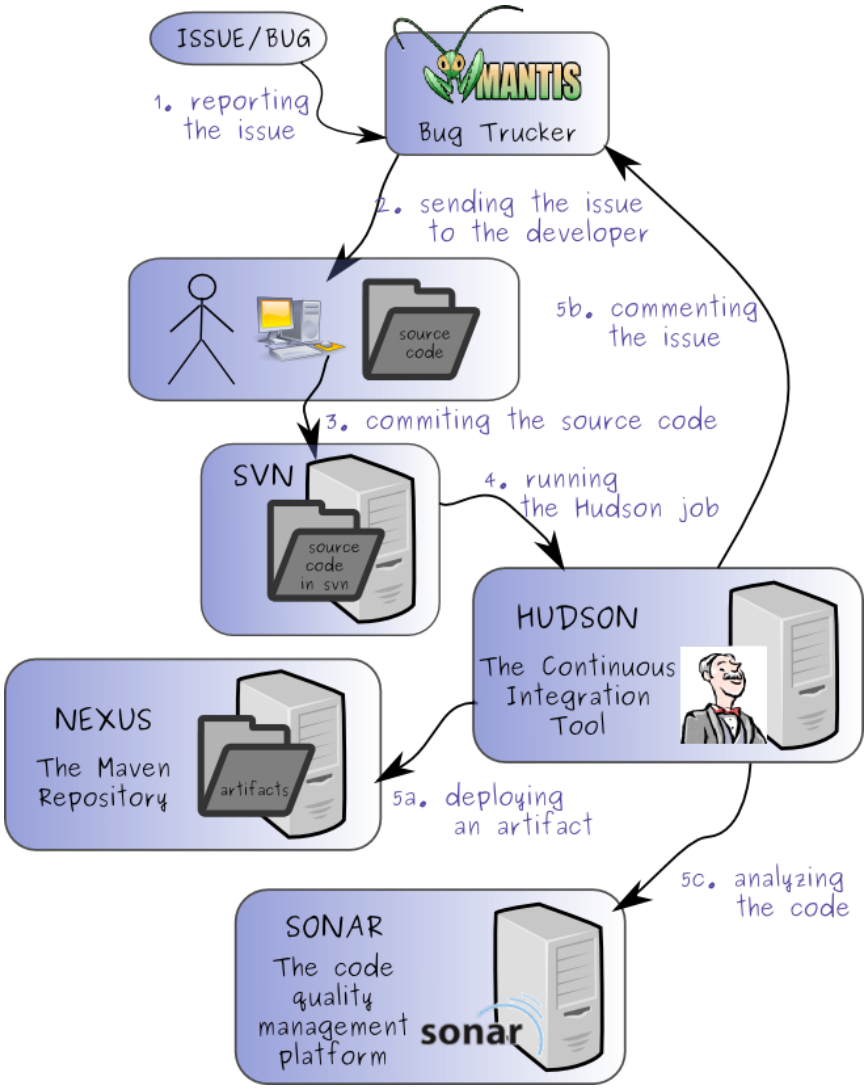

Figure 1 Development process

First, an issue is raised using bug tracking software (Mantis). The responsible developer is notified automatically by Mantis (each category of issues has one responsible person assigned). The developer can

- contact the reporter and discuss the issue,
- assign the issue to a better suited person,
- fix it herself.

In the case depicted on the Figure 1, the developer fixes the code herself. She adds a meaningful comment while committing the patch to the SVN code repository, which makes the Continuous Integration server (Hudson) automatically comment the issue on Mantis (thus making the reporter aware of the committed fix). Hudson builds the artifact with the new patch and deploys it to the Maven repository (Nexus).

The reporter can now download and use this bug-free version. The download will be performed automatically by Maven[14].

Additionally, Sonar (a quality management tool) executes various code checks, which generate reports that are made available online for the convenience of the developers.

2.9.2. Guidelines

During the development of services, the guidelines (presented in D.6.2.1 "Definition and implementation of APIs, version 1") were enhanced, and some new guidelines were added.

Currently, the following guidelines are available:

- How to install Java & Maven (ver 0.6),
- How to install the WKI system (ver 0.9),
- How to create Maven project (ver 0.9),
- How to upload missing JAR to Maven repository (ver 0.2),
- How to create OSGi service using Spring Dynamic Modules (ver 0.9),
- How to deploy OSGi bundles to the WKI system (ver 0.2),
- How to use Hudson CI (ver 0.3),
- SVN project layout (ver 0.2),

WP6 constantly updates the guidelines based on the needs of the development teams. Also, all the changes in the development infrastructure are reflected in the guidelines.

All the guidelines can be downloaded from the main project wiki page. Each guideline includes a "history" table, which informs about the changes made in the successive versions of the document.

[14] Assuming the version of fixed service has not changed. In other case, appropriate dependency entry in the pom.xml file has to be updated.

3. The Common Model

In general, the integration of the WKI services is performed on two levels – technical and conceptual. Technical issues are addressed by the SOA and the OSGi environment of the WKI System (described in the D6.1.2 deliverable). The other aspect relates to the semantic meaning of the data exchanged by the WKI services. The Common Model is related to both issues: it contributes an additional technical fragment to the WKI System architecture and, at the same time, constitutes a semantically consistent means of data exchange.

3.1. Introduction

The Common Model is a fundamental component in the development and integration of the WKI system. It provides specifications for the information that is communicated: to/from the users of the system and amongst the services provided by the Intelligence Layers. In addition to that, it provides a means of integration of information directed to/ originating from external sources. The Common Model must satisfy these three types of requirements: user, service (or technologically-driven) and interoperability.

A. User Requirements: This aspect of the Common Model is dealt with by the Interaction Model (see D1.2 "Personal Knowledge Management Technologies"). The user requirements, provided by WP7, are translated into certain functionalities, which are provided via the WP1 services, such as, e.g. tagging, reviews & rating, user profiling and modelling. These functionalities impose further requirements upon the Common Model to represent the information provided by the user and the information necessary for the system to satisfy the needs of the user.

B. Service requirements: In order for the WKI services to perform their functions, service-specific information needs to be modelled and exposed. For example, some Media Intelligence (WP2) services generate confidence weights associated with the metadata, which are automatically assigned to the input resource. Whilst this information may not be directly accessed by the user or exported to external systems, it is necessary to represent it in the Common Model, as it is used by the internal processes of the WKI system and determines what information is received by the user (or an external system).

C. Interoperability: This aspect of the Common Model is largely dealt with by the Knowledge Sharing Methodology task in WP5 (see

5.3.1). It may be desirable to receive, or provide information to external systems, in order to achieve this, the WKI Common Model is built upon the usage of standard ontologies. This enables external systems, which use these ontologies, or which can translate information represented in such standard ontologies to/from their internal representation, to share knowledge with the WKI System.

In terms of the architecture and integration of the WKI System the Common Model addresses both technological and semantic issues. That is, the model is used to derive the POJOs, which are communicated between the User Interface, Services and the WKI Data Storage. The semantics behind the POJOs and their variables are defined via their explicit links to the ontologies.

A special interest group was set up in order to solve the issue of the Common Model. Due to the fact that the issues addressed in deriving the Common Model span multiple tasks within the WPs 1-6, the group consists of partners representing CERTH (WP2,3), UoKob (WP4,5), USFD (WP1,2,5) and SMIND (WP6). The group communicated via emails and teleconferences. The WKI pages are used to document the results of the discussions and work that was done; in addition, some preliminary code was exchanged via the SVN repository.

Although the initial concern of the group was to derive the Common Model necessary to meet the functional requirements of the first prototype, from the onset, there was a desire to adhere to common standards as far as possible: both to address the need for interoperability and to exploit the results of previous related efforts.

3.2. Common Concepts

The first assignment of the group was to form a list of common concepts, which require representation, primarily ensuring that these satisfied the user requirements, specified by WP7 and abstracted in the WP1 services. For each concept candidate ontologies were identified and assessed for their suitability, according to their representational coverage (i.e. do they include all the properties that require representing) and their semantic match (i.e. does the meaning of the ontological concept and properties equate to the meaning used within the WKI System). The following subsections discuss each of these main concepts.

3.2.1. Event

An event (or series of interconnected events) can be considered to be a fundamental unit of Collective Intelligence in the WKI system. That is, the individuals and organisations involved in an event add to the knowledge of

that event, generating a collective understanding. The user requirements on event are to represent:

- Location: where is the event taking place, this can be a point or an area.
- Time: when did the event start and end.
- Participants: who is involved in the event, and what is their role.
- Documentation: the resources attached to the event, which provide information or confirmation.

The obvious candidate for the representation of an event is the Event-Model-F[15]. Not only is this model actively developed (by one of the WKI partners, UoKob) but also covers all the necessary representational requirements, unlike simpler event ontologies[16], or domain specific ontologies, such as those derived for scholarly events[17] or biological processes[18].

For temporal data, the requirements are relatively simple and thus most standards will suffice. The W3C OWL-Time[19] Ontology for describing the temporal content of Web pages and services was selected.

For geo-spatial data, the most widely adopted representation is the Geography Markup Language (GML) defined by the Open Geospatial Consortium (OGC)[20]. For Web resources, GML is being increasingly wrapped in the GeoRSS GML[21] Application Profile, to provide a standard coordinate system. The W3C Geospatial Incubator Group[22] provides an ontological representation of the GeoRSS standard.

3.2.2. Document

The requirements on representing documents for the first prototype simply involve single media types (i.e. text, images or audio). It was initially intended to use the Core Ontology for Multimedia (COMM)[23], which

---

15 http://www.uni-koblenz-landau.de/koblenz/fb4/institute/IFI/AGStaab/Research/ontologies/events/index_html

16 http://motools.sourceforge.net/event/event.html

17 http://eventography.org/sede/

18 http://obofoundry.org/cgi-bin/detail.cgi?event

19 http://www.w3.org/TR/owl-time/

20 http://www.opengeospatial.org/

21 http://www.georss.org/gml

22 http://www.w3.org/2005/Incubator/geo/XGR-geo/

23 http://comm.semanticweb.org/

is largely an ontological representation of the MPEG-7[24] standard. Whilst this, highly expressive, ontology does offer the ability to cover the identified representation requirements, it was also argued that MPEG-7 is mainly aimed at covering the decomposition and description of low-level features of audiovisual media content. Fortunately, COMM is being followed up by the M3O project[25], which abstracts away from a single standard (such as MPEG-7) and provides a more coherent infrastructure to represent both high-level semantic annotation with background knowledge as well as the annotation with low-level features and therefore is better suited to the WKI requirements. Note that both COMM and M3O are developed by one of the WKI partners (UoKob).

3.2.3. User

The requirements for representing users in WKI include their personal details and preferences (profiles) and their social interactions. These requirements are largely met by the most widely used ontology for representing community information, the Semantically-Interlinked Online Communities (SIOC) initiative[26]. The SIOC ontology is commonly used in conjunction with the Friend-Of-A-Friend (FOAF) vocabulary[27] for expressing personal profile and social networking information.

Within the SIOC there are a number of concepts that can be utilised to satisfy the requirements of WKI. These include user roles, groups and as well as tags, comments and categories to "items" created by other users.

3.2.4. Tag

For the representation of tags the CommonTag[28] format was adopted as it is well supported (including the search services provided by the WKI partner, Yahoo!) and provides a means to represent well-defined concepts necessary for supporting semantic tagging within WKI.

3.3. Interaction Model

The Interaction Model, developed by the Common Model group, is described in D1.2, as its primary motivation is ensuring that the Common Model can represent the needs of the users as realised by the Personal Knowledge Management Technologies implemented in WP1. In terms of

---

[24] http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm

[25] http://semantic-multimedia.org/index.php/M3O:Main

[26] http://sioc-project.org/

[27] http://www.foaf-project.org/

[28] http://www.commontag.org/Home

WP6 requirements the Interaction Model specifies the interconnection between the different ontologies and therefore, the references that exist within the implementation of the ontological concepts as POJOs within the WKI system. A snapshot of the current version of the model is depicted in Figure 2.

