WeKnowIt

Emerging, Collective Intelligence for Personal, Organisational and Social Use

FP7-215453

D6.2.2

Definition and implementation of APIs, version 2

Dissemination level	Public
Contractual date of delivery	Month 24, 30.03.2010
Actual date of delivery	Month 25, 30.04.2010
Workpackage	WP6, Architecture and Integration
Task	T6.2 Definition and implementation of APIs
Type	Prototype
Approval Status	Approved
Version	0.2
Number of pages	95
Filename	D622-man_2010-04-30_v02_smind_deliverable-prototype-description.doc

Abstract

This document accompanies the WeKnowIt (WKI) System prototype and presents the Application Programming Interface (API) of the WKI System.

Firstly, the WKI System is described and integration with use-case applications is presented. Later on functionality and API of the prototype is described and its installation and usage are shown.

Second part of this document lists services declared by all WPs which constitute API of the WKI System. This part is followed by information regarding development of services, and guidelines for service developers.

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1	27.04.2010	First version	T.Kaczanowski
0.2	30.03.2010	Changes after internal review	T.Kaczanowski

Author list

Organization	Name	Contact Information
Software Mind	A. Boruch	a.boruch@softwaremind.pl
Software Mind	R. Janik	r.janik@softwaremind.pl
Software Mind	T. Kaczanowski	t.kaczanowski@softwaremind.pl
Software Mind	P. Wesołowski	p.wesolowski@softwaremind.pl
UoKob	F. Schwagereit	schwagereit@uni-koblenz.de

Executive Summary

This document constitutes the final version of WeKnowIt "Definition and implementation of APIs" deliverable, which provides description of the WKI System prototype and describes the API of the system.

The first version of the WKI System prototype (presented in D6.2.1) presented how technologies, selected as building blocks of the system architecture, integrate and cooperate in order to provide some functionalities to its users.

This second prototype of the WKI System delivers the real functionalities that are used by other applications (i.e. use-case applications implemented by WP7). It presents the cooperation of services provided by WP1-WP5 and offers API that allows other applications to make use of them. The exposed API hides the complexity of the system allowing clients to seamlessly benefit from Collective Intelligence achieved by the complex interoperation of WKI services that are performed internally.

Apart from the prototype, this document also presents:

- the idea behind the WKI System and its architecture (Section 2),
- future evolution of the WKI System prototype (Section 7.1),
- information on the services used within the current WKI System (Section 9),
- list of guidelines for services developers and guidelines related to the installation and use of the D6.2.2 prototype (Section 10).

Abbreviations and Acronyms

API	Application Programming Interface
CDL	Community Design Language
CI	Continuous Integration
CL	Composition Layer
CSG	Consumers Social Group
CSV	Comma-Separated Values
CURIO	Collaborative User Resource Interaction Ontology
CXF	Apache project for service creation and execution
DFS	Distributed File System
DS	WeKnowIt Data Store
ER	Emergency Response
ESB	Enterprise Service Bus
EXIF	Exchangeable Image File Format
FAQ	Frequently Asked Questions
GB	GigaByte
HTTP	HyperText Transfer Protocol
HTTP(S)	Hypertext Transfer Protocol (Secure)
ID	identifier
IDE	Integrated Development Environment
JAR	Java ARchive
JAX-RS	Java API for RESTful Web Services
JAX-WS	Java API for XML - Web Services
JBI	Java Business Integration
JCR	Java Content Repository
JDK	Java Development Kit
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extension
OSGi	OSGi Alliance (formerly Open Services Gateway Initiative)
POI	Point of Interest
POJO	Plain Old Java Object
POM	Project Object Model
QA	Question & Answer
RDBMS	Relational Database Management System
RDFa	Resource Description Framework – in – attributes
REST	Representational State Transfer
RMI	Remote Method Invocation
SOA	Service-Oriented Architecture
SQL	Structured Query Language
URI	Uniform Resource Identifier
WKI	WeKnowIt
WP	Work package
WS	Web service

XML	eXtensible Markup Language

Table of Contents

List of Figures

List of Tables

1. Introduction

This written report of deliverable D6.2.2 "Definition and implementation of API, ver 2" accompanies the second WKI System prototype.

The aim of this prototype is to present the API of the WKI System. This API is created on basis of the expectations of use-case applications and reflects the Collective Intelligence achieved by the WKI project.

This prototype presents and proves that the API of the WKI System:

- provides access to the underlying WKI services,
- is suitable for other applications to use it,
- is already available and ready to use.

At the same time the prototype of the WKI System proves that the integration of its components (services provided by all projects partners) is implemented and provides useful functionalities. Also cooperation with external applications is presented.

This deliverable continues and refers to the previous deliverables of WP6, i.e.:

- D6.1.2 "Identification of architecture elements and relations, version 2 "
- D6.2.1 "Definition and implementation of APIs, version 1"
- D6.3 "Design, architecture and implementation of knowledge base"
- D6.4.1 "Integration of WP 1-5 tools and common components, version 1"

This written report discusses also the possible enhancement of the actual prototype that will be implemented in parallel with WP7 use-case applications development and progress of WP1-WP6 services implementation.

2. The WKI System Architecture

This section provides information on the WKI System - its purpose and technical solutions used.

For the full description of the WKI System (its requirements, technologies stack, architecture and development process) please refer to D6.1.2 "Identification of architecture elements and relations, version 2".

According to the Definition of Work, the WKI project shall provide *"system platform and the necessary user interfaces for creation, storage, manipulation and consumption of Collective Intelligence"*. The WKI System groups services created by all WPs, and integrates them in order to provide functionality that allows to create various applications with different user interfaces.

This implies that the WKI System is a "backend" solution, which can be used by various application, but is not intended to be used by final users on its own. The WKI System provides methods that allow to use WP1-WP5 services in a consistent manner. It also provides access to compositions of these services, which means that one API method of the WKI System can execute several methods of underlying services.

Technically the WKI System is realised using SOA approach and has a layered architecture with well defined responsibilities of each layer. Services of the WPs are delivered as OSGi bundles and integrated via ESB. The technical aspects of the WKI System are discussed in D6.1.2 and the integration with use-case applications is discussed in D7.3.1.

The WKI System consists of the following layers (described in more details in the next sections):

- service layer - provides runtime environment for services,
- composition layer - combines functionalities provided by services and exposes a REST API,
- translation layer - translates incoming and outgoing messages to/from format required by the client.

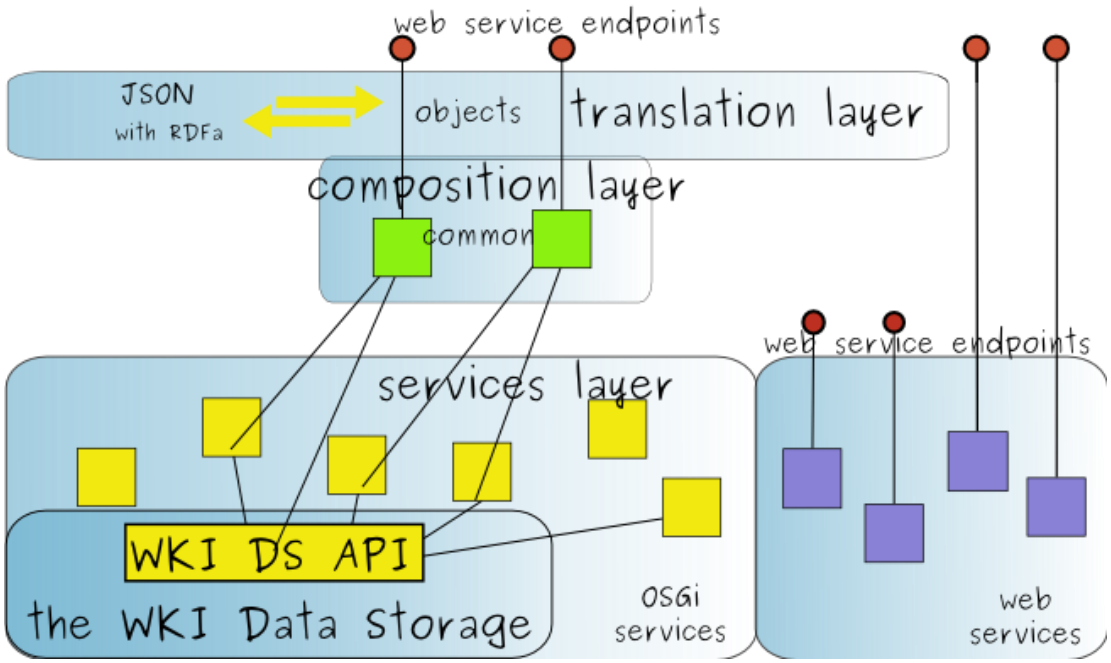


Illustration 1: Layers of the WKI System

Picture 1 presents layers and elements of the WKI System. The following subsections describe the role of each element presented on this picture.

2.1. Services layer

The services layer provides an environment for execution of the WKI services implemented by WP1-WP6.

The WKI services are the basic building blocks of the system. They were created according to the development guidelines and their compatibility with the WKI System has been tested. WKI services are continuously built on the WKI CI server. The reports of finished builds (including Javadocs[1]) are available for all partners. The services are also described on the wiki page of the project which allows for quick recognition of available functionalities.

2.1.1. OSGi environment

The majority of the WKI services are deployed to the WKI System as OSGi bundles and accessible via the OSGi registry. For the description of available services please refer to Section 9.

[1] http://en.wikipedia.org/wiki/Javadoc

OSGi runtime environment is provided by Fuse ESB 4.2[2]. This environment provides possibility of communication between services deployed via OSGi registry. This internal API of the WKI System is used for the communication of services. External applications (e.g. use-case applications) do not have direct access to this API.

2.1.2. Services hosted outside the WKI System

Some services provided by WP1-WP6 are hosted outside the WKI System and are accessible by net connection (via REST API). Nevertheless they are integrated with other services of the WKI System and cooperate with them via connections implemented in the composition layer.

2.1.3. The WKI Data Storage

The WKI Data Storage is a central part of the WKI System, as it is responsible for the storage of every piece of information in the system. The WKI DS provides various storages hidden by the common, consistent and easy to use API.

Beneath the API a hybrid solution is implemented which uses triple store

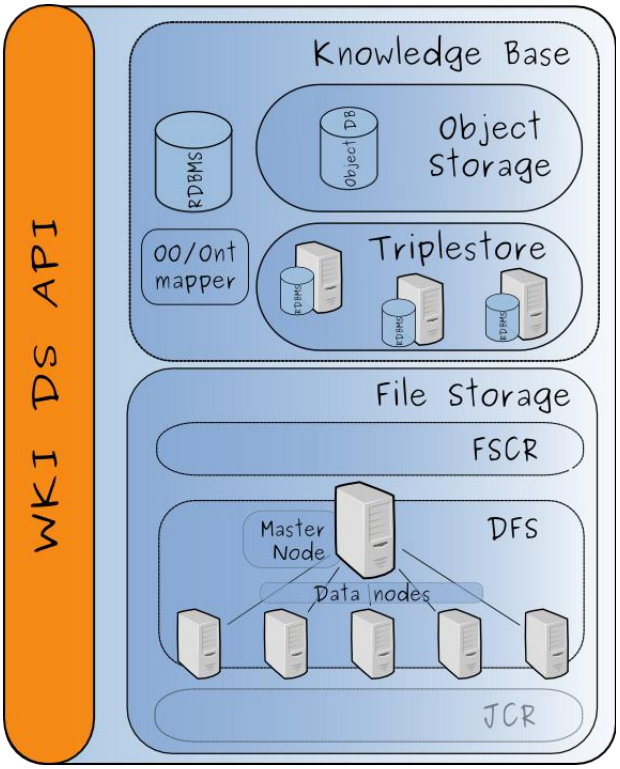


Illustration 2: Components of the WKI Data Storage

and object database (a RDBMS is also used to store some WKI DS specific data). This approach allows to achieve a unique combination of flexibility offered by triple store and speed offered by object database - a valuable features for applications that demand both fast objects access (e.g. re-

[2]http://fusesource.com/products/enterprise-servicemix4/

trieval of user/image/event data) and complex search capabilities (i.e. SparQL entry point).

Another kind of data that can be stored in the WKI DS are files. The file storing component was designed with multimedia files on mind which makes it capable to store huge amounts of data.

Picture 2 presents the internal structure of the WKI DS. It shows the common API which handles all requests from the client and hides the components of the WKI DS. Various storages are also presented - Object Storage and Triple Store for handling metadata and various implementations of content repository for storing files.

The WKI DS is configurable and allows to use different strategies for storage/retrieval of data, which makes it possible to tailor the behaviour of the WKI DS to the needs of an application that uses it as a storage. Similarly different implementations of the file storage can be used (e.g. JCR, Hadoop DFS, plain filesystem) depending on the requirements of the application.

Technically the WKI DS is one of many services integrated within the WKI System - it is an OSGi bundle which exposes its API via the OSGi registry.