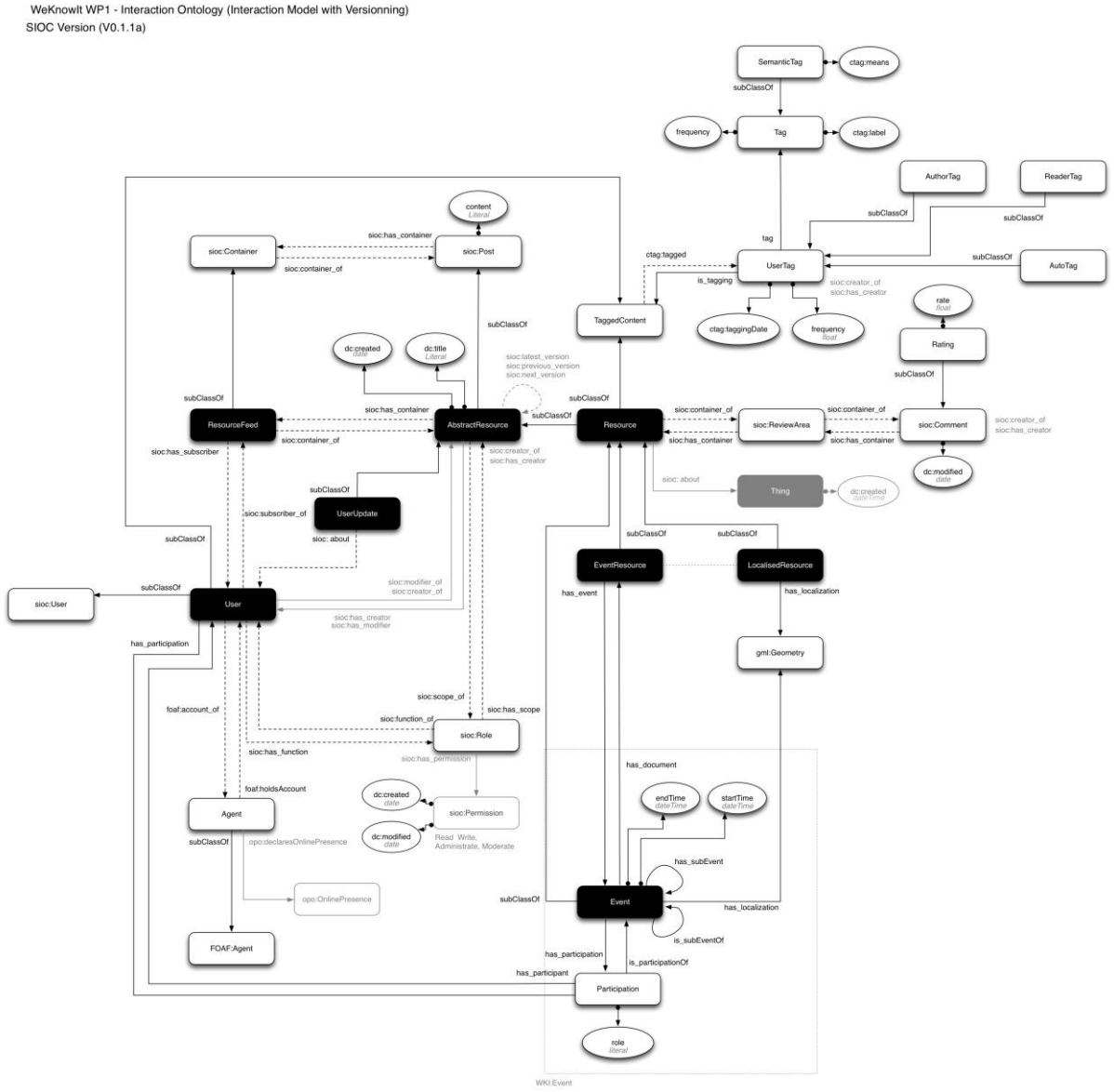

Figure 2 Interaction Ontology

3.4. Technical overview

Technically speaking the Common Model consists of two parts. The first part comprises the Data Transfer Objects (DTOs) that are exchanged between the WKI services (thus constituting a "common language" of the WKI System). The DTOs are clean Plain Old Java Objects (POJOs), and are available to all WKI services.

The other part, are the mapping classes – one per each POJO of the Common Model. Their role is to transform POJOs to triples (and vice versa). Please refer to the WKI DS architecture description (see Section 4) to learn more about the POJO/triples storages of the WKI DS. These mapping classes are hidden inside the WKI DS, and are not used by the WKI services.

POJOs (exchanged between services) are completely technology-agnostic, that is, they do not include any annotations that would connect them with any external framework. The mapping classes, on the contrary, are connected with the underlying triple store and need to be rewritten once it changes. Division into POJOs and mapping classes guarantees that such change will be transparent to the client services.

All common objects (Java POJOs) are gathered in the weknowit-commons-model project that is a part of the WKI System (see Figure 3). Services that exchange data with other services living inside the WKI System have to add this project as a dependency, to be able to use the types defined by the Common Model. Having all common objects in one project simplifies the management of data exchanged between services, and it facilitates the extension of the model with new types as they become available.

Figure 3 Objects of the WKI Common Model

4. The WKI Data Storage

This section contains the description of the WKI Data Storage. Even though, there was a separate deliverable dedicated to the WKI DS (D6.3.1 "Design, architecture and implementation of knowledge base" in M12), there are important reasons, why the report accompanying D6.4.1 prototype should also include information on the WKI DS.

First of all, the WKI DS is a primary point of integration. It is the destination and the source of information in the WKI System, which means, that almost no business activity can be performed without the need for cooperation with the WKI DS. Moreover, the WKI DS is also a service itself and is integrated on the same basis as the rest of the services. Finally, during the integration of services and work on the Common Model, the WKI DS has undergone a lot of comprehensive changes with repercussions for the whole WKI System, and thus the documentation of its current state is considered necessary and fit for the present deliverable.

4.1. Development

The development of the WKI DS was driven by the requirements stated by partners, with special focus on the requirements derived from the use-case applications. During the development (which was scheduled for months 1-18 of the project), it was obvious that some requirements would be yet to discover during the work on use-case applications. Because of this, the architecture of the WKI DS must be flexible enough to encompass such new requirements, or at least the modifications must be possible.

Due to this uncertainty that accompanied the development, the decision was taken to use some open-source frameworks that would provide basic functionality and write an additional layer that would combine and customize them.

Such an architecture model – in which basic functionalities are provided by third-party frameworks and are glued together by some additional layer – has several benefits. It permits the introduction of further changes and enables the focus to be placed on the crucial issues – for example the performance of the storages. Also, thanks to this additional layer it is possible to substitute elements of the architecture (as soon as more efficient/advanced implementations are available) – for example the object storage element – in a way that is transparent to the clients of the WKI DS.

At the same time the usage of the selected open-source frameworks added many virtues, which are typical for open-source software:

- high-level quality of components is ensured by thorough testing conducted by a large community of developers,
- documentation is available and regularly updated,
- support is provided by community members,
- potential for implementation customization due to availability of the source code.

The decision to develop the proprietary layer over existing frameworks prevailed over some other options, such as the usage of the Virtuoso Universal Server platform[29] which in fact provides both: the components and the layer that connects them together. The complexity of Virtuoso platform does not guarantee that the changes can be easily introduced, and that the software can be easily tailored to the needs of the WKI System (even taking into account the availability of the source code).

Based on this, and on the fact, that only a small subset of functionality offered by Virtuoso is required by the WKI System, the decision was taken to implement a proprietary solution. This allows to maintain the full control over the WKI DS, and ensures, that the WKI DS can be adjusted to any existing and further requirements.

Nevertheless, the documentation of Virtuoso was studied and some inspiration on the linkage between various storage modules and data formats was drawn from it.

4.2. Architecture

Logically the architecture of the WKI DS can be divided into three major parts[30], as presented in Figure 4:

- knowledge base,
- object storage,
- file storage.

The **knowledge base** is responsible for storage of the information (i.e. RDF triples). The **objects storage** takes care of storing Java objects (POJOs), while the **file storage** stores multimedia files.

[29] http://virtuoso.openlinksw.com/

[30] There is some inconsistency between the names used currently to describe the WKI DS, and the ones used in previous deliverables.

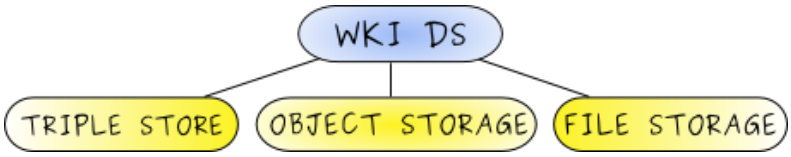

Figure 4 The WKI DS components

This logical division is, to some extent, reflected by the physical organisation of the WKI DS. The knowledge base is based on the triple store. The object storage is realized by use of an object database. The file storage uses both Hadoop[31] (for large files) and the native file system (for smaller files). Additionally, an RDBMS (PostgreSQL) is used for the storage of additional data (i.e. required due to some performance reasons).

Name	Version	Role
NeoDatis	Ver. 1.9.5	Object storage, cache
Jena	Ver. 2.6.0	Triple store
PostgreSQL	Ver. 8.3.8-1	Additional data
Hadoop	Ver. 0.18.3	Large files storage

Table 2 Elements of the WKI DS

4.3. Elements description

In the D6.4.1 prototype in-memory databases are used whenever possible in order to make the installation process easier. The limitations of in-memory databases are not problematic taking into consideration the amount of data stored within the prototype scenarios. Obviously, for the final WKI System installation, no in-memory databases will be used.

4.3.1. File storage

The file storage is responsible for storing all the files of the WKI System – especially the multimedia files that are a vital part of both use-case scenarios. File storage of the WKI DS is prepared to store various sizes of files – from small FOAF documents describing users to big multimedia files of many-megabytes size.

There are no significant changes in the file storage element of the WKI DS, since D6.3 (M12). The JCR[32] is not used any more (but still it can be

[31] http://hadoop.apache.org

[32] http://jcp.org/en/jsr/detail?id=170

easily integrated if required). Instead, the Hadoop DFS (for storage of the large files) and the native file system (FSCR, for small files) are used.

Hadoop DFS is well suited for distributed storage and processing and is designed to support streaming access to large files. Hadoop DFS offers unique features – it is fault tolerant, scalable, and simple to expand.

The reliability of Hadoop DFS is proved by an impressive list of applications and organisations using it[33].

Hadoop DFS gained a lot of interest as it is often used in conjunction with other software – e.g. MapReduce algorithms[34], search frameworks (i.e. Nutch web-crawler[35]), HBase database[36], sorting algorithms[37].

The overhead introduced by JCR was not matched by its functionalities, which are not really required by the WKI System, and because of this, its use has been considered as optional.

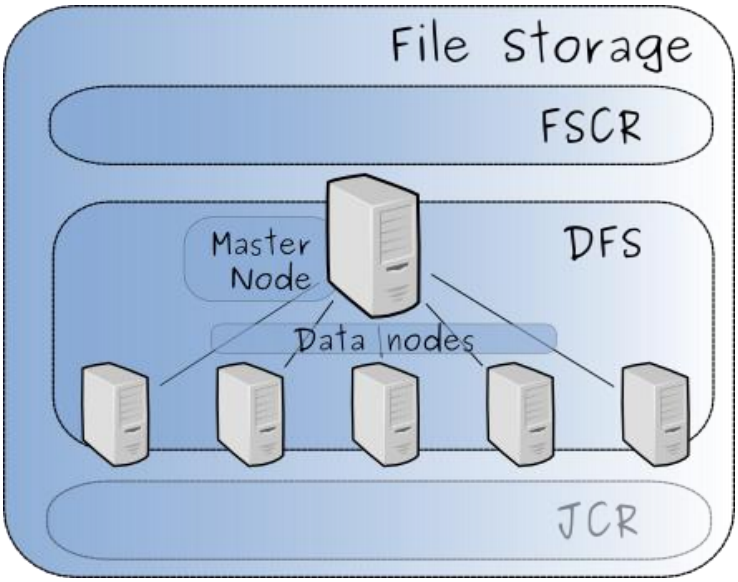

Figure 5 The WKI DS file storage architecture

In case of this prototype, only the native file system is used for storing files, in order to avoid the additional steps that would be required during the installation in case Hadoop was also used.

---

[33] http://wiki.apache.org/hadoop/PoweredBy

[34] http://en.wikipedia.org/wiki/MapReduce

[35] http://lucene.apache.org/nutch/

[36] http://hadoop.apache.org/hbase/

[37] http://developer.yahoo.net/blogs/hadoop/2008/07/apache_hadoop_wins_terabyte_sort_benchmark.html

4.3.2. Triple store

An essential element of the WKI DS is a triple store, of which the purpose is to maintain and serve all semantic data that is required by the WKI Services.