For the full description of The WKI Data Storage please refer to D6.3 "Design, architecture and implementation of knowledge base".

2.1.4. Common model

This element was not presented on the picture, but its existence is essential for the WKI System to function. It is crucial that all services use a "common language" so they can exchange the data. Such requirement was realised by establishing of a "common model" which specifies the main entities used through the system and makes it possible to exchange data between services.

A common model was developed thanks to the cooperation of many partners who discussed its requirements and implementation issues. Different aspects of this issue (e.g. ontology and POJO development) where handled by different partners working together in a special interest group inside the WKI project.

For the description of common model please refer to deliverable D6.4.1. "Integration of WP 1-5 tools and common components version 1".

Technically the common model can be seen as:

- a set of POJOs that are exchanged between services (often used in API signatures of methods exposed by the WKI services),
- mappings of the aforementioned POJOs to triples, which allows to store them in the triple store component of the WKI Data Storage.

The first version of common model was finished. Some further development is required to make it compatible with the latest ontologies developed by project partners (i.e. CURIO[3] ontology).

2.2. Composition layer

In order to provide functionality to the end users of the WKI System composition layer combines functionalities offered by services implemented by WP1-WP6 work packages.

Composition layer is used to mediate between the WKI System and applications built on the top of it. It combines services living in the services layer into integrated components of higher order and makes them available to external clients thus allowing them to execute functionalities of many services with one API call.

Technically the composition layer is a set of OSGi bundles that is deployed to the OSGi environment of the WKI System. Its role is to expose methods of services (that are available internally within the WKI System via the OSGi registry) to 3rd party applications (via REST/HTTP API).

Three OSGi bundles constitute the composition layer. Two of them - wp7-compositionlayer-er and wp7-compositionlayer-csg - contains functionality specific to ER and CSG use-cases respectively and are implemented within WP7 work. The third one - wp7-compositionlayers-common[4] - groups generic kind of functionalities that are expected to be reusable among ER, CSG and any other application that can be created on top of the WKI System. This bundle is implemented in cooperation of WP6 and WP7, as some of the functionalities originally emerge in one of the use-cases and then are moved to "common" after their generic nature is recognized.

All Data Flows described in Section 4 are implemented as a part of the wp7-compositionlayer-common project.

2.3. Translation layer

The translation layer decouples the data accepted and returned by the WKI System from their representation. It is an additional layer which role is to provide clients of the system (i.e. use-case applications) with the data in format required by them.

At the moment, this layer is used by the UI components of the ER use case, which require to obtain data as JSON/RDFa. In the future it is possible to extend this layer so the clients can retrieve data also in other formats.

Technically this functionality is implemented as JAX-RS MessageBodyReader[5] and MessageBodyWriter[6].

[3] http://purl.org/net/curio/ns#

[4] the name - with "wp7" prefix - is for historical reasons, even thought it is rather developed by WP6

[5] http://jackson.codehaus.org/javadoc/jax-rs/1.0/javax/ws/rs/ext/MessageBodyReader.html

2.4. Use of the WKI System by use-case applications

Picture 3 presents how the application built on top of the WKI System communicates with the services of the system via translation and communication layers.

Layers presented in the previous sections were specified in such manner that integration with use-case applications is possible. This subsection discusses this topic.

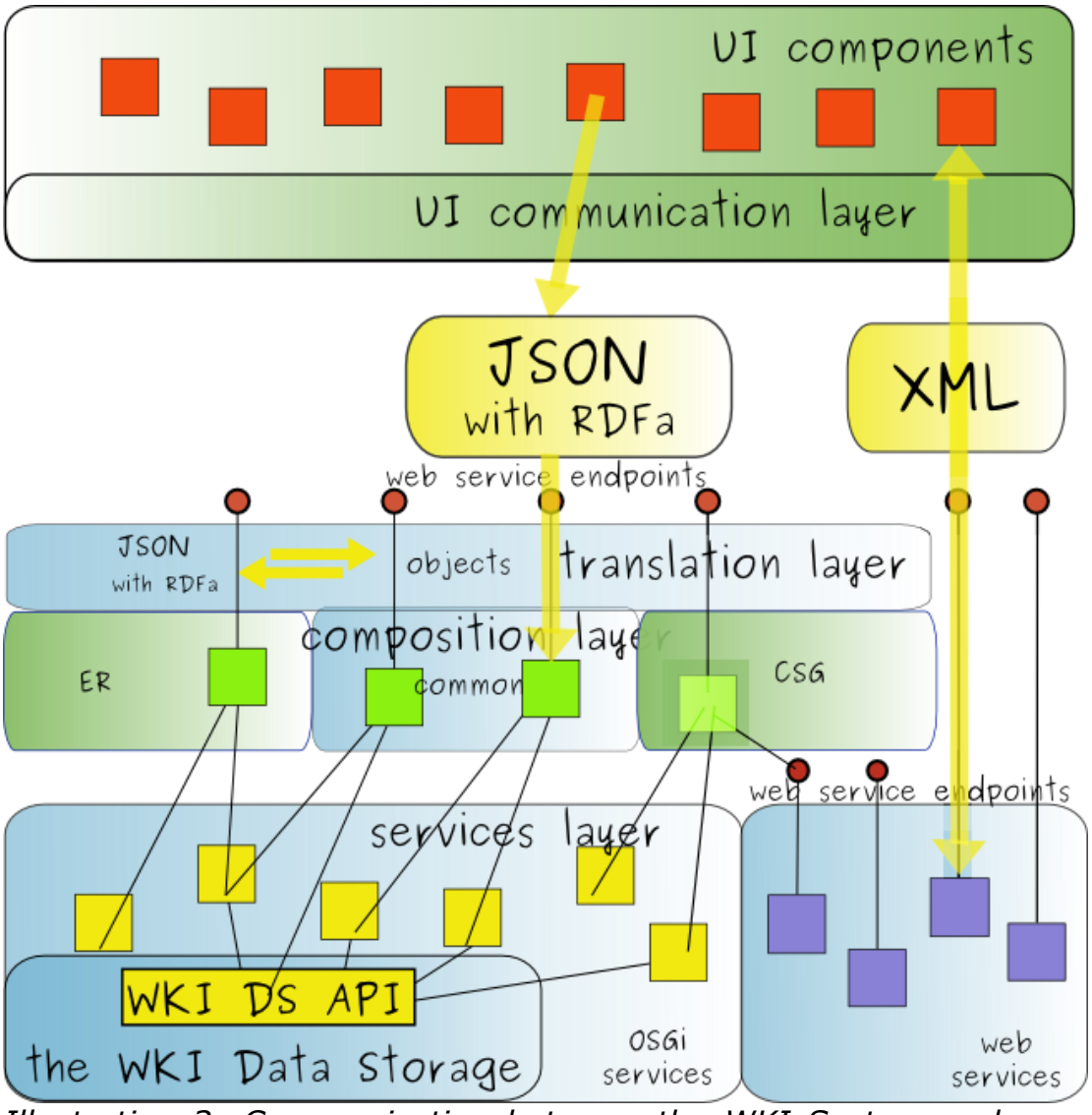


Illustration 3: Communication between the WKI System and use-case application

Depending on the requirements of the application the messages can be exchanged in XML or JSON/RDFa. As it was mentioned in the description

[6]http://jackson.codehaus.org/javadoc/jax-rs/1.0/javax/ws/rs/ext/MessageBodyWriter.html

of the translation layer additional formats might be implemented in the future.

The picture also presents, that applications built on top of the WKI System communicate only with the exposed REST API exposed by the composition layer being unaware of the internal structure and components of the system.

Elements of the composition layer are expected to evolve along with the changes introduced within the use-case applications.

3. The WKI System prototype

The WKI System prototype of M24 presents the current state of the WKI System as a whole. It contains storage components (the WKI Data Storage), business logic implemented in WP1-WP5 services, collective intelligence emerging from cooperation of these services (composition layer) and exposed API that make it useful for external applications.

The technologies used in this prototype were already presented in D6.1.2 (M12), D6.2.1(M12) and D7.3.1 (M19) and possibility and benefits of their cooperation was proven. This prototype goes further by showing a lot of functionality working together and by providing a coherent API to be used by applications build on top of it (i.e. use-case applications implemented by WP7).

This section contains information about the WKI System prototype. First, it presents main components of the prototype and specifies its API. Next, it describes how the WKI System prototype can be installed and run. Finally, usage of test client application is explained.

3.1. Components of the WKI System prototype

This section explains the components used in the WKI System prototype. The architecture of the prototype does not differ from what was presented in Section 2.

The prototype delivers a runtime environment for WP1-WP6 services. It consists of 17 OSGi bundles which gives 23 services (some bundles expose more than one service) of various size (granularity). Composition layer uses these services and provides REST API to external clients of the system (i.e. use-case applications).

Also a simple client application is delivered, that can be used to exercise the system and present offered functionalities. Its role is to upload some files and then retrieve the information (metadata) that was generated by the system based on the content of the uploaded files.

For detailed information on technologies and architecture of the WKI System please refer to D6.1.2.

This section presents an overview of services integrated within the WKI System prototype. For the full description of services please refer to Section 9.

Each of the following sections presents services implemented by different workpackages.

WP1

service	description
WP1_AccountManage	The service allows a new user to register in the

service	description
r	WKI System.
WP1_Login	The service allows users with already created accounts to authenticate via a username and password or, preferably, through the use of the OpenId[7].
WP1_ManageItem	The service is concerned with uploading and deleting items like images, videos, and documents in the WKI DS.
WP1_Tag	The tag service allows users to add, change and delete tags that have been assigned to the uploaded items.
WP1_Comment	The 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.
WP1_Rating	The service allows users to assign a numeric rating to the uploaded item (e.g. file) or entered data (e.g. tag or comment).
WP1_UsersMessaging	The service provides functionality allowing users registered in the WKI System to exchange messages between each other.
WP1_SearchKB	The service allows to query other services that provide search functionality using the same expression and returns combined results.

WP2

service	description
WP2_SemanticPhotoQuery	The service allows to retrieve Flickr photos that best match the domain ontology concepts for specific time intervals.
WP2_SpeechIndexing	The service allows to index speech recordings and search the index for specified expressions.
WP2_TagNormalization	The service allows to match the list of tags that have been assigned to each resource to one or more domain ontology concepts.

[7] http://openid.net/

service	description
WP2_TagProcessing	The service allows to exploit textual tags derived from visual content within WKI System in terms of tag matching and similarity.
WP2_TextAnnotation	The service allows to automatically annotate entities in the text, using predefined annotation models.
WP2_TextClassification	The service allows to determine the similarity between a given text (set of texts) and predefined language models relating to different categories.
WP2_TextClustering	The service allows to cluster a set of texts into related clusters.
WP2_VisualAnalysis	The service allows to exploit visual content within the WKI System in terms of visual similarity matching, retrieval and localization.

WP3

service	description
WP3_LexicalSpamDetection	The service allows to flag an answer to a question, in the context of a QA system, as spam.
WP3_LocalTagCommunityDetection	The service allows to identify a collection of related tags that form a community around given set of tags.
WP3_POIRecommender	The service allows to identify a collection of POIs related to given input POI.

WP4

service	description
WP4_CommunityDesignLanguage	The service allows to manage permissions in the WKI System.

WP5

service	description
WP5_GroupManagement	The service allows to define, modify, and delete organizations and groups. The overall goal of this service is to facilitate efficient task management in a virtual organization for ER

service	description
	through the task manager.
WP5_LogMerger	The service merges a set of event logs referring to the same incident to a single log and allows to search the merged log.

WP6

service	description
WP6_DataStorage	The service provides API for storing/retrieving data from the WKI Data Storage. This API covers all components of the WKI Data Storage - triple store(s) and other databases, and file storages.

3.2. The WKI System prototype API

The API of the WKI System was designed taking into account the requirements of both use-case applications of the WKI project (ER and CSG) that are implemented by WP7. The API presented in this deliverable was already used in the first prototypes of WP7.

As described in Section 2.2, the WKI System exposes REST API. All methods described in this section are accessible via HTTP request.

Below the API of the system is presented.

Classes and methods presented in this section are general in nature and are used by both use-case applications (and are expected to be used by any other application built on top of the WKI System).

From the client point of view the most important part of the API resides in eu.weknowit.composition.layer.common.dataflows package. Because of this, also the description of this package is more detailed. It contains not only classes information but also lists and explains the most important methods.

Packages	
eu.weknowit.composition.layer.common	API of the common composition layer.
eu.weknowit.composition.layer.common.dataflows	Interfaces of data flows.
eu.weknowit.composition.layer.common.dataflows.impl	Implementations of data flows.
eu.weknowit.composition.layer.common.providers	Providers for Rest services.

eu.weknowit.composition.layer.common.utils	Utility classes.

3.2.1. Package eu.weknowit.composition.layer.common

API of the common composition layer.

Exception Summary	
CLException	Exception class for all composition layer components

3.2.2. Package eu.weknowit.composition.layer.common.dataflows

Interfaces of data flows. Please see the detailed description in Section 4.

Interface Summary	
ICapService	ICapService provides API for executing CDL queries in the WKI System. **String** **executeCDL**(**String** query) Executes given CDL Query in the WKI System.
IDocumentService	IDocumentService provides API for retrieving documents from the WKI Data Storage. **String** **getDocument**(**String** docId) Retrieves the document with given ID and returns it in JSON format.
ISparqlService	ISparqlService provides API for executing SPARQL queries in the WKI Data Storage. **String** **executeSparql**(**String** query) Executes given SPARQL query in the WKI Data Storage. **String** **executeSparqlJson**(**String** query) Executes given SPARQL query in the WKI Data Storage.
UploadContentFlow	UploadContentFlow provides API for performing additional actions on uploaded content. **String** **upload**(eu.weknowit.commons.model.dto.Document document, **String** userId) Performs additional actions on uploaded content.

UploadService	UploadService provides API for uploading various types of content to the WKI System. **String upload(InputStream** inputStream, **String** userId) Uploads single file into the WKI System. **String uploadFromFlickr(String** request) Uploads files from Flickr[8] into the WKI System.

3.2.3. Package eu.weknowit.composition.layer.common.dataflows.impl

Implementations of data flows.

Class Summary	
AbstractCLService	AbstractCLService class provides common functionality for common layer services.
CapService	Implementation of **ICapService** interface.
CapWrapperService	Class CapWrapperService provides higher level functionality for managing permissions.
DocumentService	Implementation of **IDocumentService** interface.
SparqlService	Implementation of **ISparqlService** interface.
UploadAudioFlowImpl	Class provides set of additional actions to perform when 'audio' content is uploaded into the WKI System.
UploadImageFlowImpl	Class provides set of additional actions to perform when 'image' content is uploaded into the WKI System.
UploadServiceImpl	Implementation of **UploadService** interface.
UploadTextFlowImpl	Class provides set of additional actions to perform when 'text' content is uploaded into the WKI System.

3.2.4. Package eu.weknowit.composition.layer.common.providers

Providers for Rest services.

[8] http://www.flickr.com/

Class Summary	
InputStreamProvider	InputStream provider necessary to send inputStream through RestService.

3.2.5. Package eu.weknowit.composition.layer.common.utils

Utility classes.

Class Summary	
CLUtils	Class with utility methods for CL services.
RequestDecoder	RequestDecoder provides utility methods for request's body.

4. Data Flows

"Data flows" is a codename for connections of services implemented in the composition layer. These connections model the sequence of calls to methods exposed by the WKI services and exchange of information between them.

"Data flows" were designed in a collaboration of all workpackages.

Technically they are implemented in the composition layer. The entry point to "data flows" are methods from eu.weknowit.composition.layer.common.dataflows package (described in Section 3.2.2). The upload of files is handled by methods defined in UploadService interface.

D6.2.2 prototype of the WKI System presents "Data flows in action". The client (a part of the system prototype described in Section 6) is able to start the "data flows" and then retrieve information collected and generated during their execution.

In the following sections all “data flows” are discussed. First two actions of the system, that happen when a new file is uploaded, are discussed. Then the upload of images, audio and text files is described.

4.1. Permission Check

Before any file is uploaded to the WKI System, permissions are checked to verify if the user is allowed to upload files. This action is executed first so in case the user does not have sufficient permissions other actions (which are resource-consuming) are not executed.

After the file is uploaded the ownership of the file is set to the user who uploaded it so he/she can retrieve it later.

These functionalities are provided by WP4_CommunityDesignLanguage service.

4.2. Mime-type[9] Recognition

It is expected, that users of the applications built on to of the WKI System will upload files of various types (e.g. images, videos, audio files). Based on the type of the uploaded file the WKI System should react differently, i.e. should execute different services in order to analyse the uploaded file.

Because of this the mime-type of every file uploaded to the WKI System must be recognized. Further actions of the WKI System are determined by the mime-type.

The recognition of the mime-type is performed by one of internal services of the WKI DS.

[9]http://en.wikipedia.org/wiki/MIME_type

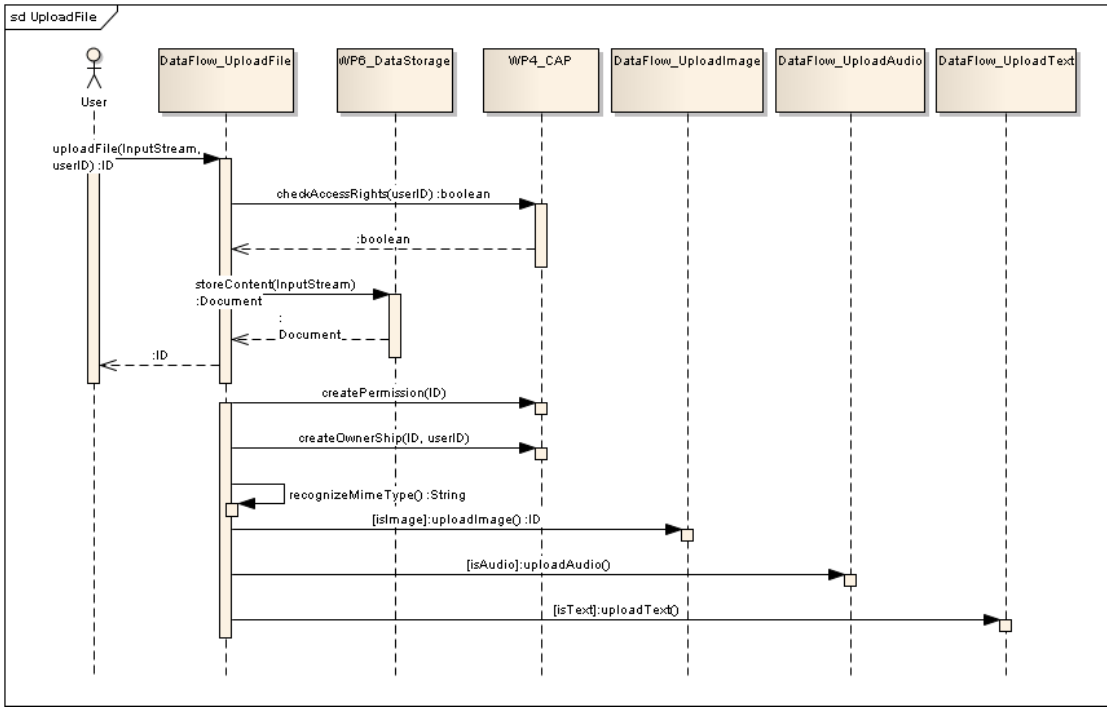


Illustration 4: Sequence diagram of file upload (permission check and mime-type recognition)

Picture 4 presents the initial phase of file upload. It shows the two preliminary steps - permission check and mime-type recognition. The sequence ends when the appropriate successive flow is chosen based on the mime-type of uploaded file.

It is worth to notice that the ID of the uploaded file is returned to the user as soon as the file is stored. The user can continue its operations without waiting for execution of all services of the WKI System. This is very important as the operations performed by some services might be time-consuming.

The following sections discuss the flows for specific types of uploaded files.

4.3. Audio File Upload

The WKI System realises two functionalities in this flow. First, it stores the uploaded file in the WKI DS module. Second, it creates indexes that will allow for searching for the information included in the uploaded audio file.

The operation of indexing is executed asynchronously because its execution time can not be predicted so the user should not wait for it. Because of this the ID of created document is returned right after the document is stored in the WKI DS element.

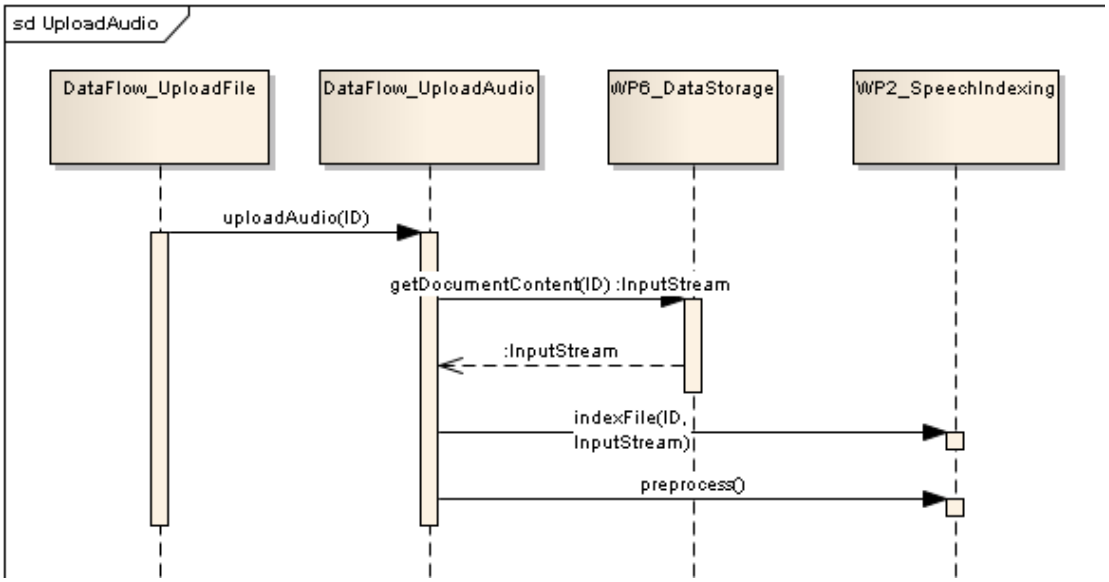


Illustration 5: Sequence diagram of audio file upload

As seen on picture 5 services of WP2 and WP6 are involved.

4.4. Image Upload

When an image is uploaded to the WKI System various services (implemented by WP1, WP2, WP3 and WP6) are invoked in order to:

- extract metadata from this image,

- enhance set of metadata related to this image based on some other available data.

First, the WKI System stores the uploaded image file in the WKI DS module. This allows for further analysis - recognition of mime-type and extraction of location data (from EXIF tags[10], if available). These two tasks are performed internally by the WKI DS service.

Then, WP2_VisualAnalysis service is requested to provide information on similar images (along with their tags and geo-location information).

Based on the information returned by WP2_VisualAnalysis service, metadata of the uploaded image are enhanced with geo-location (saved by the WP6_DataStorage service) and tags (saved via WP1_Tag service).

Next, WP2_TagNormalization service performs tags normalization; results of work of this service are stored with use of WP1_Tag service and WP6_DataStorage service. Finally WP3_LocalTagCommunityDetection performs its task and returns a list of tags which are also handled by WP1_Tag and WP6_DataStorage services.

[10]http://en.wikipedia.org/wiki/Exchangeable_image_file_format

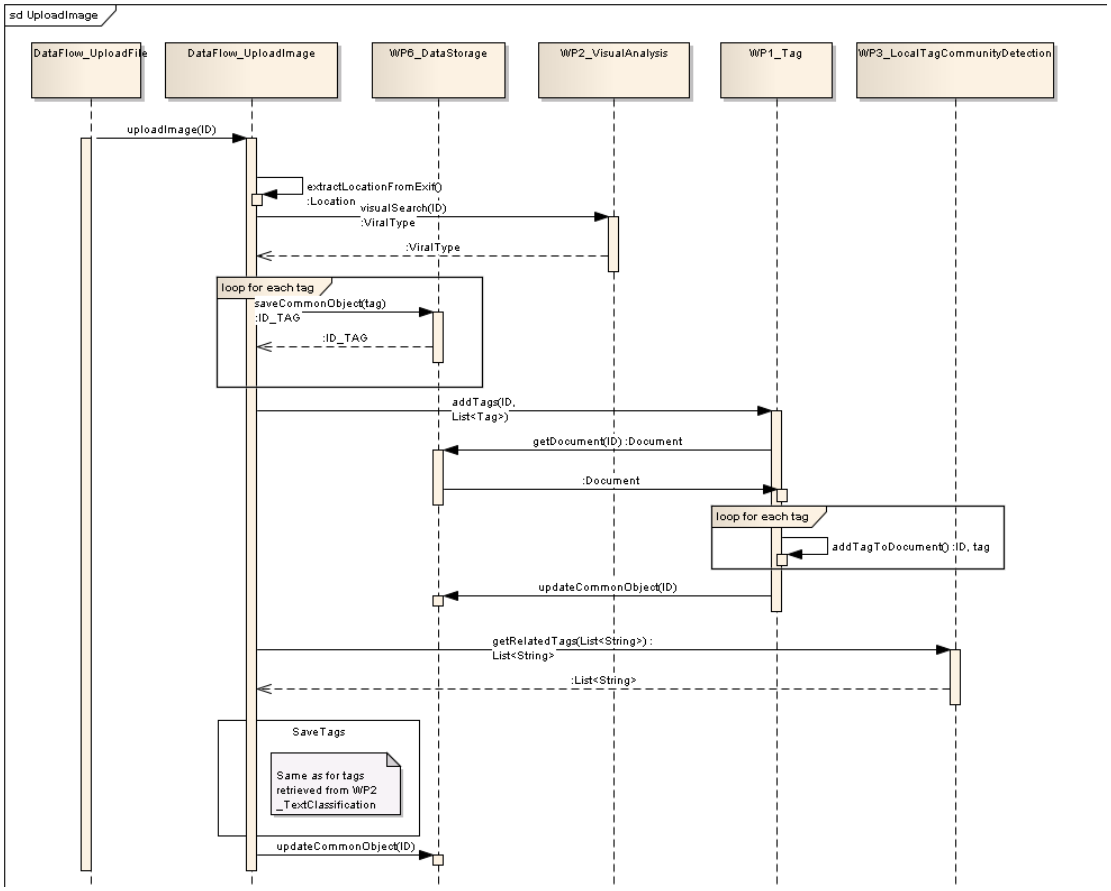


Illustration 6: Sequence diagram of image file upload

4.5. Upload of Flickr Images

Apart from the single image upload scenario described in Section 4.4 the WKI System also provides a possibility to batch download images from Flickr. The idea is that the user can request the WKI System to acquire Flickr images (providing time period and geo location parameters to filter the results). The images are fetched from Flickr by WP2_SemanticPhotoQuery service and then each image is uploaded to the WKI System with the "Upload image" flow as described in Section 4.4.