The Jena Semantic Web Framework[38], ver. 2.6.0, is used as a basis for the triple store. The flexible architecture of the WKI DS enables the substitution of Jena with some other triple store implementation (for example Sesame[39], Rdf2Go[40], 4store[41]) at the cost of rewriting some classes of the WKI DS, but without affecting the clients of the WKI DS.

4.3.3. Object storage

The object storage has several uses in the WKI DS. In general, its goal is to store POJOs from the Common Model.

The idea of the object storage comes from the fact, that many clients are interested in Java objects rather than in RDF triples. This is because Java objects are much more convenient to work with. This fact is leveraged by the Common Model, which provides common Java objects for all services to work with.

The other task of the object storage is to improve the efficiency of object retrieval from the WKI DS. In this case one could think of the object storage as a caching mechanism, of which the sole purpose is to provide fast access to objects stored in the WKI DS. The decision on which objects should be kept in the object storage is made by the WKI DS through an storage strategy (see 4.4.1).

A common solution for the storage of objects in the database is to use a relational database and a object-relational mapping (ORM) framework like Hibernate[42], TopLink[43] or iBatis[44], to translate objects into relational data and vice versa. Based on the requirements of the WKI DS object storage, the decision was taken to use an object database instead. There is no requirement for SQL queries – and still the advanced querying can be done using SPARQL queries against the WKI DS triple store. The use of an object database eliminates the need for object mapping (provided by ORM tools), thus making the component perform better (see [15]).

---

[38] http://jena.sourceforge.net/

[39] http://www.openrdf.org/

[40] http://semanticweb.org/wiki/RDF2Go

[41] http://4store.org/

[42] http://www.hibernate.org

[43] http://www.oracle.com/technology/products/ias/toplink/index.html

[44] http://ibatis.apache.org

Current implementation of this component is based on the NeoDatis Object Database[45]. In addition to the features described in the previous paragraph, NeoDatis directly persists objects in the way that they exist in the Java programming language, without any conversion.

Thanks to this approach, objects do not need to provide any additional information (such as annotations or interfaces like Serializable[46]) in order to be stored in the database.

From many available object databases, NeoDatis was chosen because of the following reasons:

- licensing policy (released under the GNU LGPL[47] license),
- active development and community,
- reliable and fast support from developers and community,
- good performance proved by benchmarks (performance comparison between NeoDatis and one of the most popular object databases db4o[48]),
- well-written documentation,
- simple, easy to use API.

Currently the WKI DS uses NeoDatis version is 1.9.5, which is a stable and mature release.

[45] http://www.neodatis.org/

[46] http://java.sun.com/javase/6/docs/api/java/io/Serializable.html

[47] http://www.gnu.org/copyleft/lesser.html

[48] http://www.db4o.com/

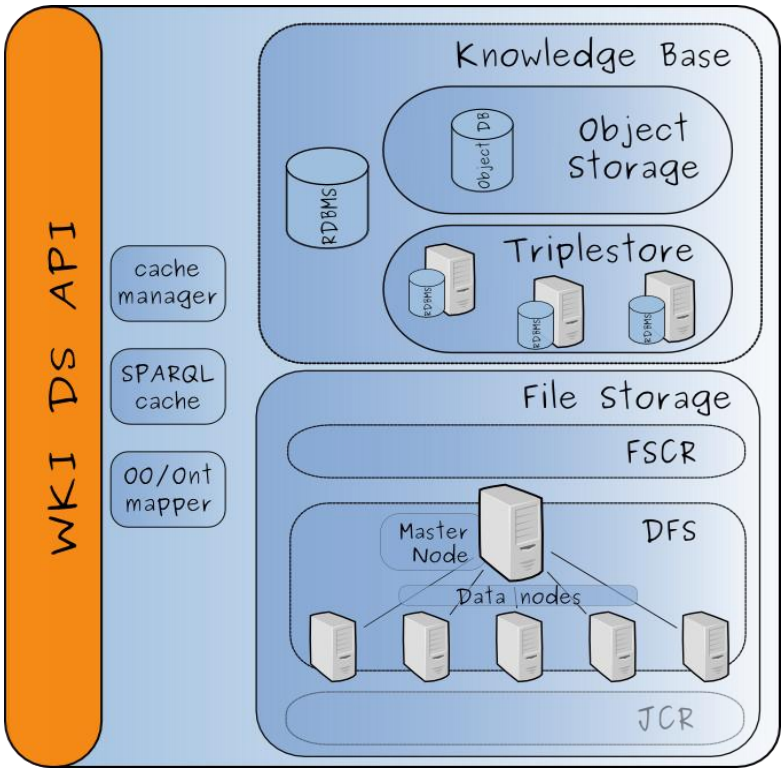

Figure 6 The WKI DS architecture

4.4. Important concepts

4.4.1. Strategies

Strategies are responsible for performing a given operation (store, update, get, remove, find) on objects from the Common Model within the WKI DS. Strategies distribute tasks between various components, which are parts of the WKI DS. The use of strategies simplifies the modification of the whole WKI DS module behaviour. With strategies, it is easy to adjust the WKI DS to specific requirements without affecting the rest of the code. Such behaviour is needed when the WKI DS is extended with new storage units. Without strategies this situation would require deeper code modifications. The advantage of interacting with individual storage elements through strategies is that most of the code outside the strategy is not aware of the underlying solutions. Thus, improvements in the implementation of strategies could significantly boost the performance of the operations performed on the WKI DS through more efficient use of the individual components.

Figure 7 presents three different "store strategies". Each of them results in an entity being stored in different storage elements. This Figure also shows, that the object of the Common Model must be converted using a specialized mapper utility before getting stored in the triple store.

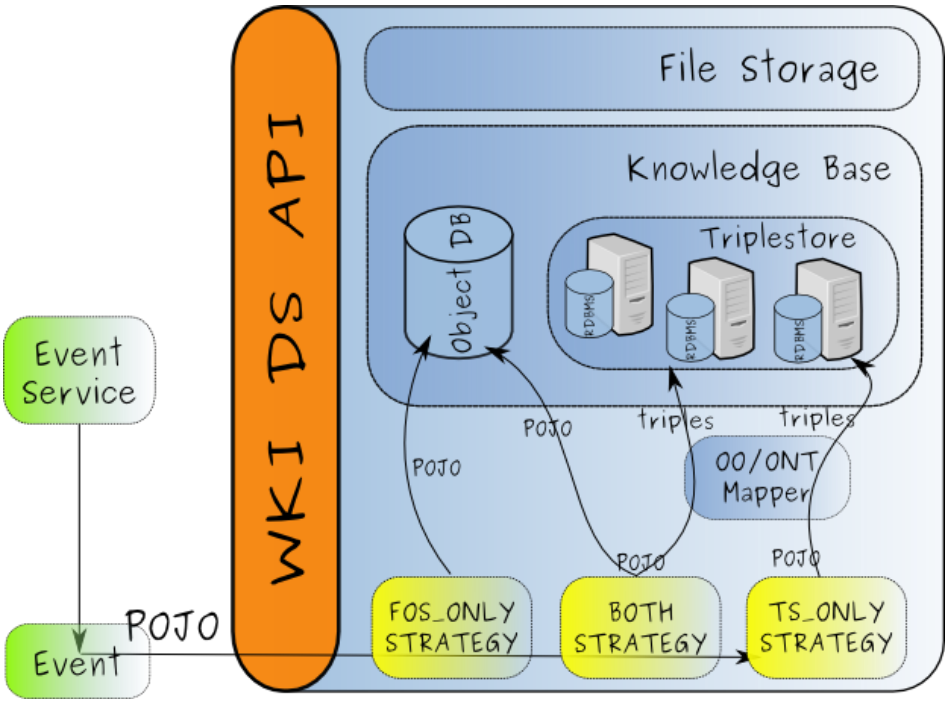

Figure 7 Storage strategies

The following types of strategies are available:

- store strategy,
- retrieve strategy,
- remove strategy,
- search strategy

The WKI DS can only use one strategy per type at the same time; however, strategies can be quite complex – for example they can be composed of several simple strategies.

Figure 8 and Figure 9 present two strategies:

- store strategy, which writes object in two storages – object storage and triple store,
- retrieve strategy, which retrieves an object, by first looking into the object storage and then, if nothing is found there, by trying to retrieve it from the triple store.

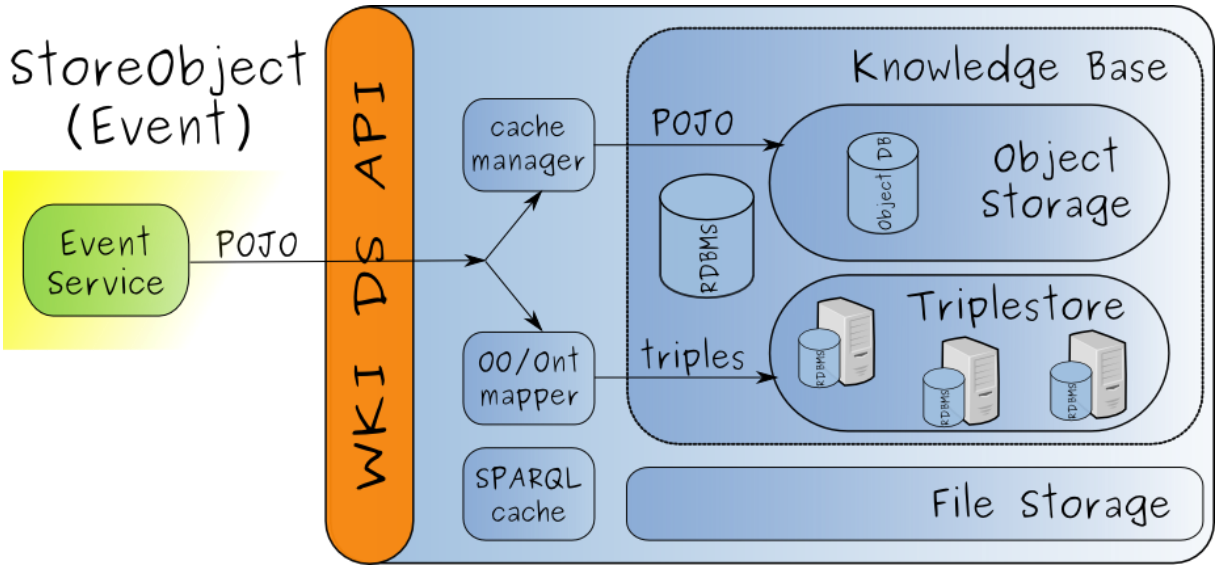

Figure 8 Storage strategy

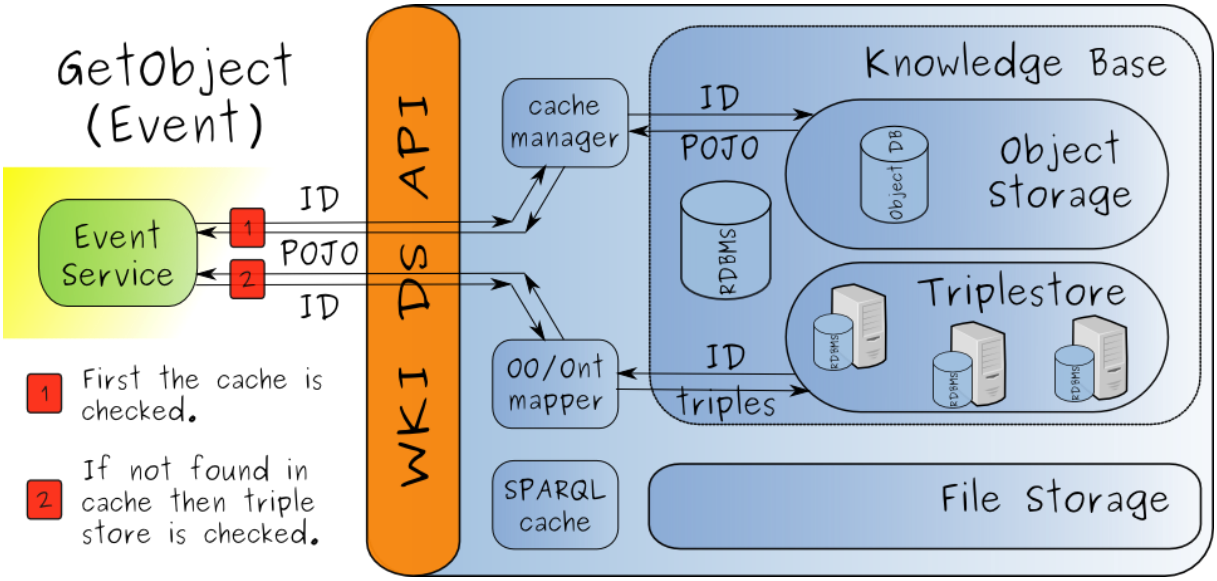

Figure 9 Retrieve strategy

4.4.2. Redundancy

The examples of strategies clearly show that it is possible to store some information in more than one storage element of the WKI DS. This redundancy is intentional. The main idea behind such behaviour of the WKI DS can be expressed with the well-known mantra of programmers "use the right tool for the job". The Object Storage is "the right tool" for fast retrieval of the POJOs. The triple store is "the right tool" for answering SPARQL queries. Usage of both, allows the WKI DS to perform well in both scenarios – both when a client needs to work with POJO, and when it is interested in retrieving triples.