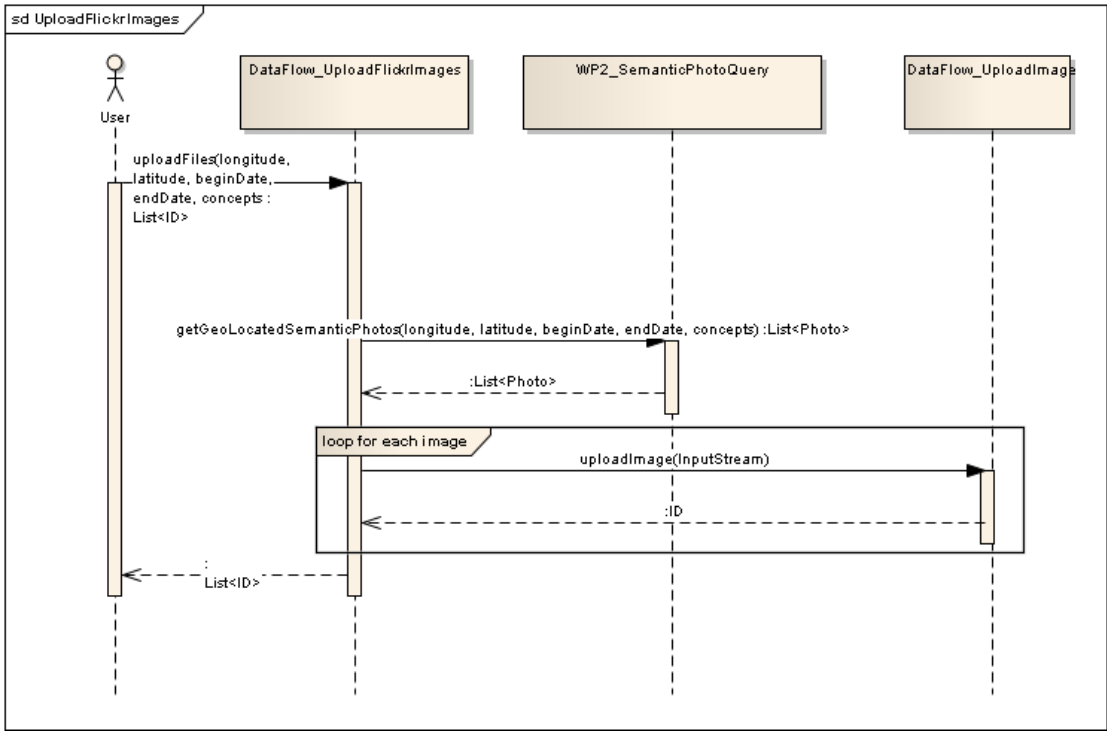


Illustration 7: Sequence diagram of upload of Flickr images

4.6. Text File Upload

The WKI System is capable of storing text files. Such files are passed to services implemented by WP2 and WP3 that deal with the textual content.

- WP2_TextAnnotation,
- WP2_TextClassification,
- WP3_LocalTagCommunityDetection.

Information extracted/generated by these services is saved in the WKI DS by WP1_Tag and WP6_DataStorage services.

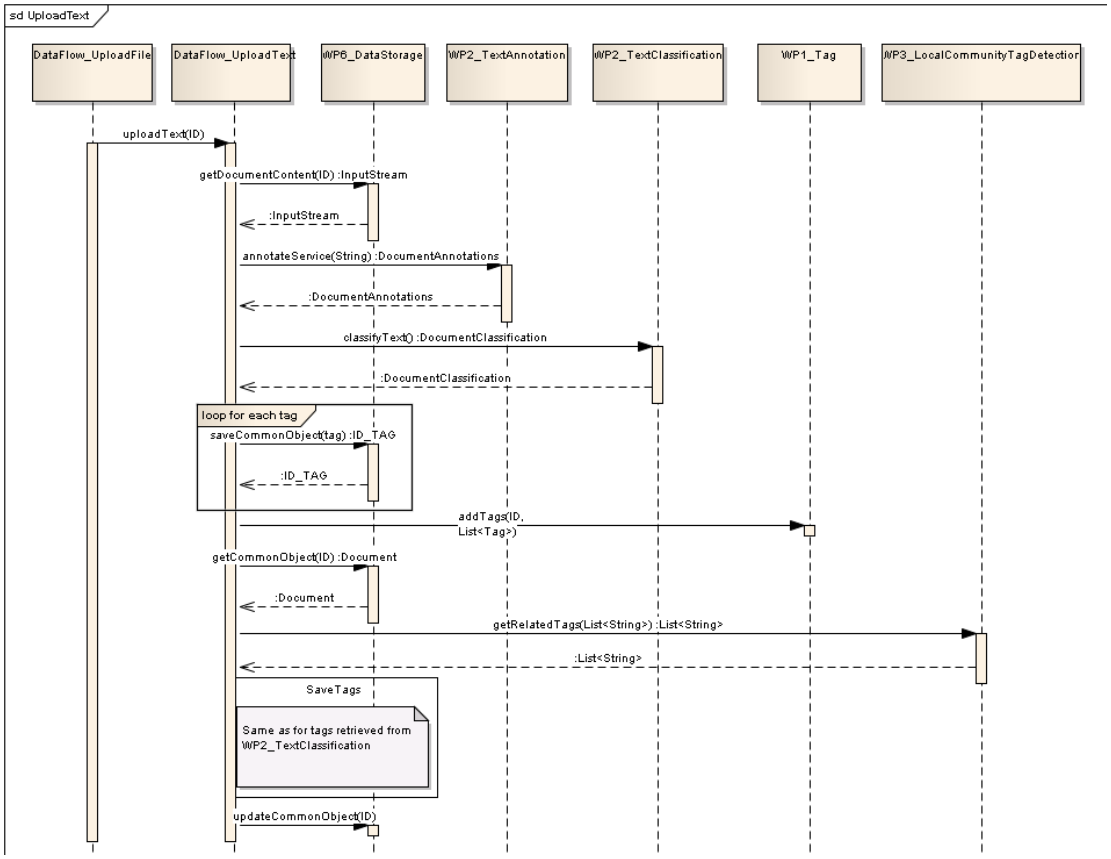


Illustration 8: Sequence diagram of upload of text file

4.7. Upload of Generic File

In case of files which mime-type does not belong to any of above groups or is not known (can not be recognized) the WKI System simply stores them in the WKI DS and returns generated ID to the client.

4.8. Retrieval of Data

After the file is uploaded some metadata are extracted from it or generated by WKI services and stored in the WKI DS as described in previous points. Client application can now retrieve these information using methods exposed by IDocumentService and ISparqlService via REST API (described in Section 3.2.2):

- String getDocument(String docId),
- String executeSparql(String query),
- String executeSparqlJson(String query).

Such requests are served by WP6_DataStorage service which provides access to the data stored in the WKI DS.

4.9. Data Flows - Conclusions

Data flows described in this sections show cooperation of services of the WKI System. A supposedly simple action of file upload causes execution of

many services and generates a lot of data that are processed and stored in the WKI DS. Later on this data can be retrieved and presented to the user in a meaningful way.

Next two sections discuss the practical use of the WKI System prototype. First, installation of the system is described (Section 5). Then description of client application (which allows to examine the functionalities of the system) is given (Section 6).

5. Installation of the WKI System prototype

This section describes steps required in order to install and run the D6.2.2 prototype.

The WKI System prototype **requires Linux operating system** and about 2GB of free disk space. More detailed requirements can be found in "How to install Java and Maven" guideline, Section 10.2. Information on deployment of bundles to the WKI System are included in the "How to deploy OSGi bundles to the WKI System" guideline, Section 10.4.

Prototype of the WKI System is distributed as `wki-d622-prototype.tar.gz` archive file.

5.1.1. Prerequisites

The WKI System requires installation of PostgreSQL[11] database. Information how to install PostgreSQL can be found in "How to install the WKI System" guideline, Section 10.3.

To run scripts creating database structure Java, Ant and Maven must be installed (as described in "How to install Java and Maven" guideline, Section 10.2).

Installation and running of the WKI System requires use of a command line console (Linux shell):

Step1 – preparation

Open *shell* and unpack file `wki-d622-prototype.tar.gz`:

```
tar xvzf wki-d622-prototype.tar.gz
```

A new folder - `wki-d622-prototype` - with a set of folders inside will be extracted. The purpose of these folders is discussed in "How to install the WKI System" guideline, Section 10.3.

Now you need to set a `WKI_HOME` variable. Please notice that:

- the backticks are used and not single quote characters,
- the syntax for setting an environment variable might differ on your linux distribution,
- `WKI_HOME` variable must be set separately for each shell you use.

```
cd wki-d622-prototype
export WKI_HOME=`pwd`
```

Make sure the `WKI_HOME` variable is set:

```
echo $WKI_HOME
```

Now the database schema for the WKI System must be created. Run `build.xml` script from `WKI_HOME/configs` directory:

[11] http://www.postgresql.org/

```
cd $WKI_HOME/configs
ant
```

Execution of Ant script should finish with the following message:

```
BUILD SUCCESSFUL
```

Now the WKI System is ready to start.

Step2 – start of the system

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

```
cd $WKI_HOME
chmod 755 bin/servicemix
chmod 755 bin/start
cd bin
./start
```

The WKI System is running now (in the background). You can check it by executing the following command:

```
ps aux | grep servicemix
```

which should show a running process of the WKI System (the username and folders will differ):

```
rafal    17251 44.1  2.3 2454804 78352 pts/2   Sl   13:08   0:02
/usr/lib/jvm/java-6-sun/bin/java -server -Xms512M -Xmx1500M -
XX:MaxPermSize=800m -Dcom.sun.management.jmxremote -
Dstorage.location=/home/rafal/tmp/prototyp/instances -
Dkaraf.home=/home/rafal/tmp/prototyp -
Dkaraf.base=/home/rafal/tmp/prototyp -
Djava.util.logging.config.file=/home/rafal/tmp/prototyp/etc/java.util.
logging.properties -Dkaraf.startLocalConsole=false -
Dkaraf.startRemoteShell=true -classpath
/home/rafal/tmp/prototyp/lib/karaf-
client.jar:/home/rafal/tmp/prototyp/lib/karaf-jaas-
boot.jar:/home/rafal/tmp/prototyp/lib/karaf.jar:/home/rafal/tmp/prototyp/lib/servicemix-version.jar org.apache.felix.karaf.main.Bootstrap
```

In order to log to the WKI System console type:

```
java -jar $WKI_HOME/lib/karaf-client.jar
```

After the WKI System starts, status of all installed bundles can be checked using *list* command[12]:

```
list
```

Please note that it may take few minutes to install all bundles *(*eventually *list* command should return 192 active and started bundles). To learn about the bundles containing services provided by WP1-WP6 use the following command :

```
list | grep wp
```

The output will show all available WKI services:

[12]Description of basic administrative tasks of OSGi console can be found in Section 10.3

```
karaf@root> list | grep wp
[  34] [Active     ] [            ] [       ] [   60] wp7-
compositionlayer-csg (1.2.0.SNAPSHOT)
[  40] [Active     ] [            ] [       ] [   60] wp3-lexical-
spam-detector (1.2.0.SNAPSHOT)
[  41] [Active     ] [            ] [       ] [   60] wp2-tag-
processing (1.2.0.SNAPSHOT)
[  46] [Active     ] [            ] [       ] [   60] wp2-speech-
indexing-service (1.0.0.SNAPSHOT)
[  50] [Active     ] [            ] [       ] [   60] wp7-
compositionlayer-common (1.2.0.SNAPSHOT)
[  52] [Active     ] [            ] [       ] [   60] wp3-local-tag-
community-detector (2.1.0.SNAPSHOT)
[  53] [Active     ] [            ] [       ] [   60] wp7-
compositionlayer-er (1.5.0.SNAPSHOT)
[  54] [Active     ] [            ] [       ] [   60] wp2-semantic-
photo-query (1.0.0.SNAPSHOT)
[  57] [Active     ] [            ] [       ] [   60] wp1-user-
services (1.2.0.SNAPSHOT)
[  58] [Active     ] [            ] [       ] [   60] wp2-tag-
normalization (1.1.0.SNAPSHOT)
[  60] [Active     ] [            ] [       ] [   60] wp5-log-merger
(1.4.0.SNAPSHOT)
[  62] [Active     ] [            ] [       ] [   60] wp2-visual-
analysis (1.2.0.SNAPSHOT)
[  63] [Active     ] [            ] [       ] [   60] wp2-text-
analysis-services (1.1.0.SNAPSHOT)
[  72] [Active     ] [            ] [       ] [   60] wp5-
dgfoafservice (1.3.0.SNAPSHOT)
[  73] [Active     ] [            ] [       ] [   60] wp4-cap
(2.0.0.SNAPSHOT)
[  79] [Active     ] [            ] [       ] [   60] wp3-poi-
recommender (1.0.0.SNAPSHOT)
[  80] [Active     ] [            ] [       ] [   60] wp6-data-storage
(1.10.0.SNAPSHOT)
```

Now the WKI System is ready to receive requests from client applications which is described in section 6. The REST API of the WKI System is accessible at *http://localhost:8191.*

6. Client application

The general purpose of prototype client application is to test the functionalities (upload and access) of the WKI System prototype. The client is implemented as a standalone application, using the Jersey[13] implementation of JAX-RS technology to create a web service client.