The handling of redundant data is, in general, error-prone. The WKI DS deals with this problem by offering an API that forces clients to make use of POJOs rather than triples. This makes synchronization of data in both storage elements manageable.

4.5. API

The WKI DS API provides a common programming interface for the WKI DS, shielding client applications from the internal structure of the WKI DS. The API is the entry point to the WKI DS. All requests for storage or retrieval of data are served through this API.

Use of an API for the whole WKI DS module makes internal implementation of the WKI DS completely transparent to the services that make use of it. This approach enables the introduction of modifications and improvements to the internal structure of the WKI DS without affecting the implementation of any external client services.

During the work on the API of the WKI DS special attention was paid to the triple store related methods. Even though, currently the WKI DS uses Jena as a triple store solution, the API is written in such way, that it is possible to change it to 4Store or Sesame if required. Of course, some changes in the internal parts of the WKI DS would be required, but they would be transparent to the clients of the WKI DS API.

All the methods of the WKI DS API throw `WkiDataStorageException` exception. For the sake of brevity this was omitted in the method signatures. Also, because some of the methods of the WKI DS API are just for the convenience of clients, and do not offer any new functionality (e.g. having less parameters and assuming some default values), they are not presented in the following sections.

4.5.1. The model part

Methods presented in Table 3 allow the WKI DS to perform pre- and post-work actions. Usage of these methods is obligatory, and important for internal processes to ensure correct and effective work of the data storage module.

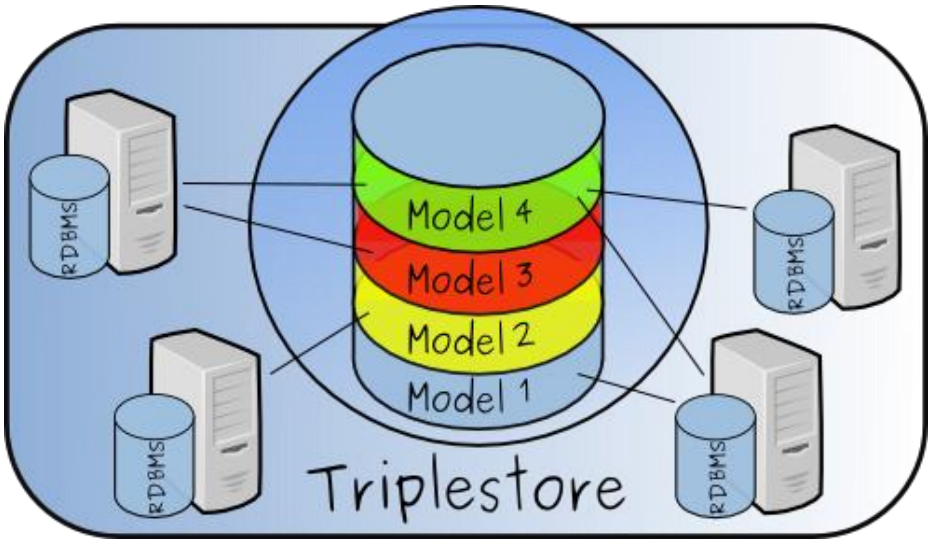

Figure 10 Storage divided into models

The storage is logically divided into models. A model can be described by the analogy to the relational database scheme. Different data can be stored in different models, which helps to impose some structure to the data model, and can be also used to restrict the scope of search queries. Each model is uniquely identified by its URI, which is used in methods signatures presented in Table 3.

Method signature	Description
`void createModel(String modelUri)`	Creates new model in the triple store with specified URI. If a model with the given URI already exists, then no action is carried out.
`void startModel(String modelUri)`	Opens database connection to the model with the given URI. If a connection with this model already exists, nothing is done. Throws exception if no such model exists.
`void stopModel(String modelUri)`	Closes connection to the model with the given URI. If connection with this model is already closed, then nothing is done. Throws exception if if no such model exists.

Table 3 WKI DS API – model related methods

4.5.2. The common object part

The WKI DS API is designed to interact with the rest of the WKI System mainly through objects from the Common Model (see Section 3). This approach ensures that the WKI DS is independent from any of the underlying ontologies, and can be easily extended to use new sets of them. Table 4 presents methods related to the WKI Common Model.

Method signature	Description
`String storeCommonObject(String modelUri, Class<? extends IWkiCommonObject> type, IWkiCommonObject commonObj)`	Adds a new common object to the given model and returns an automatically generated ID. Operation depends on the store strategy used in the WKI DS, which decides where the object must be saved. Every time a store operation is successfully performed, the exact timestamp of this operation is recorded with additional information about the ID and the type of the stored object.
`void updateCommonObject(String modelUri, Class<? extends IWkiCommonObject> type, IWkiCommonObject commonObj)`	Updates common object from a given model. Operation depends on the store strategy, which decides where objects are stored.
`IWkiCommonObject getCommonObject(String modelUri, Class<? extends IWkiCommonObject> type, String id)`	Returns a common object for a given type and ID from a specified model. The type must be one of the IWkiCommonObject types. Returns null if an object with the given type and ID does not exist. Throws exception if this model does not exist Operation depends on the retrieve strategy used in the WKI DS, which describes how to read the object from the internal storage units. The purpose of this method is to retrieve a previously stored object in the most efficient way.

Method signature	Description
`void removeCommonObject(String modelUri, Class<? extends IWkiCommonObject> type, String id)`	Removes the specified resource form the model. Furthermore, the method removes information about the store and update date, type and id of the given object. Operation depends on the remove strategy, which describes the steps to remove an object from the internal storage units. The purpose of this method is to remove unused objects from storage, since a great amount of unused objects kept in WKI DS could slow down the search and retrieval of other objects.

Table 4 WKI DS API – common objects methods

In the future this part of the API may be extended to satisfy emerging needs.

4.5.3. The triple part

The WKI DS API is not restricted only to operations on objects from the Common Model. There are methods enabling the storage of whole RDF graphs (sets of triples) and the retrieval of data from specified models (presented in Table 5) through a simple interface. This part was added to the WKI DS API upon request of the partners, however, it is strongly recommended to use methods from 'common object' part of API instead.

Method signature	Description
`void storeData(String modelUri, InputStream data, WkiDataFormat dataFormat) throws WkiDataStorageException`	Stores data (set of triples - graph) in the given model. The data is read from the given InputStream. The method supports the storage of a great amount of semantic data gathered by other services.
`void getData(String modelUri, OutputStream outStream, WkiDataFormat dataFormat)`	Writes all data from the given model to the given Output stream. The method supports the retrieval of

Method signature	Description
`throws WkiDataStorageException`	all data gathered in a model.

Table 5 WKI DS API – triple store methods

4.5.4. The file part

The WKI DS is responsible for storing all the data meant for use by the WKI System and its services; therefore, for its API to be complete, there is a need to provide an interface for storage and retrieval of any kind of files. To meet this requirement, the following API methods (Table 6) expose the required operations on file content.

Method signature	Description
`InputStream getFileContent(String contentUri)` `throws WkiDataStorageException`	Returns the content of the file with the given URI.
`String storeFileContent(InputStream content)` `throws WkiDataStorageException`	Stores the content of the file and returns its URI. A unique filename will be generated. The method should be used to store any kind of content that can not be saved using the storeData method.
`String storeFileContent(InputStream content, String contentID)` `throws WkiDataStorageException`	Stores the content of the file and returns its URI. ContentID should be unique and point to the content during access via HTTP. The content stored through this method can also be retrieved using the getFileContent method together with the returned URI. The method should be used to store any kind of content that should be accessible through HTTP protocol. During access via HTTP protocol a content ID must be used to identify the requested data uniquely.
`void updateFileContent(InputStream content, String contentID)` `throws WkiDataStorageException`	For some given content ID (used previously with different content), the method stores new content. The method should be used to override content pointed by the given content ID.

Table 6 WKI DS API – file management methods

4.5.5. The query part

The WKI DS needs to provide access to the data for a variety of services. Not all of them maintain, or have access to the necessary information needed to directly retrieve resources (based on URIs). Table 7 contains a list of methods which enable external services to perform customized queries for data gathered by the WKI DS.

It is expected that during the further development of the WKI use case applications, more "query-related" methods will be added to the WKI DS API.

Method signature	Description
`String executeQuery(String modelUri, String sparqlQuery)` `throws WkiDataStorageException`	Executes a SPARQL query on the triple store and given model. The result is formatted as "SPARQL result format in XML"[49].
`Iterator<IWkiCommonObject> getRecentCommonObjects(String modelUri, Class<? extends IWkiCommonObject> type, Date since)` `throws WkiDataStorageException`	Returns an iterator over common objects of the given type that have been added or updated since the given date. The purpose of this method is to enable the retrieval of all objects of a given type added or updated after given date. Every necessary piece of information is gathered during execution of the storeCommonObject method. Executing the removeCommonObject method removes the information about store and updates the date from the WKI DS. The object is no longer accessible via this method even if its store/update date and type meet the specified criteria.
`<T extends IWkiCommonObject> Collection<T> findCommonObjects(String modelUri, Class<T> type, T objectPattern)`	Returns a collection of all found common objects which fulfil all criteria passed through the given object

[49] as described in http://www.w3.org/TR/rdf-sparql-XMLres

`throws WkiDataStorageException;`	pattern. The method returns only those objects which have all fields equal to the corresponding non-null fields from the given object pattern. Only non-null fields from object pattern and their equivalents from stored objects are compared. An object pattern should have filled only those fields which should be used during search. The rest of the fields must have null values. The method is intended for performing simple search operations on data stored in the WKI DS without the need to create complex queries. The method implementation depends on the search strategy.

Table 7 WKI DS API – query methods

4.5.6. The web part

Upon request from WP5 the WKI DS component provides an API and a simple implementation allowing for the retrieval of file content from the WKI DS through the use of the HTTP protocol. This part of the WKI DS API is presented in Table 8. This functionality is intended for use with FOAF[50] profiles and is consistent with the FOAF project philosophy, which assumes

> *"[...] creating a Web of machine-readable pages describing people, the links between them and the things they create and do [...]"*[51]

Although it is intended for use with FOAF profiles, this part of the API is not restricted to them and allows access to every data stored (using the 'storeFileContent' method with content ID parameter) in the WKI DS as a web resource (data accessible via HTTP protocol).

Internally, the solution is based on Apache Camel[52] and its jetty component[53]. The jetty component provides HTTP-based endpoints for

[50] http://www.foaf-project.org/

[51] http://www.foaf-project.org/

[52] http://camel.apache.org/

[53] http://camel.apache.org/jetty.html, based on the Jetty Web Server (http://www.mortbay.org/jetty/)

consuming HTTP requests. That is, the jetty component behaves as a simple Web server. Requests received by jetty HTTP endpoint are routed to a message processor that parses them, retrieves requested data and returns it. The message processor is deployed inside the WKI System and has access to all its components including the WKI DS. The WKI System and the WKI DS will handle all requests with a valid URL (previously configured in the WKI System).

The currently available solution does not support any authentication and authorization method. Content accessible through HTTP protocol can be retrieved by everyone, although, jetty component provides SSL support and allows configuring a list of jetty handlers on the endpoint, which can be used to enable advanced jetty security features.

Method signature	Description
`http://host:port/path?resID=someID`	Retrieves data via HTTP protocol stored with content id 'someID'. The data is returned in the HTTP response. Only files stored with `storeFileContent(InputStream content, String contentID)` can be accessed with the HTTP request.

Table 8 WKI DS API – web access

In the future version of the WKI DS, it will be possible to use slash-based URLs instead of hash-based ones, which is more appropriate for RDF resources.

4.5.7. Code sample

In the following, some code samples are presented on the basis of a simple set of examples. They illustrate the ease of use of the WKI Data Storage API and the transparency of its internals to the client.

In all examples below, it is assumed that a reference to WKI Data Storage is reachable and stored in `wkiDataStorage` variable. Exception handling was omitted for the sake of simplicity.

Let us assume that a new user has appeared in the WKI System. It is necessary to store all his personal details in WKI Data Storage to be able to access them later.

Initialization

First, before performing any operation on common objects, the model, in which the data will be kept, needs to be created and initialized:

```
String userModelUri = ...;
```

```
wkiDataStorage.createModel(userModelUri);
wkiDataStorage.startModel(userModelUri);
```

Listing 1 Model initialization

The model needs to be created only once and after that it will be available at all times in the data storage module. Once the model is created, the client must start it, in order to inform the WKI DS that store/retrieve operations will be performed on the given model, and to let the WKI DS perform internal initialization.

Store/retrieve operations

After the initialization is complete, clients are allowed to execute operations on common objects.

```
User user = ...; // user object created
user.setXYZ(...) ; // all required fields filled
...
String userID = wkiDataStorage.storeCommonObject(
  userModelUri, User.class, user);
```

Listing 2 Storing of common object

As presented on Listing 2, to store common object in the WKI DS, only one line of code is actually required. Depending on the store strategy used by the WKI DS, user object will be saved in object storage, triple store, or in both; however, this is completely invisible to the clients.

When the service needs information about a previously added user, the user's personal details have to be retrieved in the following way:

```
User userObj = (User) wkiDataStorage.getCommonObject(
  userModelUri, User.class, userID);
```

Listing 3 Retrieval of common object

Retrieval of the object internally in the WKI Data Storage could be optimized by using a properly constructed retrieve strategy. Listing 3 shows that retrieving objects is as simple as adding them to WKI Data Storage.

Finalization

If all work relating data from the given model is done, the model should be stopped:

```
wkiDataStorage.stopModel(userModelUri);
```

Listing 4 Model stop

Operations on file

After a user's personal details are saved, one of the WP5 services wants to store a file with the user's FOAF profile and make it accessible through the HTTP protocol.

To accomplish this task, the following code snippet is required:

```
File foafProfile = ...;
String foafProfileUri = wkiDataStorage.storeFileContent(new
FileInputStream(foafProfile), "john_doe");
```

Listing 5 Store FOAF file

Now the content of the FOAF profile file is available under the `http://host:port/weknowit?resID=john_doe` address.

In case the file content should not be accessed via the HTTP protocol, the last line of Listing 5 has to be substituted for the following one:

```
String foafProfileUri = wkiDataStorage.storeFileContent(new
FileInputStream(foafProfile));
```

Listing 6 Store file content

Furthermore, the content of the FOAF profile file can be also retrieved via a 'normal' method of WKI Data Storage:

```
InputStream content = wkiDataStorage.getFileContent(
  foafProfileUri);
```

Listing 7 Retrieve file content

Simple searching

When a user logs into the WKI System, information about the latest events should appear on her screen. The code snippet presented below shows how to obtain all events added or updated in the last 24 hours.

```
String eventModelUri = ...;
wkiDataStorage.startModel(eventModelUri);
Calendar c = Calendar.getInstance();
c.add(Calendar.DAY_OF_MONTH, -1);
Date yesterday = c.getTime();
Iterator<IWkiCommonObject> it = getRecentCommonObjects(
```

```
  eventModelUri, Event.class, yesterday);
// here code performing operations on returned iterator it
wkiDataStorage.stopModel(userModelUri);
```

Listing 8 Search for recently added/updated objects

Conclusion

All presented examples show that common tasks related to storing/retrieving data require only few lines of code, and all internal mechanisms are hidden from the clients.

5. Integration of the UI components

The D6.4.1 prototype presents integration of the WKI System services. However, having in mind the requirements of the use case applications, it is crucial to ensure that UI layer components can be also integrated within the WKI System.

Such integration is already planned (in close WP6-WP7 cooperation), and partially implemented. Even though the full description of use case applications, and their cooperation with the WKI System is outside the scope of this deliverable, some information on this subject should be given, in order to present more comprehensive picture of the integration issues[54]. Also, it is important to stress, that the integration process of the UI components was already initiated.

5.1. UI components integration - architecture

Figure 11 reflects the current state of the architecture. This is a layered architecture, with strictly defined communication paths, which means that the communication is allowed only between the direct neighbours.

The colours of components on Figure 11 mark the division between the use-case applications (green colour), and the WKI System (blue colour). Yellow vertical arrows represent messages exchanged between the WKI System and the UI components.

[54] The full picture of use case applications and UI components integrations will be presented in WP7 deliverables.

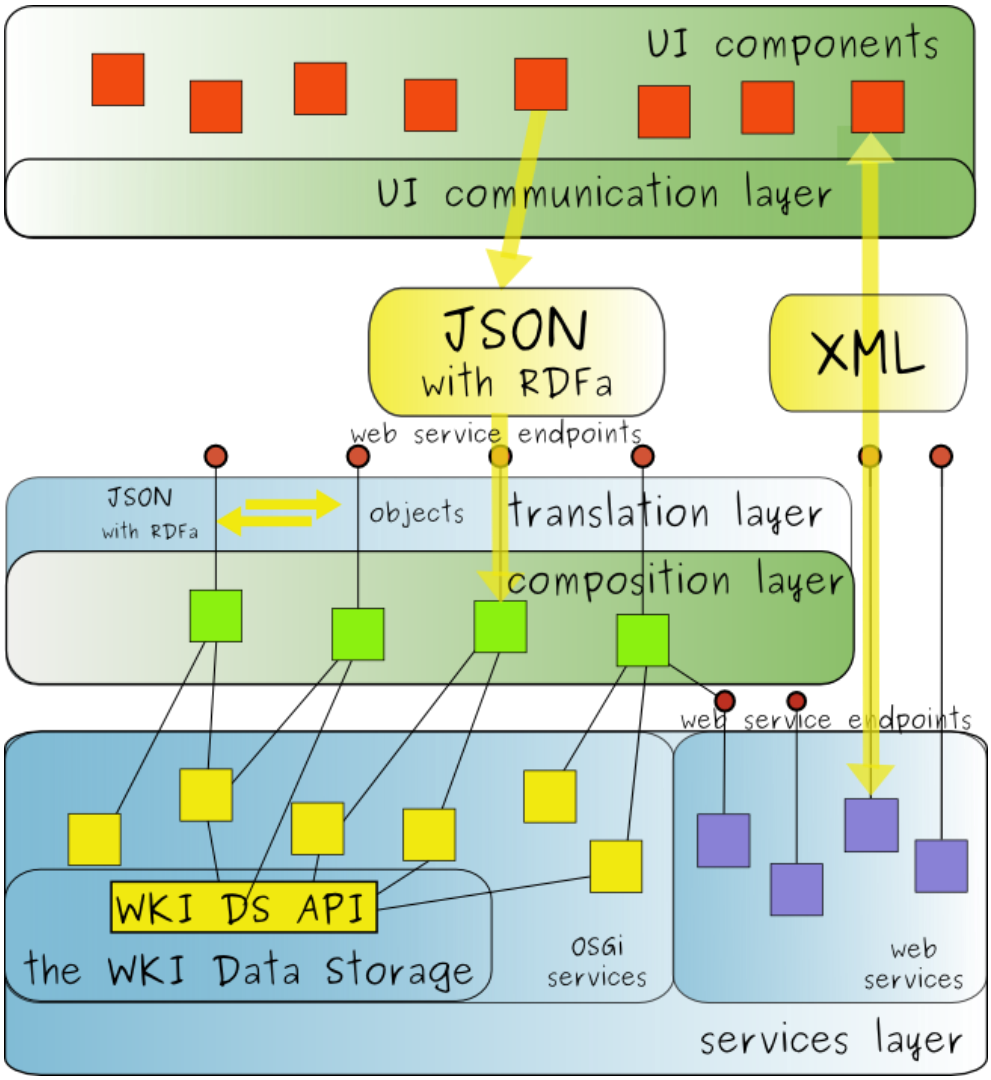

Figure 11 UI components integration - an overview

5.1.1. The use-case applications

The use case applications contains:

- UI components (represented as red rectangles on Figure 11),
- UI communication layer,
- the composition layer.

It is worth noticing, that UI components and UI communication layer live outside the OSGi environment provided by the WKI System, while the composition layer lives in this environment.

UI components

These components are responsible for presentation of data. They can also include some (hopefully thin) business logic (even thought the real logic -

especially combining of data from many services - should be handled by the composition layer).

UI communication layer

This layer is responsible for making calls and receiving responses from the layers underneath (i.e. composition layer).

Composition layer

Even though this layer lives in the OSGi environment of the WKI System, it is strictly related to the particular use-case.

The idea behind the compositions layer is as follows. It is important that the UI components receive only data that are really required (because of the bandwidth/performance reasons). Also, it is important to keep the UI layer as thin as possible (in terms of business logic performed by this layer).

The composition layer fulfils both requirements. It gathers required data from various services and provide it to the application in the desired form. Thus, UI components are not required to do the parsing and combining of data provided by various services, and the size of messages passed via the HTTP protocol is cut to minimum. The composition layer provides methods, that exactly answers the requirements of the UI components, so the data returned by the composition layer fits perfectly their needs.

Some UI components does not make use of the composition layer and prefer to obtain data directly from the services.

5.1.2. Messages

Messages between the WKI System and the UI components are exchanged via the HTTP protocol.

UI components of the ER use case rely on the messages in RDFa/JSON[55] format. UI components of the CSG use case use different types of messages. Some services of CSG use case return XML, which is consumed directly, without any transformation, by the UI components.

The translation layer (part of the WKI System) is responsible for the translation of messages format.

5.1.3. The WKI System

The WKI System consists of:

- WS endpoints (represented as red dots on Figure 11),

[55] http://json.org/

- translation layer,
- services layer.

WS endpoints

WS endpoints can expose compositions of services as well as single services. The first options is preferred, as it takes the burden of combining outputs derived from many services from the UI layer.

The translation layer

The translation layer is responsible for translating messages between formats used by the WKI System and by the UI components of ER use case. This layer is capable of translation of Java POJOs to JSON + RDFa (and vice versa). This layer is still under development and it is not yet fully decided, if it is possible to be completely use case agnostic, or if it has to be somehow configured to fit the requirements of a specific use case. On Figure 11 this layer is placed within the WKI System, as ideally it should be independent of the use case application.

The WKI Services

The services (yellow and violet squares) on the bottom of Figure 11 represent the low-level, atomic services provided by WPs, as described in section 2. Some of the services live in the OSGi environment, while other are accessible via Web Services and are outside the OSGi environment of the WKI System (as described in section 2.1).

Among the services, one is singled out - it is the WKI DS service. Technically, it is just one of many services, but it plays a central role in the system. Please notice, that the use-case application can contact with the WKI Data Storage only by using other services - the interface of the WKI Data Storage is not exposed via Web Services.

5.2. Interaction of the UI components with the WKI System

As explained in the previous section, the use case applications will communicate with the WKI System via WS endpoints. They will rarely talk with the single WKI Service, but rather with the composition layer. The elements of the composition layer:

- will be accessible via WS endpoints (so they are accessible via the WS calls),

- will gather data from one or more of services of the WKI System; will repack/transform/combine received data, and back to the use case application.

The composition layer will contain elements some elements that are use case agnostic (common for many use case applications), but the majority of this layer will be created based on the specific requirements of the particular use case application.

6. The prototype

The goal of this prototype is to demonstrate the integration of the WKI services provided by partners. This prototype is a step forward from the bare architecture prototype presented in deliverable D6.2.1 "Definition and implementation of APIs version 1" towards the full integration (including use-cases) that will follow in WP7 deliverables of M19. A very important step accomplished in this deliverable was to set a "common language" for services to communicate – this was made possible by the introduction of the Common Model (see Section 3).

6.1. Installation

The installation of this prototype is very similar to the installation of the WKI System presented in the "How to install the WKI System" guideline (see Appendix B.). For the convenience of the prototype users, the number of installation steps was reduced to a minimum (e.g. by using in-memory RDBMS and native file system storage which does not require any installation).

The WKI System prototype **requires Linux as the operating system** and about 200MB of free disk space. More detailed requirements can be found in guideline "How to install the WKI System" (see Appendix B.).