The prototype of the WKI System exposes its functionalities via the REST interface[14]. Its capabilities can be presented by the following generic flow of requests:

- initialize the system,
- provide some input data,
- retrieve the additional data that were generated (extracted) by the WKI System based on the input.

Wp6-d622-prototype-client is a tool used to start the WKI System "Data Flows" (described in Section 4). The client plays the role of external application and connects with the REST API exposed by wp7-compositionlayer-commons (described in Section 3.2). It initiates the data flows by sending a file along with ID of the user or by passing parameters required to start the fetching of Flickr images (Section 4.5).

Depending on the input file mime-type the proper flow is started. The user can upload an image file (Section 4.4), audio file (Section 4.3), text file (Section 4.6).

After data (i.e. files and generated metadata) is uploaded, simple HTTP calls (made with cURL[15] tool) can be used to retrieve the enhanced data in order to present the effects of work of various services of the WKI System that were executed during the upload of files. For information on retrieval options please refer to Section 4.8 and 5.1.1).

6.1. Prerequisites

To use the client application the WKI System must be run first, as described in Section 5.

It is also advisable to watch logs of the WKI System (using separate console):

```
tail -f $WKI_HOME/data/log/karaf.out
```

This way you will be able to watch the actions performed insi

Now unpack the client file `wki-d622-prototype-client.tar.gz`:

[13] https://jersey.dev.java.net/

[14] It is also possible to deploy a custom OSGi bundle within the OSGi environment of the WKI System and gain access to the API of all services (available via the OSGi registry). This solution is not discussed further in this document which concentrates on the externally accessible REST API.

[15] http://curl.haxx.se/

```
tar xvzf wki-d622-prototype-client.tar.gz
cd wki-d622-prototype-client
```

6.1.1. Creating the CAP structure

To make the rights management system work first the appropriate structures within WP4_CommunityDesignLanguage service need to be created and instantiated:

1. A group of "users" is created.
2. A user group "ER personnel" is modelled.
3. A user group "System" is modelled.
4. Users can be assigned to the groups listed above. A "citizen" is then an "user" without an association to "ER personnel". A "system process" is a user with an association to "System"
5. "Pieces of information" (called "files" for simplification) exist and is assigned a collector "files".
6. "Files" have an owner from the "users".
7. "Files" can be assigned a public attribute.
8. The following operations on files are distinguished: "upload", "view", "anything else".
9. Policy I: ER personnel can do anything with a file.
10. Policy II: A user can "upload" files. As can a system process.
11. Policy III: A user can "view" own files.
12. Policy IV: A user can "view" public files.

The client provides a bash script which makes WP4_CAP execute the commands listed above. The commands are executed via WP7 webservice by issuing HTTP POST requests.

Use the following command to execute the script:

```
./cdl/cdl-init-cap.sh
```

The script will execute a number of HTTP requests:

```
curl -d "CREATE SETS \"user\";"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE SETS \"usergroup\";"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE ELEMENTS \"usergroup\": { \"er personnel\", \"system\"
};" http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE RELATION \"user-usergroup\" (\"user\",
\"usergroup\");" http://localhost:8191/cl/capService/executeCDL -H
"Content-Type: text/plain"
```

```
curl -d "CREATE SETS \"file\";"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE RELATION \"owner\" (\"file\", \"user\");"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE SETS \"flag\";"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE ELEMENTS \"flag\": { \"public\" };"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE RELATION \"hasflag\" (\"file\",\"flag\");"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE SETS \"permission\";"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE ELEMENTS \"permission\": { \"upload\", \"view\" };"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE ACCESSCONDITION er_candoanything: {
([\"user\"].\"user-usergroup\", {\"er personnel\"})};"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE ACCESSCONDITION user_canupload: {([\"permission\"],
{\"upload\"})};" http://localhost:8191/cl/capService/executeCDL -H
"Content-Type: text/plain"

curl -d "CREATE ACCESSCONDITION view_ownfiles: {([\"file\"].\"owner\",
[\"user\"]),([\"permission\"], {\"view\"})};"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"

curl -d "CREATE ACCESSCONDITION view_publicfiles: {
([\"file\"].\"hasflag\", {\"public\"}),([\"permission\"],
{\"view\"})};" http://localhost:8191/cl/capService/executeCDL -H "Con-
tent-Type: text/plain"
```

All HTTP requests share a similar pattern. For example:

```
curl -d "CREATE SETS \"user\";"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"
```

part of command	explanation
curl	tool used to issue HTTP requests
-d	sends data as HTTP POST
"CREATE SETS \"user\";"	a CDL command that will be executed by CAP service
http://localhost:8191/cl/capService/executeCDL	address of CAP service

part of command	explanation
-H "Content-Type: text/plain"	content-type HTTP header

6.1.2. Creating the user

When all CAP sets[16], relations, elements and access conditions are created a new user can be added to the WKI System by executing:

```
./cdl/cdl-create-user.sh
```

This script executes WP7-composition-layer-service HTTP POST method:

```
 curl -d "CREATE ELEMENTS \"user\": { \"1234\" };"
http://localhost:8191/cl/capService/executeCDL -H "Content-Type:
text/plain"
```

where the '1234' is an example user ID.

This is a preliminary step required in order to allow further actions. An account for the user must be created and upload permission must be granted so the user can upload files to the system.

Now the WKI System is ready to be tested with wp6-d622-prototype-client by the user with ID '1234'.

6.2. Example Files

Folder input of client application contains example files that can be used to upload to files.

file name	description
inputAudio.wav	Audio file used to initiate audio file upload flow (Section 4.3)
inputText.txt	Text file used to initiate text file upload flow (Section 4.6)
inputImage.jpg	Image file used to initiate image file upload flow (Section 4.4). This file does not contain geo-location data, so the geo-location will be obtained from WP2_VisualAnalysis service.
inputImageWithGeo.jpg	Image file used to initiate image file upload flow (Section 4.4). This file contains geo-location data, so they will be extracted from EXIF tags.

[16]sets, relations, elements and access conditions are CAP specific entities described in D4.2 "Prototype of the Community Administration Platform" deliverable

6.3. Usage of the client application

In order to upload a file to the WKI System (and start one of the data flows as described in Section 4) one have tu:

- compile the wp6-d622-prototype-client application ,
- run client application.de the WKI System that are initiated by execution of client commands.

6.3.1. Compilation of the client application

In order to compile the client application run the following command:

```
mvn clean compile
```

6.3.2. Running of the client application

To run the client application use the following command:

```
mvn exec:java -Dfilename=input/inputImage.jpg -DuserId=1234
```

There are two parameters that needs to be given meaningful values:

Value	Description
-Dfilename=filename	Filename is a name of the file to upload
-DuserId=properUserId	The ID of the user

Now the system behaves in a manner described in Section 4 depending on type of the uploaded file. It starts with permission checks and then (if the permissions are granted) continues with mime-type check.

6.3.3. Permission Check

If the user has permission to upload the file the flow is started. In other case[17] the following message will be printed:

```
Error status: 400 [ User has no permission to upload file]
```

6.3.4. File Stored in the WKI DS

If permissions to upload are granted the file is saved in the WKI DS and generated ID of the document[18] is returned. The following information will be printed:

```
File 'inputImage.jpg' successfully  stored in WKI-DS with document id
= '1269592566964'
```

where the ID is a unique identifier of document which contains uploaded 'example.jpg' file in the WKI System.

[17] This will happen if requests are issued with userID different than the 123 that was registered in CAP in the previous section.

[18] A document is an abstraction over a file and related metadata

6.3.5. Retrieval of Data

The user can get the specified document from the WKI System in JSON format by connecting directly with IDocumentService via REST API using HTTP GET method:

```
curl
http://localhost:8191/cl/documentService/getDocument/1269592566964
```

where the 'http://localhost:8191/cl' is the address of the composition layer service and the '1269592566964' is a resource (document) identifier (as returned by the WKI DS).

The result of this action is a Document presented in JSON format:

```
 {"contentUri":"file://rafal#/home/rafal/tmp/wki-
sys-
tem/fs_cr_repo/1269597443342.file","mimeType":"image/jpeg","taggedCnt"
:{"tagged":[]},"id":"1269592566964"}.
```

This way upload of all types of documents (image, audio and text) and retrieval of data extracted and generated by the WKI System can be tested.

6.3.6. Flickr Images Upload

The Flickr images analysis is also available via REST API. User can start this action by executing HTTP POST method. The service address is '*http://localhost:8191/cl/uploadService/uploadFileFromFlickr*'. The request is a String in CSV format where the following values must be set:

- latitude - the latitude decribing image
- longitude - the longitude decribing image
- beginDate - date in milliseconds
- endDate - date in milliseconds
- userId - the ID of the user

and the content is of type 'text/plain'. The service can be accessed with cURL tool. An example of a valid call is presented below:

```
curl -d "latitude=53.366667&longitude=-
1.5&beginDate=1182722400000&endDate=1190671200000&userId=1234"
http://localhost:8191/cl/uploadService/uploadFileFromFlickr -H "Con-
tent-Type: text/plain"
```

Because the number of images returned from Flickr might be significant this request might take few minutes before finished.

The result can be examined by watching logs.

6.4. Client Application - Summary

As presented in previous sections the client application delivered with the WKI System prototype can be used to examine the functionalities provided by the system. It can be used to store data in the system and observe how these data were enhanced by the cooperation of WKI services.

7. The WKI System Prototype - Conclusions

Previous sctions described architecture (Section 2), connection with the use-case applications (Section 2.4), elements (Section 3.1), functionality (Section 4), installation (Section 5) and usage (Section 6) of the WKI System prototype.

This section provides some conclusions that can be derived from what was presented previously.

The previous prototype (presented in D6.2.1) proved the technological ability of integration and provided some very basic functionality. The current prototype enhances substantially what was presented before. It presents the cooperation of services implemented by different workpackages. This is in accordance with the plans that were presented in D6.2.1 deliverable.

The current prototype is based on the same technology stack that was presented in D6.2.1 but in terms of provided functionalities is much more mature. This prototype delivers true integration of independently developed services thus making a serious step forward towards Collective Intelligence by combining outputs of different services.

Below some of the most important features of the current WKI System prototype are listed:

- integration of WP1-WP6 services which brings benefits to the end user (Collective Intelligence),
- usable external REST API used by use-case applications,
- internal API (via OSGi registry) that is used for integration of services inside the WKI System,
- the WKI DS component capable of storing and retrieval of both files and metadata (including a SparQL endpoint).

7.1. Evolution of the WKI System Prototype

The evolution of the WKI System prototype towards the final WKI System depends on few factors. Some of them were already discussed in D6.1.2 and has not changed since then:

- new services created by WPs and updates of existing services,
  - based on the requirements of use-case applications,
- requests made by use-case applications regarding exposition of some services via REST API,
- internal changes introduced by WP6.

Because of the significant changes in use-case applications (which have been under heavy development since D6.1.2) and because of the continu-

ous efforts towards the implementation of Collective Intelligence a new driving force of the system evolution has emerged:

- requirements regarding Data Flows.