The prototype of the WKI System is distributed in the form of a tar archive, namely `wp6_d641_prototype.tar.gz`. Below, we present in brief the series of installation steps that are necessary in order to test it.

6.1.1. Unpack the system

Open the shell and unpack the file `wp6_d641_prototype.tar.gz`:

```
tar xzf wp6_d641_prototype.tar.gz
```

A `wki_prototype` folder will be created in the current directory. It contains a set of subfolders:

```
|-- wki_prototype
    |-- LICENSE.txt
    |-- NOTICE.txt
    |-- README.txt
    |-- RELEASE-NOTES.txt
    |-- ant
    |-- bin
```

```
    |-- configs
    |-- deploy
    |-- etc
    |-- fs_cr_repo
    |-- lib
    |-- licenses
    |-- scenario
    |-- sql
    |-- system
    |-- wki-readme.txt
    `-- wki-release-notes.txt
```

Two new subfolders (in comparison to the WKI System) are available:

- `scenario` – it contains bundle that runs the scenarios when deployed to the `deploy` folder,
- `sql` – contains an installation version of the PostgreSQL database.

Also, much more bundles are available in the `deploy` folder.

6.1.2. Set WKI_HOME

`WKI_HOME` variable must be set (please consult documentation of your linux shell – the syntax might be different on your distribution):

```
cd wki_prototype
export WKI_HOME=`pwd`
```

6.1.3. PostgreSQL installation

Installation of the PostgreSQL database is required by some services. Please follow the installation instruction included in Appendix B. (both WP2 and WP4). The only difference is, that the installation version of PostgreSQL does not need to be downloaded as it is available in the `sql` subfolder of the `WKI_HOME`.

6.2. Running of the WKI System

Proceed to the `WKI_HOME/bin` directory and run the WKI System:

```
chmod 755 wki-start.sh
```

```
chmod 755 servicemix
./wki-start.sh
```

After the system is up and running, the status of all installed OSGi bundles can be checked using the `list` command:

```
smx@root>osgi
smx@root:osgi>list
```

Please note that it may take a few minutes to install all bundles.

6.2.1. Reading logs

In order to monitor the progress of the scenarios open the second shell and watch the logs:

```
tail –f $WKI_HOME/data/log/servicemix.log
```

6.2.2. Execution of scenarios

To execute the scenarios (described in section 6.3) copy the jar file from `WKI_HOME/scenario` to `WKI_HOME/deploy`:

```
cp $WKI_HOME/scenario/*.jar $WKI_HOME/deploy
```

The scenarios will be executed automatically. The output, which describes the progress of the scenarios will be printed to the `WKI_HOME/data/log/servicemix.log` file. Example outcome is presented below[56]:

```
11:43:19,775 | INFO  | DocumentProcessing                    |
[d641: document scenario] Found 4 document(s):
11:43:19,775 | INFO  | DocumentProcessing                    |
[d641: document scenario]    *  PROTOTYPE  DOCUMENT  -  id  =
[1256287475466]
11:43:19,775 | INFO  | DocumentProcessing                    |
[d641: document scenario]    *  PROTOTYPE  DOCUMENT  -  id  =
[1256287475469]
11:43:19,775 | INFO  | DocumentProcessing                    |
[d641: document scenario]    *  PROTOTYPE  DOCUMENT  -  id  =
[1256287475477]
```

[56] scenario logs can be filtered out by issuing `less servicemix.log | grep d641` command

```
11:43:19,775 | INFO  | DocumentProcessing                    |
[d641: document scenario]    * PROTOTYPE DOCUMENT - id =
[1256287475480]

11:43:19,775 | INFO  | DocumentScenario                      |
[d641: document scenario] DOCUMENT SCENARIO finished
SUCCESSFFULY.

11:43:19,775 | INFO  | VisualScenario                        |
[d641: visual scenario] Running VISUAL SCENARIO...

11:43:19,775 | INFO  | VisualScenario                        |
[d641:       visual       scenario]       Storing       image
[http://farm1.static.flickr.com/197/481818086_62f0d8495d.jpg]
in WKI Data Storage...
```

6.2.3. Rerunning of the WKI System and the scenarios

In order to rerun the scenarios, please do the following:

- shutdown the WKI System (see section B.5)
- remove the WKI_HOME folder
- repeat the installation procedure as described in section 6.1

These steps are required due to the fact, that the prototype uses some in-memory databases.

6.3. Scenarios

In the D6.4.1 prototype some scenarios presenting the integration of the WKI services are implemented. The main idea of the WKI project is to show collective intelligence. These scenarios focus on ensuring that from the technical point of view, the WKI System architecture and the integration techniques are capable of combining the WKI services in such a way that the **collective intelligence** can be achieved. The implemented integration scenarios show that the services can communicate and exchange meaningful information with the WKI DS and with each other. Also, the role of objects from the Common Model is presented.

The GUI layer will be implemented in further tasks (in WP7), so no visual output, except the information printed to log files, is yet available.

In the scenarios services from different WPs are used. They are listed in Table 9:

Service name	**WP**
WP1_AccountManager	WP1
WP1_Comment	WP1

Service name	WP
WP1_Rate	WP1
WP1_Tag	WP1
WP2_TagProcessing	WP2
WP2_VisualAnalysis	WP2
WP3_LocalTagCommunityDetector	WP3
WP4_CommunityDesignLanguage	WP4
WP6_DataStorage	WP6

Table 9 WKI Services used in prototype scenarios

In each scenario different services are used. As presented on Figure 12 only the WP6_DataStorage service appears in every scenario.

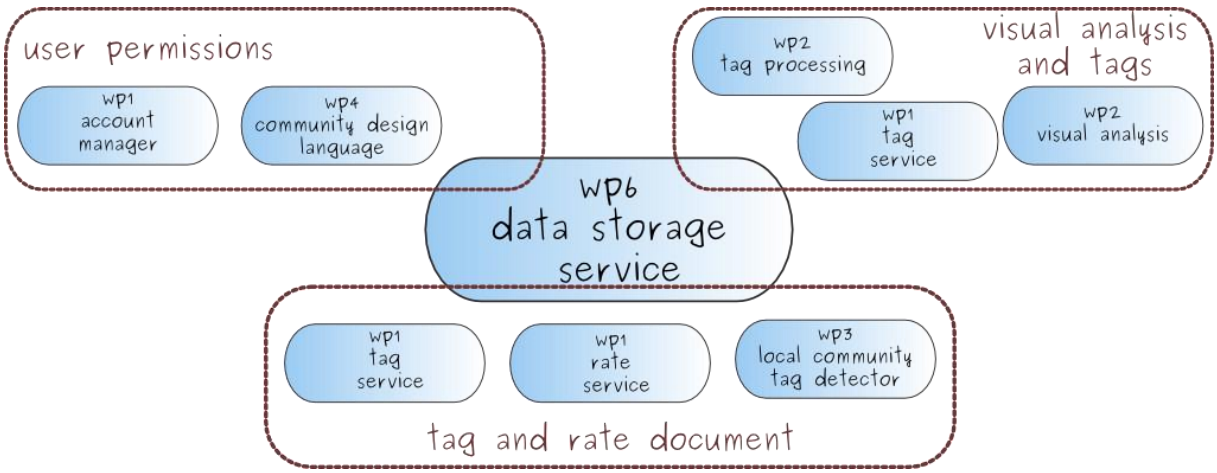

Figure 12 WKI Services used in prototype scenarios

In each scenario a **client service** appears. This service was created for the prototype and its role is to execute methods of selected WPs services in order to present possible integration paths that involve them. This service is OSGi-based. After deployment to the WKI System it obtains references to other WKI services via the Spring dependency injection mechanism[57] and executes their method. One can think of a **client service** as an application that makes use of the WKI System and its services.

The client service used in the scenarios is not the only way for achieving integration among services in the WKI System. Another possibility is to use Apache Camel and bind services together using a more declarative approach to achieve a similar effect.

[57] http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/ch01.html#overview-dependency-injection

6.3.1. Tag and rate document

This scenario presents a typical usage of the WKI System – a document is created, and, later on, it is enhanced with some metadata (tags and ratings). Some metadata is provided by the client of the WKI System, while some is automatically derived by a specialized service. From the technical point of view, this scenario shows:

- integration of services (provided by WP1, WP3 and WP6),
- how the Common Model facilitates communication between services,
- initialization of services with data (for instance, `WP3_LocalTagCommunityDetection` requires some data to respond to requests issued by other services),
- cooperation between services and the WKI DS,
- some features of the WKI DS
    - ability to store common objects,
    - ability to retrieve common objects by ID,
    - query functionalities – retrieval of "similar" objects and recently added objects of the given type.

Figure 13 presents the sequence diagram of this scenario[58]. First, the client service creates a document and stores it. After this, the document is retrieved by ID from the WKI DS (only to present the retrieving functionality of the WKI DS). Then, the document is passed to `WP1_Tag` and `WP1_Rate` which add tags and ratings to it respectively (saving it back to the WKI DS afterwards).

After some tags have been added to the document, the `WP3_LocalTagCommunityDetector` service is asked for a list of related tags (to the ones just added). This service responds with a list of related tags. These tags are also added to the document by the `WP1_Tag` service.

In order to show some more capabilities of the WKI DS, the client service first adds few more documents to the WKI DS, and then asks the WKI DS for:

- recently added documents,
- similar documents (with the same title).

In both cases, a list of documents is returned to the client.

[58] Green arrows represent tasks, that are not really required to implement the scenario, and were added in order to present various functions of the WKI DS.

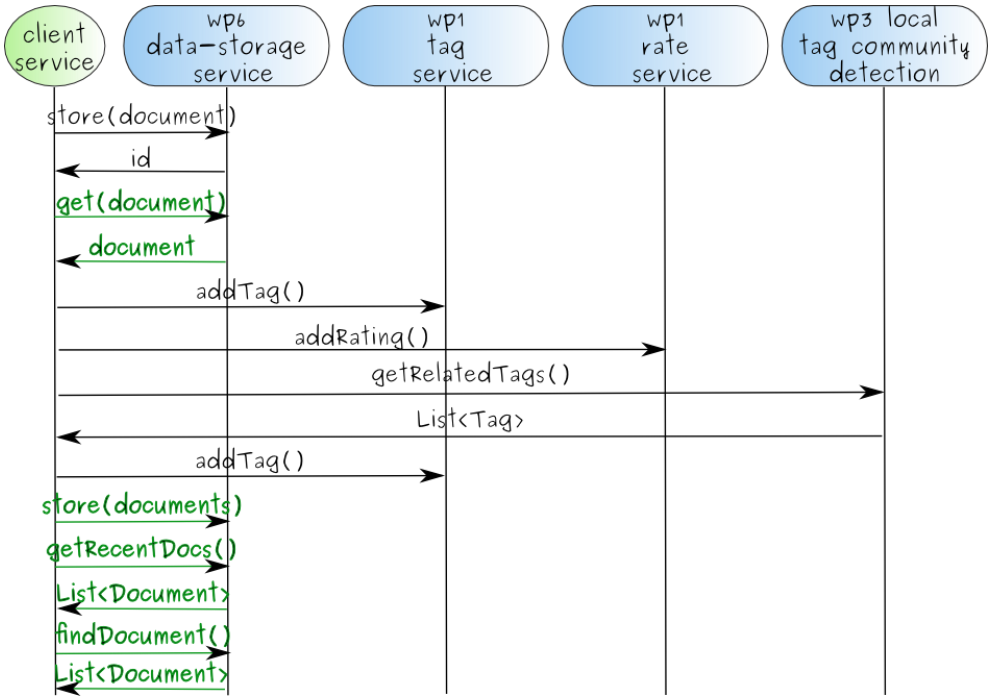

Figure 13 Tag and rate document scenario – sequence diagram

6.3.2. User permissions

The aim of this scenario (depicted on Figure 14) is to present how a permission system can be used to control access to the resources of the WKI System.

First, the client service creates a new user U (using `WP1_AccountManager`) and two documents (A and B). All three objects are stored in the WKI DS. The client service informs the CAP (`WP4_CommunityDesignLanguage`) about the existence of user U and documents A, B, and creates one group for each of them – user U belongs to group G, document A to group D1 and document B to group D2. Now, a set of permissions is created that allows group G to access resources from group D1. Finally, CAP is asked to decide if user U can access document A and document B. In the first case (document A), the answer is "yes", in the second case (document B) CAP answers "no".

This scenario presents:

- integration of new services (provided by WP1 and WP4),
- more objects from the Common Model (`User` class)
- usage of the permission system.

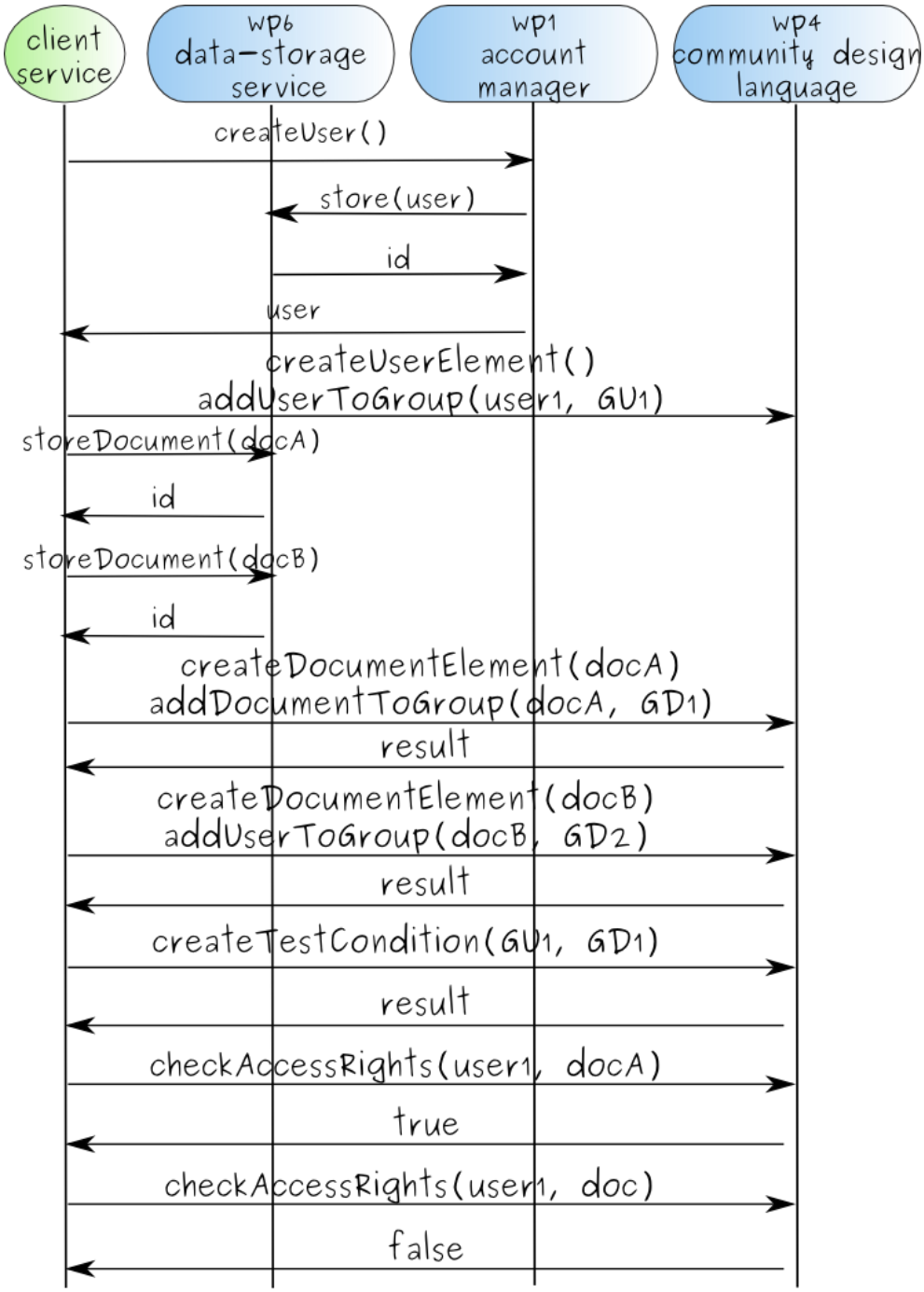

Figure 14 User permissions scenario – sequence diagram

6.3.3. Visual analysis and tags

This scenario presents how multimedia and metadata can be combined in the WKI DS. The interaction of services is presented on Figure 15.

The client uploads an image to the WKI System. The image is saved in the WKI DS. Then, a document (one of the objects of the Common Model) is created which serves as a wrapper around the uploaded image. The

document is also stored in the WKI DS. Now, the `WP2_VisualAnalysis` analyzes the image and returns a data structure (of `ViralType` class) which contains:

- geo location,
- list of tags (String),
- list of images URIs (String).

These metadata are then added to the document, which is again stored (updated) in the WKI DS.

The geo location and list of tags are passed to the `WP2_TagProcessing` service which returns a list of tags. `WP1_Tag` service adds these tags to the document and stores it in the WKI DS.

This scenario presents:

- integration of new services (provided by WP2 and WP3),
- utilization of multimedia files in the WKI System,
- combination of multimedia files and metadata.

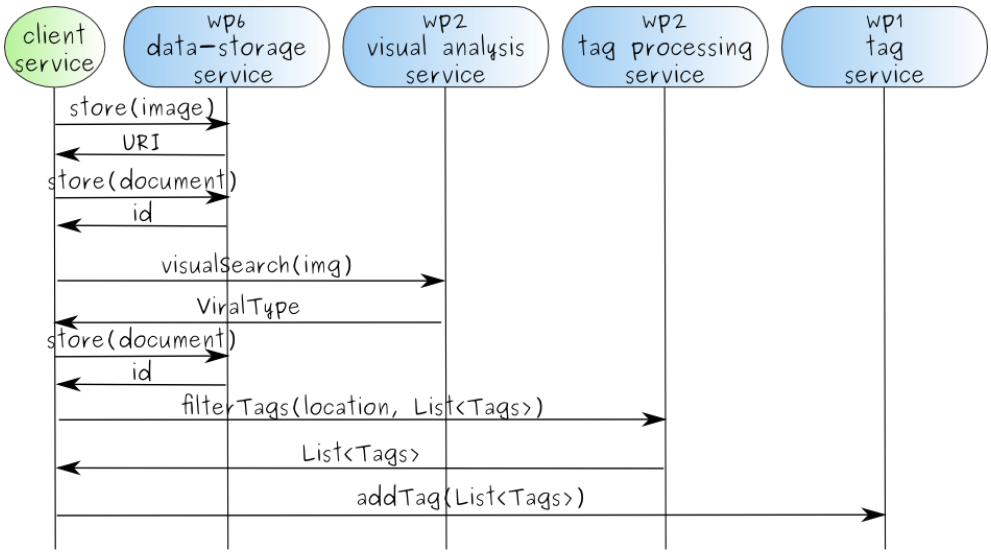

Figure 15 Visual analysis and tags scenario – sequence diagram

6.4. Further work (towards M19 deliverables)

The integration of services presented in this prototype will be used as a base for the further development of the 1st prototype (M19 deliverables of WP7). As all the technical means are in place, the emphasis will be placed on the cooperation of services that exposes the collective intelligence of the WKI System.

The composition layer – described in section 5.1.1 – will be extended, so it fulfils all the requirements of use case applications. The services of the composition layer will:

- pass the request received from the UI to appropriate services of the service layer,
- gather the results, combine them and wrap them in POJOs,
- return the results to the UI layer.

The composition layer will be developed based on the experiences gained during the development of D6.4.1. It is planned to reuse some code from the scenarios presented in this prototype.

Actions planned for the further development towards the M19 deliverables also include:

- usage of real databases instead of in-memory substitutes,
- usage of Hadoop DFS storage for storing of large multimedia files,
- evolution of the WKI DS API
    - new query methods,
    - polishing of the API (e.g. removal of `type` parameters).
- refinement of the Common Model (driven by use-case and by D5.3.1 deliverable),
- usage of Apache Camel to describe the interaction between the WKI services.

7. Conclusions

Deliverable D6.4.1 presents the **first integration of the WKI services**. It shows how OSGi bundles, the Common Model and the WKI DS together constitute an integrated environment that enables and facilitates the cooperation among services. Integration with the UI components of use case applications is also discussed.

D6.4.1 demonstrates, that the technical measures used in the WKI System are capable of achieving the goal of collective intelligence by combining the results of work of different services provided by WP1-WP5.

D6.4.1 supports the adequacy of the development infrastructure set up and maintained by WP6 for the needs of the project.

Further tasks for WP6 (and also WP7) will be build upon the work carried out for this deliverable.

8. References

[1] Humberto Cervantes, Jean-Marie Favre, "Comparing JavaBeans and OSGi Towards an Integration of Two Complementary Component Models," euromicro, pp.17, 28 th Euromicro Conference (EUROMICRO'02), 2002

[2] U. Radetzki, G. Witterstein, A. B. Cremers, "An Integration Platform for Heterogeneous Services in Life Science Applications", 2002.

[3] Frank Jennings, Matjaz Juric, Poornachandra Sarang, Ramesh, Loganathan, "SOA Approach to Integration: XML, Web services, ESB, and BPEL in real-world SOA projects", Packt Publishing, 2007.

[4] Gregor Hohpe, Bobby Woolf, "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions", Addison-Wesley Professional, 2003.

[5] Binildas A. Christudas, "Service Oriented Java Business Integration", Packt Publishing, 2008

[6] Osgi Alliance, "OSGi Service Platform: The OSGi Alliance", 2003.

[7] A. Chaudri, M. Loomis, "Object Databases in Practice", Prentice-Hall, 1997.

[8] C. S. R. Prabhu, "Object Oriented Database Systems", Prentice-Hall of India, 2004.

[9] Leonard Richardson, Sam Ruby, "RESTful Web Services", O'Reilly Media, 2007.

[10] Bill Burke, "RESTful Java with JAX-RS", O'Reilly Media, 2009.

[11] Martin Kalin, "Java Web Services: Up and Running", O'Reilly Media, 2009.

[12] Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu, "Professional Java Development with the Spring Framework", Wrox, 1 edition, 2005.

[13] T. Segaran, J Taylor, Colin Evans: Programming the Semantic Web, O'Reilly Media, 2008

[14] S. Powers: Practical RDF, O'Reilly Media, 2003

[15] Pieter Van Zyl, Derrick G. Kourie, Andrew Boake, "Comparing the Performance of Object Databases and ORM tools", SAICSIT; Vol. 204, 2006

[16] David Taniar, Johanna Wenny Rahayu, "Web Semantics Ontology", 2006.

[17] Sharman Raj, Kishore Rajiv, Ramesh Ram, “Ontologies”, 2007.

[18] Grigoris Antoniou, Frank van Harmelen, “A Semantic Web Primer”, 2004.

[19] John Davies, Rudi Studer, Paul Warren, “Semantic Web Technologies : Trends and Research in Ontology-based Systems ”, 2006.

[20] A. R. J. Francois, R. Nevatia, J. Hobbs, and R. C. Bolles. VERL: An ontology framework for representing and annotating video events. MultiMedia, 12(4), 2005.

A. Attached files

`wp6_d641_prototype.tar.gz` – contains the D6.4.1 prototype. Its content is described in Section 6.1.

B. Installation of the WKI System

B.1. Introduction

B.1.1 Rationale

The WKI system must be installed on a local machine, so that developers are able to verify whether their services are compatible with the WKI architecture.

B.1.2 Prerequisites

- Java & Maven installed and configured - as it is described in 'How to install Java & Maven' guideline.
- Working network connection.

B.1.3 Outcome

The WKI System installed.

B.2. Installation of the WKI System

Download latest version of the WKI system installer - the download link can be found on the WP6 wiki page.

Choose version appropriate for your operating system:

1. `wki-X.Y.zip` for windows
2. `wki-X.Y.tar.gz` for linux

Uncompress the downloaded file. A `wki-X.Y` folder will be created. Set system property `WKI_HOME` pointing to this folder.

In case of Windows, please DON'T use WinRAR – some problems were reported while unpacking the wki zip file.

B.2.1 Files and folders in the WKI System

This is how `WKI_HOME` looks like:

```
|-- bin
|-- configs
|-- data
|-- demos
|-- deploy
```

```
|-- etc
|-- examples
|-- lib
|-- licenses
|-- system
|-- LICENSE.txt
|-- NOTICE.txt
|-- README.txt
|-- RELEASE-NOTES.txt
|-- wki-readme.txt
|-- wki-release-notes.txt
```

The most important files and directories are:

- `bin` - startup/shutdown scripts
- `configs` – in this directory will be copied bundle's properties files for each WPs in its sub-directory

```
|-- configs
     |-- wp1
     |-- wp2
     |-- wp3
     |-- wp4
     |-- wp5
     |-- wp6
     |-- wp7
```

- `data` - all runtime data (including deployed bundles and log files) can be found here
- `deploy` - every bundle copied to this directory will be deployed to the WKI system
- `wki-release-notes.txt` - information about this particular release

B.2.2 Log level

By default the WKI System logs only information about errors. In case you need to change the log level printed by the WKI System, edit `config.properties` file in `etc` directory:

```
|-- etc
    |-- config.properties
```

Open this file with text editor and find line:

```
#felix.log.level=4
```

Now, you can uncomment it (by removing '#' sing) and set one of the values presented in Table 10.