Even though no major changes of technologies used by the WKI System are used, WP6 constantly updates versions of used libraries / frameworks / tools in order to benefit from new, enhanced or fixed functionalities that they provide.

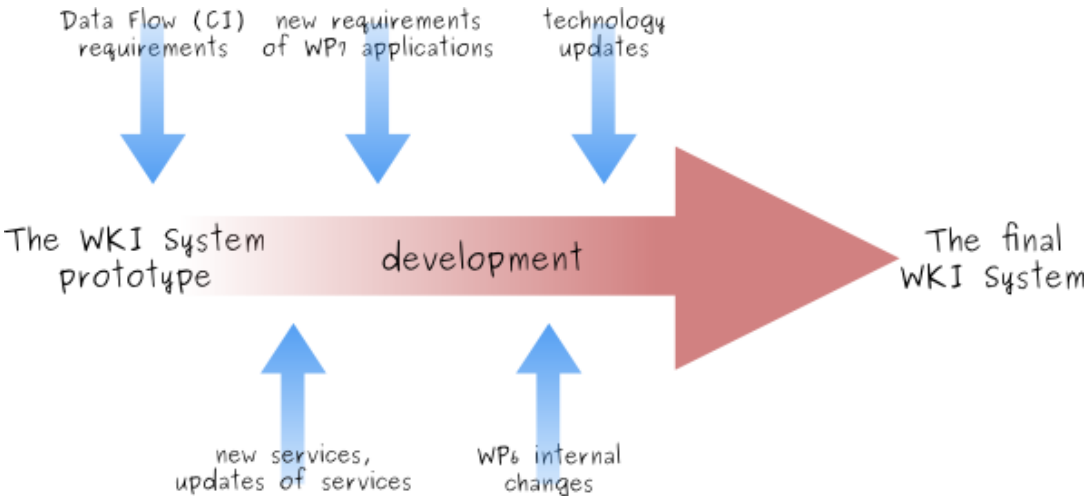


Illustration 9: Driving forces of the WKI System evolution

Picture 9 presents the main forces that push the development of the WKI System.

The consecutive versions of the WKI System are released frequently (the current version number is 0.11). In specific case SNAPSHOT versions are released in order to solve some urgent issues reported by WPs developers.

All released version of the WKI System are downloadable from the WKI infrastructure (WebDAV server).

8. Attachments

Following attachments are available:

- wki-d622-prototype.tar.gz – archive that contains the WKI System prototype and some scripts useful for installation of the system,
- wki-d622-prototype-client.tar.gz – archive with client application.

9. Appendix A - WKI Services

WP1-WP6 implement services - "building blocks that provide well-defined functionality and can be used as element of larger structures" (D6.1.2). These services are developed by different teams, built on WKI CI server and integrated into the WKI System by WP6.

In order to facilitate this process and to make information on services available to all participants of the WKI project, proliferate the information on available services a set of wiki pages was created. Each existing service is described there using the common format as presented in Table 1.

Name	Unique name (serves as identifier), in format WPx_ServiceName, e.g. WP99_StockService
CI Project	Link to CI project.
Description	General purpose - what is the general function.
Required Services	Names of other services this service requires.
Usage example	Which services might use the defined service and/or which services are used by that service. Information about possible interactions with other services to give a good impression of the service functionality. Example use-case (general or for CSG and ER if possible).
Method signature	List of all methods and their signatures provided by this service. Short description of each method: • what does it do, • what parameters are required, • what is returned. (this field might be left blank if a link to javadocs on CI server is given)
Types description	Short description of each complex type used in service method input or output (what data that type contains). Left blank if only simple types (e.g. String, Long) are used.
Exceptions	List of exceptions, reasons and implications.

Table 1 Service declaration schema

Services description are constantly updated and reflects the current status of work and include new services.

This section provides list of services as declared by WPs.

9.1. WP1 services

The services of WP1 aim to provide functionalities for users to upload and access Personal Intelligence in the WeKnowIt systems. In particular services inside WP1 can be divided into three groups:

1. Services for managing the user details and login procedure: this enables to model the user in the background, storing user and context details that are necessary to the working of all types of intelligence. From the user perspective, these services enable the delivery of personalised and contextualised knowledge in a custom-made interface. For example in the ER use case knowledge about the user profile will enable the system to provide different functionalities for the different user groups, i.e. the ER Admin user will be able to access all the data uploaded to the system and publish them for public consumption while a citizen will only be able to browse and search the information that has been previously made public.

2. Content uploading services are defined to allow users to upload content of different type (i.e. images, video, text) and metadata to the WeKnowIt Data Storage. These services will make use of the user details management services (WP1) and the Group Management services (WP5) in order to personalise the interaction. In particular, from the user perspective, these services will be beneficial for enriching the content that is uploaded, therefore having a better search/browse/recommendation experience. These services will be particularly important for example in the CSG use case, as users will be enabled to provide comments and rating over places they visited, thus empowering a better recommendation service for new trips.

3. Search and browsing services are provided to guarantee a more sophisticated access to the knowledge uploaded to the WeKnowIt data Storage, by empowering the user to easily perform complex queries seamlessly. This will be particularly useful i/.e. in Emergency Response situations, as it will allow to quickly gather all the information necessary at a given time.

4. Messaging services provide the functionalities for users to establish a communication with different actors in the system, them being other personal users or organisations. In particular in the ER use case, Messaging services could be adopted to enquire and receive notifications about the status of a member of the family, or to receive constant updates about the status of an event.

Name:	**WP1_AccountManager**
Description	This service allows a new user to register in E-WKI. Once the user has an account, he is able to access the other services provided by WKI. This is centred on the OpenId standard and the concept of a federated identity. A federated identity is one which spans multiple

	information systems.
Usage	A user wants to create an account for using a WKI service. Also, an invitation for doing so could easily be made by forwarding to her a link to the front page or registration page of the application. A number of questions about the person (e.g. age, gender, preferences) or the possibility to upload a photograph of himself/herself will define an initial user profile, at this stage. However, it should be possible that this phase be skipped and/or deferred until a later time in case of on-the-fly account registrations; such an option is essential in time-critical ER situations. The user can always modify the details of his account or delete it.
Required Services	None.
Method signature	*Credentials createUser(String userName,boolean onThefly,String password)* *boolean resetPswd(int UserId, String oldPassword, String newPassword)* *boolean modifyUser(int UserId, Attribute PropertyX, String password)* *boolean deleteUser(int userId, String password)*

Types description	typedef struct {
	bool onthefly; //?
	int userID;
	String password; //validation check: password must be at least 8 characters with at least 2 numbers }
	Credentials;
	Attribute-> An OpenID AX RDF Triple

Exceptions	Bad Password userName already existing UserId already existing Bad UserName

Table 2 WP1_AccountManager

Name	**WP1_LogIn**
Description	The login service is the main entry point to the rest of the services in WKI, 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.
Usage Example	A registered user can log in to his/her created account and be granted access to it.
Method signature	*boolean userLogin(URI OpenID, Token)* *boolean userLogout(URI OpenID, Token)* Case of direct login (i.e. non-OpenID)? examine also the following overloaded functions: *boolean userLogin(int userID, String userName, String password)* *boolean userLogout(int userID, String userName, /*string password*/)*
Types description	None.
Exceptions	Bad password Bad/non-existing userName Logged-in user attempts to login (e.g. from a different connection)

Table 3 WP1_LogIn

Name:	**WP1_ManageItem**
Description:	Deals with the initial upload of information into WKI. It is concerned with uploading and deleting items like photo photographs, videos, and documents in the WKI knowledge base.
Usage Example:	A user captures a photograph, a video or a (plain text , or html-rich) document and wants to upload it to the KB. S/he may also state the permission level of the item to be uploaded: public, private, or shared amongst signified communities. A user may then want to remove the item by deleting it.
Method signature:	*int uploadItem(URI path, Permission perm)* - Returns itemID; negative values could be reserved for exception error messages *boolean deleteItem(int itemId)* - Returns true if successful
Types description:	enum Permission { public, private, communityShared};

Exceptions:	itemToobig Corrupted file uploaded (if error detection schemes be implemented)

Table 4 WP1_ManageItem

Name:	**WP1_Tag**
Description:	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.
Usage Example:	The user attaches a list of tags which describes the content to be uploaded. Preferably, the application may suggest tags based on the media analysis of the content, and/or on an ontology model (see WP2->Text Annotation Tool; WP2->VisualAnalysis) A user having access to an item can edit or delete tags.
Method signature:	*boolean addTag(int itemId, Array<String> tags)* - Returns true if successful *Array<String> GetItemTags(int ItemId)* - retrieve tags array attached to item *boolean modifyTag(int itemId, int tagId, String tags)* *boolean deleteTag(int itemId, int tagId)*
Types description:	Tags can be a vector of simple String tag and
Exceptions:	No permission -> Insufficient permission rights for the requested action Bad itemID Bad tagId

Table 5 WP1_Tag

Name:	**WP1_Comment**
Description:	This is another major data entry point in WKI. 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.
Usage Example:	The user attaches a comment which describes the content to be uploaded. A user having access to an item can edit or delete comments.
Method sig-	*int addComment(int itemId, String comment)* - returns

nature:	an ID number of the attached comment *boolean modifyComment(int itemId, int commentId, String comment)* *boolean deleteComment(int itemId, int commentId)*
Types description:	None.
Exceptions:	No permission -> Insufficient permission rights for the requested action Bad itemId Bad commentId

Table 6 WP1_Comment

Name:	**WP1_Rate**
Description:	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 have previously uploaded.
Usage:	The user attaches a rating which describes the content to be uploaded. Rating can take place during upload or later. If two users add default rating to an item an average value is calculated. A user having access to an item can edit or delete ratings. Default values are pre-defined like 1-2 -3 -4 -5 managed by Wp1_Rating_manager (internal service)
Method signature:	*int boolean addRating(int itemId, float rating)* *boolean modifyRating(int itemId, int ratingId, float rating)*
Types description:	Rating is between 0 and 1
Exceptions:	No permission -> Insufficient permission rights for the requested action Bad ratingId Bad itemId

Table 7 WP1_Rate

Name:	**WP1_SearchKB**
Description:	The access of the KB Data Storage based on the user-specified queries; the KB outputs a (ranked) list of re-

	sults, by exploiting all levels of intelligence (WP2-WP5)
Usage:	The user can filter the resources of the KB, according to the user and task profiling. The system should suggest querying keywords (semantic-search) based on initial search. Further, the system should preferably support the free posing of questions by the users.
Method signature:	*Array<String> searchQuery(int userId, Array<String> query_keywords)* - returns an array of suggested querying keywords (in a ranked order according to relevance/closeness)
Types description:	None.
Exceptions:	Bad query

Table 8 WP1_SearchKB

Name:	**WP1_UsersMessaging**
Description:	The possibility of messaging exchange amongst users. This function should enable the integrated mechanism of an instant messaging facility.
Usage:	A user should have the possibility to instantly communicate to one or more selected users or groups of users. S/he can indicate whether this messaging is private, or public (i.e. a posting). In the latter case, this communication can be used for text analysis by WP2. The messaging should support html language in order to exchange/share any format of content item. A message can have attachments.
Method signature:	*int sendMessage(int userId, Array<int> recipientIds, String content, Array<URI> attchment, Permission perm)* *boolean modifyMessage(int userId, int messageId, String content, Array<URI> attchment)* *boolean deleteMessage(int messageId, int userId)*
Types description:	recipientIds can be a vector of simple recipientId content can be simple text or HTML formatted text Permission as declared in Manage Items.
Exceptions:	BadContent Message too long Bad userId Bad recipientId

Table 9 WP1_UsersMessaging

9.2. WP2 services

WP2 services aim to provide functionalities for users to upload and access multimedia content within the WeKnowIt system framework. Services inside WP2 may be divided into three groups, according to the affected modality: textual, visual and speech services.

Textual services include tools for text categorization to predefined categories based on their similarity, tools for text clustering to help the user focus on similar documents, as well as automatic annotation tools to aid the multimedia content annotation task. They are applicable and will significantly aid both WeKnowIt Use Cases, since they will provide meaningful information at a given time.

A visual search service will be provided structured on top of visual similarity matching, retrieval and localization functionalities. Using this particular service, WeKnowIt end-users will be able to efficiently search and retrieve information about specific landmarks or points of interest. This will be particularly useful in both the Emergency Response and Consumer Use Cases, as it will allow to quickly retrieve relevant content.

Finally, speech services will provide the functionalities for users to establish a vocal communication with the WeKnowIt system by providing them speech indexing and searching functionalities. In particular in the Emergency Response Use Case, speech services could be adopted to enquire and retrieve critical information about the status of end users or the actual status of an event.

Text Analysis Tool

Although the input to the Text Analysis Tool (Services) is a set of texts, these could also be accompanied by metadata which could be utilised in the analysis process (such as user (author) profile information).

Note that these tools do not relate to the Tools for generating the (classification, clustering, annotation) models. These models are generated offline. The Classification Categories and Annotation Types are determined a priori, using domain specific information and resources, for example from tags and texts provided by WP1/7. The services described here provide access to these models so that they can be utilised by users or other services.

Note that the modelling tools will build models on specific language texts.

Name:	WP2_Text_Classification
Description:	To determine the similarity between a given text (set of texts) and some pre-defined language models relating to different categories. The predefined language model is created as part of

	the off-line 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.
Usage:	ER - does the text relate to flooding/non-flooding, or fire/non-fire Consumer - does the text relate to hotel? Is it a review or description? Is it a positive/negative review (i.e. sentiment analysis)
Method signature:	void addClassificationModels(List<TextClassificationModel> models) - adds the **List** of **TextClassificationModel** models to the classifier. Note that if two models exist with the same identifier then the earlier model is overwritten. • *models - **List** of **TextClassificationModel** models to add* List<TextClassificationModel> getClassificationModels() - gets the **List** of all the **TextClassificationModel** models which are available in the classifier. • *return - **List** of the available **TextClassificationModel** models.* TextClassificationModel getClassificationModel(String modelId) - Gets the **TextClassificationModel** specified by the model identifier. • *modelId - the **TextClassificationModel** identifier* • *return - **TextClassificationModel** specified but the model identifier;* **null** if no model exists with the specified identifier. List<String> getClassificationModelIdentifiers() - gets the **List** of the **String** identifiers of the **TextClassificationModel** models which are available in the classifier. • *return - **List** of the identifiers of the available **TextClassificationModel** models.* List<Classification> classifyText(String text, String modelId) - classify the text using the specified **TextClassificationModel** model.

- *text - the text to classify.*
- *modelId - **String** identifier of the **TextClassificationModel** model.*
- *return - a **List** of Classification objects one for each category. Note that if a classification* thresold is set within the Classification model this list may be a subset of the possible categories, including an empty list.

List<Classification> classifyDocument(Document doc, String modelId) - classify the **Document** using the specified **TextClassificationModel** model.

- *doc - the **Document** to classify.*
- *modelId - **String** identifier of the **TextClassificationModel** model.*
- *return - a **List** of Classification objects one for each category. Note that if a classification* thresold is set within the Classification model this list may be a subset of the possible categories, including an empty list.

List<List<Classification>> classifyDocuments(List<Document> docs, String modelId) - classify the **List** of **Document** using the specified **TextClassificationModel** model.

- *docs - a **List** of **Document** to classify.*
- *modelId - **String** identifier of the **TextClassificationModel** model.*
- *return - a **List** (one for each classified document) of **List** of Classification objects* one for each category. Note that if a classification thresold is set within the Classification model this list may be a subset of the possible categories, including an empty list.
- *return - **List** of arrays of values [0,1] indicating the degree to which each **Document** in* the **List** of is similar to the model's categories.

List<Classification> classify(String docId, String modelId) - classify the **Document**, specified by the identifier, using the specified **TextClassificationModel** model.

- *docId - the identifier of the **Document** to classify.*
- *modelId - **String** identifier of the **TextClassifi-***

	cationModel *model.* • *return - a* ***List*** *of Classification objects one for each category. Note that if a classification* thresold is set within the Classification model this list may be a subset of the possible categories, including an empty list. List<List<Classification>> classifyAll(List<String> docIds, String modelId) - classify the **List** of **Document**, specified by the identifiers, using the specified **TextClassificationModel** model. • *docIds - a* ***List*** *of identifiers of* ***Document*** *to classify.* • *modelId -* ***String*** *identifier of the* ***TextClassificationModel*** *model.* • *return - a* ***List*** *(one for each classified document) of* ***List*** *of Classification objects* one for each category. Note that if a classification thresold is set within the Classification model this list may be a subset of the possible categories, including an empty list.
Types description:	TextClassificationModel The classification model parameter indicates the model to use in classifying the text, i.e. defines the possible categories against which the text is compared. Document The common WKI Document object. Classification The returned classification String category: A URI identifying the category in the model float certainty: A value [0,1] indicating the degree to which the classified text is similar to the categories.
Exceptions:	IOException - if one of the models fails to parse or a **TextClassificationModel** with the specified identifier is not available in the classifier.

Table 10 WP2_Text_Classification

Name:	**WP2_Text_Clustering**

Description:	To cluster a set of texts into related clusters, this can be used to help the user to focus on similar documents. Text are clustered according to there content, the clustering process applies NLP techniques to normalise the text and then groups text containing similar terms into clusters. Therefore a cluster can be seen as the set of (weighed) terms, derived form the texts which are members of the cluster, which discriminate that cluster from the others.
Usage:	ER - combine a set of social network posts which are closely related
Method signature:	void addClusteringModels(List<TextClusteringModel> models) - adds the **List** of **TextClusteringModel** models to the classifier. Note that if two models exist with the same identifier then the earlier model is overwritten. • models - **List** of **TextClusteringModel** models to add List<TextClusteringModel> getClusteringModels() - gets the **List** of all the **TextClusteringModel** models which are available in the classifier. • return - **List** of the available **TextClusteringModel** models. TextClusteringModel getClusteringModel(String modelId) - gets the **TextClusteringModel** specified by the model identifier. • modelId - the **TextClusteringModel** identifier • return - **TextClusteringModel** specified but the model identifier; **null** if no modelexists with the specified identifier. List<String> getClusteringModelIdentifiers() - gets the **List** of the **String** identifiers of the **TextClusteringModel** models which are available in the classifier. • return - **List** of the identifiers of the available **TextClusteringModel** models. List<List<String>> clusterText(List<String> text, String modelId) - cluster the text using the specified **TextClusteringModel** model. • text - the text to cluster. • modelId - **String** identifier of the **TextCluster-**

	ingModel model. • return - a **List** (one for each cluster) of **List** of containing the text strings contained in each cluster. List<Cluster> clusterDocuments(DocumentCollectionType docCollection, String modelId) - cluster the **DocumentCollection** using the specified **TextClusteringModel** model. • docCollection - the **DocumentCollection** to cluster. • modelId - **String** identifier of the **TextClusteringModel** model. • return - a **List** of Clusters List<Cluster> clusterDocuments(List<Document> docs, String modelId) - cluster the **List** of **Document** using the specified **TextClusteringModel** model. • docs - a **List** of **Document** to cluster. • modelId - **String** identifier of the **TextClusteringModel** model. • return - a **List** of Clusters List<Cluster> cluster(String docCollectionId, String modelId) - cluster the **DocumentCollection**, specified by the identifier, using the specified **TextClusteringModel** model. • docCollectionId - the identifier of the **DocumentCollection** to cluster. • modelId - **String** identifier of the **TextClusteringModel** model. • return - a **List** of Clusters List<Cluster> clusterAll(List<String> docCollectionIds, String modelId) - cluster the **List** of **DocumentCollection**, specified by the identifiers, using the specified **TextClusteringModel** model. • docCollectionIds - a **List** of identifiers of **DocumentCollection** to cluster. • modelId - **String** identifier of the **TextClusteringModel** model. • return - a **List** of Clusters
Types description:	TextClusteringModel The clustering model parameter indicates the model to

	use in cluster the text, i.e. methods, thresholds, etc. Document The common WKI Document object. Cluster The returned clusters, where each cluster contains: * List<String> docURIs: A List of Strings identifying the documents in the cluster * List<float> certainty: A value [0,1] indicating the degree to which the document belongs to the cluster.
Exceptions:	IOException if a **TextClusteringModel** with the specified identifier is not available in the classifier or if one of the models fails to parse.

Table 11 WP2_Text_Clustering

Name:	**WP2_Text_Annotation**
Description:	Automatically annotates entities in the text, using some predefined annotation models. For each entity type (e.g. geospatial, temporal) an annotation model is developed (off-line). These entity models are then applied to text to recognise and annotate entities, adding this information to the text's metadata.
Usage:	E.g. To geo-tag the text so that they can be mapped.
Method signature:	void addAnnotationModels(List<TextAnnotationModel> models) - adds the **List** of **TextAnnotationModel** models to the classifier. Note that if two models exist with the same identifier then the earlier model is overwritten. • models - **List** of **TextAnnotationModel** models to add List<TextAnnotationModel> getAnnotationModels() - gets the **List** of all the **TextAnnotationModel** models which are available in the classifier. • return - **List** of the available **TextAnnotationModel** models. TextAnnotationModel getAnnotationModel(String modelId) - gets the **TextAnnotationModel** specified by the model identifier. • modelId - the **TextAnnotationModel** identifier

	• return - **TextAnnotationModel** specified but the model identifier; **null** if no model exists with the specified identifier. List<String> getAnnotationModelIdentifiers() - gets the **List** of the **String** identifiers of the **TextAnnotationModel** models which are available in the classifier. • return - **List** of the identifiers of the available **TextAnnotationModel** models. List<Annotation> annotateText(String text, String modelId) - annotate the text using the specified **TextAnnotationModel** model. • text - the text to annotate. • modelId - **String** identifier of the **TextAnnotationModel** model. • return - a **List** of Annotations. List<Annotation> annotateDocument(Document doc, String modelId) - annotate the **Document** using the specified **TextAnnotationModel** model. • doc - the **Document** to annotate. • modelId - **String** identifier of the **TextAnnotationModel** model. • return - a **List** of Annotations. List<List<Annotation>> annotateDocuments(List<Document> docs, String modelId) - annotate the **List** of **Document** using the specified **TextAnnotationModel** model. • docs - a **List** of **Document** to annotate. • modelId - **String** identifier of the **TextAnnotationModel** model. • return - **List** (one for each document> of **List** of Annotations. List<Annotation> annotate(String docId, String modelId) - annotate the **Document**, specified by the identifier, using the specified **TextAnnotationModel** model. • docId - the identifier of the **Document** to annotate. • modelId - **String** identifier of the **TextAnnotationModel** model. • return - a **List** of Annotations.

	List<List<Annotation>> annotateAll(List<String> docIds, String modelId) - annotate the **List** of **Document**, specified by the identifiers, using the specified **TextAnnotationModel** model. • docIds - a **List** of identifiers of **Document** to annotate. • modelId - **String** identifier of the **TextAnnotationModel** model. • return - **List** (one for each document> of **List** of Annotations.
Types description:	TextAnnotationModel The annotation model parameter indicates the model to use in annotate the text, i.e.annotation types, data used, etc. Document The common WKI Document object. Annotation The returned annotations, where each annotation contains: * String method: The method used to create the annotation * Statement: the annotation Statement contains: ** Description: a URI of the annotiation, which indicates type and value (e.g. geo-spatial, temporal) ** Triple (Subject, Predicate, Object) (Optionally): Indicates a relationship annotation
Exceptions:	java.io.IOException if one of the models fails to parse or if a **TextAnnotationModel** with the specified identifier is not available in the classifier.

Table 12 WP2_Text_Annotation

Name:	**WP2_VisualAnalysis**
Description:	Exploitation of visual content within WKI system in terms of visual similarity matching, retrieval and localization. WeKnowIt users will be able to efficiently search and retrieve information about specific landmarks or points of interest during their travel or an emergency event, as well as facilitate 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. WP1, WP3, WP6 and WP7 are expected to be directly influenced by these services.
Usage:	Potential benefits from its usage will be evident to all above WKI Workpackages, as well as to WKI end-users. E.g. a WKI user uploads his/her photos to the system and subsequently seeks additional content, according to his/her own preferences, as well as the preferences and relevant (i.e. similar) content of his/her friends. Using a large database of geo-tagged images the tool will match the given query and return a ranked list of images according to their visual (and potentially also their social) similarity. The geo-tags of the returned images will be used to provide an estimate of the location of the query photo and localize it using Google Maps™.
Method signature:	public ViralType visualSearch(String imageURI) throws URINotFoundException - returns a ViralType with the results of the query • imageURI - the URI of the query image public ViralType visualSearch(InputStream image) throws URINotFoundException - returns a ViralType with the results of the query • image - an InputStream to the image, as returned by the wki data storage (or any other source) public ViralType visualSearch(String imageURI, boolean external) throws URINotFoundException - return a ViralType with the results of the query • imageURI - the uri of the query image, either local or remote • external - declares whether the image is from the wki data storage or (if true) from an external source. In the case of external images, a prefix of the protocol required to get the image is requred: i.e. '**file:///**' for local files and 'http://' for remote ones. public double visualSimilarity(String imageURI1, String imageURI2) - visually compares two images, and re-

	turns the similarity between them (normalized in the 0-1 scale) • imageURI1 - the (data storage) URI of the first image • imageURI2 - the (data storage) URI of the second image • return - the normalized similarity between the two images public double visualSimilarity(InputStream imageStream1, InputStream imageStream2) - visually compares two images, and returns the similarity between them (normalized in the 0-1 scale) • imageStream1 - an InputStream to the first image to be compared • imageStream2 - an InputStream to the second image to be compared
Types description:	• **ViralType** - This type contains the following information: images (List<Image> images), image tags (List<String> imageTags) and estimated image location (GeoTag location). • ***GeoTag*** - This type contains the following information: longitude (double longitude) and latitude (double latitude). • ***Image*** - This type contains information about an image like its uri (String uri), its location (GeoTag location), the similarity with the query image (double similarity), and the title and description of the image (String title, String description).
Exceptions:	

Table 13 WP2_VisualAnalysis

Name:	**WP2_SpeechIndexing**
Description:	The service prepares an index for the search in the 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 (if a file represented by an URI from the list is not accessible, URINotFoundException is raised). The service also checks whether the recording file type is supported (if it is not, FileTypeNotSupportedException is raised).