Value	Log level
0	NO LOG
1 (default)	Error
2	Warning
3	Information
4	Debug

Table 10 Log levels

For example to set log level to “Information” the line should look like:

```
felix.log.level=3
```

B.3. PostgreSQL installation

Some services require an additional relational database to store various data that they use during computations. PostgreSQL must be installed to satisfy this need.

- Download PostgreSQL package appropriate for your operating system from `https://cr.weknowit.eu/wp4/postgres-install/`
    - Services were tested to work with this particular version of PostgreSQL that you find on WebDavs, so if you use it we guarantee that they will cooperate smoothly. It is very possible, that they also work properly with any other (especially newer) version, but it was not tested, so we encourage you to use exactly this version.

- Install PostgreSQL according to instructions described on `http://www.postgresql.org` page using packages downloaded in previous step.
- if you install PostgreSQL on a remote machine (not the same as the WKI System) remember to configure the PostgreSQL server correctly. Please refer to:
    - http://www.postgresql.org/docs/8.3/static/config-setting.html
    - http://www.postgresql.org/docs/8.3/static/runtime-config-connection.html
    - http://www.postgresql.org/docs/8.3/static/client-authentication.html
- remember to allow access to your PostgreSQL installation by including appropriate information in the `pg_hba.conf` file (please refer to documentation to find out more about this file)

In the following sections names of databases, users and passwords are given. If such need arises, they can be changed. In that case, appropriate properties files, of services that connect with these databases, have to be changed. Please refer to documentation of each service to learn where such files can be found, and how to modify them.

B.3.1 WP2 services

For some WP2 services to work correctly, some database steps should be performed:

- create database role with name 'viral'
- create database 'geodb'
- run two script from `WKI_HOME/configs/wp2` dir (as viral role):
    - `schema.sql`
    - `data.sql`

Example of creating role and running script using 'psql' command tool:

```
// connect to local postgresql server as a superuser
# psql -U postgres postgres

// create role
=> CREATE USER viral PASSWORD 'viral-client';
// create database
=> CREATE DATABASE geodb;
=> \q
```

```
// login as viral to geodb
# psql -U viral geodb

// run sql script
=> \i  WKI_HOME/configs/wp2/schema.sql
=> \i  WKI_HOME/configs/wp2/data.sql
\q
```

B.3.2 WP4 services

Services delivered by WP4 require PostgreSQL database installed. Follow presented steps to install and configure PostgreSQL correctly:

- create database role with name 'wp4' and password 'wp4_password', then grant login and createdb privileges to the role (see the listing below),
- run two script from `WKI_HOME/configs/wp4` dir (as wp4 role):
  - `wki-structure.postgres.sql`
  - `wki-data.postgres.sql`

Example of creating role and running script using 'psql' command tool:

```
// connect to local postgresql server as a superuser
# psql -U postgres postgres

// create role and grant privileges to login and create databases
=> CREATE USER wp4 PASSWORD 'wp4_password';
=> ALTER ROLE wp4 WITH CREATEDB;
=> \q

// login as wp4 role
# psql -U wp4 postgres

// run sql script
=> \i  WKI_HOME/configs/wp4/wki-structure.postgres.sql
=> \q
```

B.4. Starting the WKI System

Use command line. Go to `WKI_HOME/bin` directory. Now depending on your operating system type:

Linux:

First make the startup script executable (you have to do this only once):

```
chmod 755 wki-start.sh
chmod 755 servicemix
```

then run the startup script

```
./wki-start.sh
```

Windows:

```
wki-start.bat
```

Some start-up information will be printed and if everything goes fine, the ServiceMix console will start:

```
 ____                   _           __  __ _
/ ___|  ___ _ ____    _(_) ___ ___|  \/  (_)_  __
\___ \ / _ \ '__\ \ / / |/ __/ _ \ |\/| | \ \/ /
 ___) |  __/ |   \ V /| | (_|  __/ |  | | |>  <
|____/ \___|_|    \_/ |_|\___\___|_|  |_|_/_/\_\

 ServiceMix Kernel (1.1.0.0-fuse)
 ServiceMix (4.1.0.0-fuse)

Type 'help' for more information.

smx@root:/>
```

ServiceMix console contains a decent help system, so please refer to it if you need more information.

B.5. Stopping the WKI System

To exit and to stop WKI system type the following command in ServiceMix console:

```
exit
```

You will go back to your system shell. Please note that if you have entered some 'sub-consoles' (for example by typing `osgi`) – you will need to enter `exit` command twice because the first `exit` will take you back to the root console of WKI system.

B.6. Cleaning of the WKI System

If you develop new services, you will likely need to remove the older versions before deploying new ones to the WKI system. In order to do this, some manual cleaning will be needed.
IMPORTANT - first stop the WKI system before performing these steps !

To clean the WKI system completely (to bring it to the initial state):

- delete the whole `data` directory,
- if the zero-length file named `lock` exists, delete it,
- remove any jars you have put in `deploy` directory.

B.7. The WKI DS components configuration

B.7.1 Knowledge Base

The knowledge base configuration file is located in WP6 WKI System configuration folder:

```
${WKI_HOME}/configs/wp6/koda-db.properties
```

and contains the database connection parameters.

By default an in-memory HSQL database model is used.

B.7.2 Object Storage

For the time being, only one object storage is available (koda-fscr) to store files in the local file system. This component configuration file is located in:

```
${WKI_HOME}/configs/wp6/koda-fscr.properties
```

and contains information about the folder where the files are stored.

B.8. Troubleshooting

B.8.1 Address already in use

When ServiceMix starts the port 1099 must be available for RMI registry. If it is unavailable, e.g. another instance of ServiceMix is already running or any other process uses it, the error `java.net.BindException: Address already in use occurs.`

To resolve this problem port 1099 must be freed up before starting a ServiceMix instance properly.

If it is impossible to free up this port, the ServiceMix RMI port number must be changed in the `etc/org.apache.servicemix.management.properties` file by changing the `rmiRegistryPort` and `serviceUrl` property to a port number that is available:

```
rmiRegistryPort = port
jmxLogin        = smx
jmxPassword     = smx
serviceUrl                                                    =
service:jmx:rmi:///jndi/rmi://localhost:port/jmxrmi
daemon          = true
threaded        = true
objectName      = connector:name=rmi
```

B.9. Links

The latest version of the WKI system:

- **http://mklab.iti.gr/weknowit/index.php/T6.2#Download**

Further information can be found here:

- **http://fusesource.com/docs/esb/4.0/getting_started/index.html**

Information about log levels:

- **http://fusesource.com/docs/esb/4.0/runtime/DeployESBLogConfig.html**

C. OSGi services

In order to use any OSGi-based service two pieces information are required:

- dependency data (groupId, artifactId and version) – which makes development possible (by informing Maven what library should be added to the classpath),
- Spring configuration data – used at runtime by the Fuse ESB framework to inject proper dependencies.

To facilitate the usage of services by partners, a wiki page was created that includes this important information regarding each OSGi-based service.

Service name	**Service interface(s) and dependency**
WP1_Tag	eu.weknowit.personal.intelligence.tags.ITagManager <dependency> <groupId>eu.weknowit.personal</groupId> <artifactId>wp1-user-services</artifactId> <version>1.0-SNAPSHOT</version> </dependency>
WP1_Rate	eu.weknowit.personal.intelligence.rating.IRateService <dependency> <groupId>eu.weknowit.personal</groupId> <artifactId>wp1-user-services</artifactId> <version>1.0-SNAPSHOT</version> </dependency>
WP1_AccountManager	eu.weknowit.personal.intelligence.account.IAccountManager <dependency> <groupId>eu.weknowit.personal</groupId> <artifactId>wp1-user-services</artifactId> <version>1.0-SNAPSHOT</version> </dependency>

Service name	Service interface(s) and dependency
WP1_Comment	eu.weknowit.personal.intelligence.comments.ICommentService <dependency> <groupId>eu.weknowit.personal</groupId> <artifactId>wp1-user-services</artifactId> <version>1.0-SNAPSHOT</version> </dependency>
WP1_LogIn	eu.weknowit.personal.intelligence.login.ILoginService <dependency> <groupId>eu.weknowit.personal</groupId> <artifactId>wp1-user-services</artifactId> <version>1.0-SNAPSHOT</version> </dependency>
WP1_ManageItem	eu.weknowit.personal.intelligence.items.IItemManager <dependency> <groupId>eu.weknowit.personal</groupId> <artifactId>wp1-user-services</artifactId> <version>1.0-SNAPSHOT</version> </dependency>
WP2_VisualAnalysis	eu.weknowit.media.visualanalysis.IVisualSearchService <dependency> <groupId>eu.weknowit.media</groupId> <artifactId>wp2-visual-analysis</artifactId> <version>1.0-SNAPSHOT</version> </dependency>
WP2_TagNormalization	eu.weknowit.tag.osgi_wki_tagnormalizationservice.osgi.ITagNormalization <dependency> <groupId>eu.weknowit.tag</groupId> <artifactId>wp2-tag-normalization</artifactId> <version>1.0-SNAPSHOT</version> </dependency>

Service name	Service interface(s) and dependency
WP2_TagProcessing	eu.weknowit.media.tagprocessing.ITagProcessingService <dependency> <groupId>eu.weknowit.media</groupId> <artifactId>wp2-tag-processing</artifactId> <version>1.0-SNAPSHOT</version> </dependency>
WP2_SpeechIndexing	eu.weknowit.media.speech.ISpeechIndexingService <dependency> <artifactId>wp2-speech-indexing-service</artifactId> <groupId>eu.weknowit.media</groupId> <version> </version> </dependency>
WP3_LocalTagCommunityDetector	eu.weknowit.mass.tcd.osgi.ILocalTagCommunityDetector <dependency> <artifactId>wp3-local-tag-community-detector</artifactId> <groupId>eu.weknowit.mass</groupId> <version>1.1-SNAPSHOT</version> </dependency>
WP3_AnswerSpamDetector	eu.weknowit.mass.spam.osgi_wki_spamdetectionservice.IAnswerSpamDetector <dependency> <artifactId>wp3-lexical-spam-detector</artifactId> <groupId>eu.weknowit.mass.spam</groupId> <version>1.0-SNAPSHOT</version> </dependency>
WP4_CommunityDesignLanguage	eu.weknowit.social.cap.cdl.ws.IExecuteCdl <dependency> <groupId>eu.weknowit.social</groupId> <artifactId>wp4-cap-cdl-wc</artifactId> <version>1.3-SNAPSHOT</version> </dependency>

Service name	Service interface(s) and dependency
WP4_Cat_Algorithms	eu.weknowit.social.cat.algorithms.ws.ISNAGraphStatistics eu.weknowit.social.cat.algorithms.ws.ISNABetweennessCentrality eu.weknowit.social.cat.algorithms.ws.IInverseDistanceClosenessCentrality eu.weknowit.social.cat.algorithms.ws.IHermiteanEigensystemDecomposition <dependency> <groupId>eu.weknowit.social</groupId> <artifactId>wp4-cat-algorithms</artifactId> <version>1.0-SNAPSHOT</version> </dependency>
WP5_LogMerger	eu.weknowit.organisational.foaf.dgfoaf.service.IGroupManagementService <dependency> <groupId>eu.weknowit.organisational</groupId> <artifactId>wp5-dgfoafservice</artifactId> <version>1.2-SNAPSHOT</version> </dependency>
WP5_GroupManagement	u.weknowit.organisational.lm.osgi.ILogMerger <dependency> <artifactId>wp5-log-merger</artifactId> <groupId>eu.weknowit.organisational</groupId> <version>1.0-SNAPSHOT</version> </dependency>
WP6_DataStorage	eu.weknowit.datastorage.IWkiDataStorage <dependency> <artifactId>wp6-data-storage</artifactId> <groupId>eu.weknowit.datastorage</groupId> <version>1.3-SNAPSHOT</version> </dependency>