	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. The service reads the content of the provided speech files then. If it is not compatible with the parameters given in settings or it does not correspond to the specified file format, ParameterMismatchException or FileTypeMismatchException is generated respectively. Based on the given settings, the service can retrieve any relevant metadata to improve the indexing process. The speech data is transformed and indexed finally. The service generates a unique URI for the index and returns it as a result. If any problem appears during the indexing, ServiceInternalErrorException is raised.
Required services:	WP6_DataStorage
Usage example:	**ER**: Process speech data and prepare indexes for subsequent searches. The service can be invoked manually but the automatic processing will be probably more frequent. It is expected that this service will be automatically called through WKI workflow management system whenever a new speech file appears. A service linked to this workflow should also take care of storing the resulting speechindexURI together with necessary metadata acquired by other parts of the WKI system (the time recordings corresponds to, identification of persons speaking (if available) and any other contextual information).
Method signature:	***String indexSpeech(List<String> speechfileURIs, ParameterSpecification settings)*** - based on settings - service-specific parameters of the indexing process - the service prepares an index from the set of recordings given by speechfileURIs and returns speechindexURI of the indexed data which can be used in the calls of the related SearchInSpeech service.
Types description:	**ParameterSpecification** - a structured format defining values of attributes that specify the settings under which indexing should operate.
Exceptions:	InsufficientPermissionException - user does not have the right to perform this operation SpeechFileURIListEmptyException - the parameter is

	null or empty SpeechFileURINotFoundException - the file corresponding to one of the URIs provided in the input list cannot be read (the particular URI is also reported) FileTypeNotSupportedException - processing of the given type is not supported (the particular URI and the identified file type are reported) FileTypeMismatchException - the content of the file does not correspond to the specified format (the particular URI and the problem encountered are reported) ParameterMismatchException - a parameter given by settings conflicts with the file content (the particular speechfileURI and the problem encountered are reported) ServiceInternalErrorException - any other error (that usually needs an action by the service administrator, e.g., lack of the disk space to store the generated indexes) if any of the above exception occurs, the speech index will not be created

Table 14 WP2_SpeechIndexing

Name:	**WP2_TagNormalization**
Description:	The list of 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. 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 points of interest identification. WP1, WP3, WP6 and WP7 are expected to be directly influenced by this service.
Required services:	
Usage ex-	Potential benefits from its usage will be evident to all

ample:	above WKI Workpackages, as well as to WKI end-users. The service is used internally by the system to add formal descriptions to uploaded content. More specifically, when the tags added by the user are passed to the KB, the intelligent module will analyse the tags and, if matching a concept in the domain ontology, will automatically add the annotation to the metadata; otherwise they will just be added as tags. The user-assigned tags of a resource are given as input to this service, and a list of concepts, along with a list of weights, are returned as a formal description for that resource. The suggested weights are used to provide an estimate of the matching between the assigned tags and the proposed concepts.
Method signature:	Note: this is just a preliminary tentative list of methods that are subject to change! • ***Map<String, Double> normalizeTags(Vector<String> tags)*** - Normalizes tags to relate to the domain ontology
Types description:	
Exceptions:	

Table 15 WP2_TagNormalization

Name:	**WP2_TagProcessing**
Description:	Exploitation of textual tags derived from visual content within WKI system in terms of tag matching and similarity. By utilizing the additional textual information, WeKnowIt users will be able to efficiently enhance visual retrieval and localization task about specific landmarks or points of interest during their travel or an emergency event. More specifically, The list of suggested tags for the specific image are paired with known landmarks close by the location the image was taken (as listed in the Geonames 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. For the appropriate mapping between tags and formal descriptions, external sources of knowledge (such as WordNet, Wikipedia, etc) will be exploited. Furthermore, string processing, matching and comparison functionalities will enhance the results of the provided visual analysis in terms of landmarks or points of

	interest identification. WP1, WP3, WP6 and WP7 are expected to be directly influenced by these services.
Required services:	
Usage example:	Potential benefits from its usage will be evident to all above WKI Workpackages, as well as to WKI end-users. The service is used internally by the system to add tectual tag to an image, after a visualSearch (similarity search). The most common tags of similar images are given as input to this service along with the estimated location, and a tag (or a list of tags) is returned as a recommendation for that image. The suggested tags will be used to provide an estimate of the textual description of each photo.
Method signature:	List<String> filterTags(GeoTag location, List<String> suggestedTags) - The list of suggested tags for the specific image are paired with known landmarks close by the location the image was taken (as returned by the Geoplanet service), or discarded. The name of landmarks that persist through the pairing process, returned to the system/user. • Location - the location used for filtering the tags • suggestedTags - the suggested tags to be filtered • return - a List of strings, containing the matched landmark names near the given location. List<Landmark> getLandmarks(ViralType vt) - the list of suggested tags for the specific image are paired with known landmarks close by the location the image was taken (as returned by the Geoplanet service), or discarded. The list of landmarks that persist through the pairing process are returned to the system/user. • Vt - the ViralType containing the results of the visual-analysis service • return - a List of possible landmarks, along with their location and wikipedia links
Types description:	In addition to the ones from wp2-visual-analysis, we have: • **Landmark** - An object containing all needed information about a landmark. Among others, it includes its name (String name), its wikipedia id (int wikiID), its

	wikipedia link (String link) and its city (String city).
Exceptions:	

Table 16 WP2_TagProcessing

Name:	**WP2_SemanticPhotoQuery**
Description:	Retrieving Flickr photos that best match the domain ontology concepts for specific time intervals.
Required services:	ER - An emergency situation is reported in a specific area. The service downloads photos from Flickr around this specific area and ranks them based on their tags' matching with the ER-domain ontology concepts (i.e. the more relevant a photo's tags with the ER-concepts are, the higher rank the photo gets). Then, the photos that are ranked above a certain threshold are returned to the user.
Usage example:	public ArrayList<Photo> getGeoLocatedSemanticPhotos(double latitude, double longitude, Date beginDate, Date endDate) - getGeoLocatedSemanticPhotos returns a sorted list of photos, which are geolocated in the given latitude and longitude and which are taken between the dates begin and end
Method signature:	***Photo*** - A Flickr image object, along with its metadata. For more information see: **http://flickrj.sourceforge.net/api/**
Types description:	
Exceptions:	

Table 17 WP2_SemanticPhotoQuery

9.3. WP3 services

Services where originally designed to address specific user needs in the question answering system. Main issues include finding relevant expertise among users for a given document content, measuring information quality, and providing better annotation of resources.

We provide five services: local community detection based on tags, spam detection, latent topic detection and document categorization, topic-expert finder, and service for measuring answer quality.

Local community detection service allows providing of better annotation for resources by providing a network of related and relevant tags. In ER scenario it can be directly applied to improve annotation of uploaded content. Expert finder service provides a list of users that have expertise in

areas defined by given content. Experts are selected based on previously detected latent topics and groups in the system. For CSG scenario, information about expert users in specific topic or location helps to verify trustworthiness of provided information. Detecting of latent topics in provided document allows to find areas covered within document, and later use this information to get other related documents from the system. Services for information quality include spam detection and measuring relevancy of an answer with regard to provided question. Measures for quality of information are directly applicable in ER (e.g. for detecting irrelevant inputs to current situation or spam), or in CSG (e.g. for ranking reviews)

Name:	**WP3_LexicalSpamDetection**
Description:	In the context of a Questions & Answers (QA) system, this service can flag an answer to a question as spam.
Required services:	
Usage example:	The service is intended for use in the context of filtering user contributed questions before presenting them to the interface or before another service requests them for processing. At the moment, the service operates on simple text fragments, but in the future it is planned to operate on Question-Answer pairs as well.
Method signature:	**SpamType** ***isLexicalSpam***(**String** textFragment) - Checks whether the input text fragment is spam from a superficial text point of view (output semantics: SPAM -> Lexical spam with high confidence, NO_SPAM -> Not lexical spam with high confidence, SPAM_CANDIDATE -> Potentially spam, but human judgement would be necessary to reach the final decision)
Types description:	SpamType: enumeration {NO_SPAM, SPAM, SPAM_CANDIDATE}
Exceptions:	No exceptions identified so far.

Table 18 WP3_LexicalSpamDetection

Name:	**WP3_LocalTagCommunityDetector**
Description:	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 whatever we want to support). Based on the tag co-occurrences we can create a network of tags (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, it is possible to extract groups of tags that are closely connected with each other and less connected with the rest of the network. In our problem, that would correspond to identifying topics in the network. 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.
Required services:	
Usage example:	GUI components, Tag recommendation, Trend detection service. E.g. suppose we want to recommend a set of tags to user for tagging a piece of text (e.g. answer) that he/she has submitted to the system. Then, we would extract the keywords from this text and call this service to come up with a set of related tags for the user. For instance, starting from a seed tag "caribbean" will return a tag community (set of tags) containing this tag, e.g. islands, corals, palm trees, jamaica, trinidad, etc. Obviously, the quality and relevance of the returned community is dependent (among other factors) on the tagging behaviour of the users, i.e. on the underlying tag network.
Method signature:	• **List<String>** ***getAvailableTagDatasets*****()** - returns a list of available datasets to use when asking for tag recommendation (see next methods) • **List<String>** ***getRelatedTags***(**String** tag, **String** tagDataset, **int** maxNrOfReturnedResults) - returns a list of tags that are recommended by the system based on the input *tag* using the dataset *tagDataset* (which should belong to the list of datasets returned by the first method); a maximum number of *maxNrOfReturnedResults* tags is returned • **List<String>** ***getRelatedTags***(**String** tag, **String** tagDataset) - same as previous, but uses a default value for *maxNrOfReturnedResults* • **List<String>** ***getRelatedTags***(**List<String>** tags, **String** tagDataset, **int maxNrOfReturnedResults**) - same as second method, but instead of a single tag, provides a list of tags as input • **List<String>** ***getRelatedTags***(**List<String>**

	tags, **String** tagDataset) - same as previous, but uses a default value for *maxNrOfReturnedResults*
Types description:	
Exceptions:	No exception has been identified yet.

Table 19 WP3_LocalTagCommunityDetector

Name:	**WP3_POIRecommender**
Description:	Given an input POI, the service comes up with suggestions for related POIs.
Usage:	The service is intended for use in the context of a travel application. Users plan their trip by use of the WKI-based tourist application. As soon as they navigate to a page w.r.t. a specific POI, the system identifies related POIs and provides users with the recommendations.
Method signature:	• **List<String>** ***getRelatedPois*****(String inputPoi, int maxNumberOfRecom)** - returns a list of maximum *maxNumberOfRecom* POIs as recommendation given the *inputPoi* • **List<String>** ***getRelatedPois***(**String** inputPoi) - returns a list of POIs as recommendation given the *inputPoi* using a default value for the maximum number of recommendations
Types description:	
Exceptions:	No exceptions identified so far.

Table 20 WP3_POIRecommender

9.4. WP4 services

WP4 provides services for two groups of problems: the administration of communities and the analysis of communities. The services for the community administration provide functionality to manage access rights. Access rights describe the which data and system functionality a user is allowed to access. The services for the community analysis provide functionality to analyze and visualize online communities on different levels of aggregation: on the user-level, on the level of subgroups and on the level of the complete community. For example, the services aim to provide help to estimate community importance or trustworthiness of community

members, find sub-communities and visualize the different layers of communities structures.

Name:	**WP4_Community_Design_Language**
Description:	General purpose of this service is to provide permission management system which will extend the functionality of the WKI System. The service calls the Community Administration Platform (CAP) 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
Required services:	only internal
Usage example:	
Method signature:	public String executeCdl (String cdlExpression);
Types description:	String literals
Exceptions:	WKISocialException, IdentifierException

Table 21 WP4_Community_Design_Language

9.5. WP5 services

The services of WP5 add to the content and the users represented in the WeKnowIt system additional well defined structures to support the users conducting their task. The services of WP5 are divided into three groups of services:

The services of Group Management allow for maintain and use distributed defined groups within the system. This enables the inclusion of members of other professional organizations or authorities but also of private organisations into the virtual organisation created to respond to the emergency. In the CSG use case the provided services allow for reuse of social structures defined in other contexts before.

Task Management services provide functionality to define tasks that can be given to users or groups of users to conduct in order to improve speed

and efficiency. These tasks consist of a textual description, but include services and data structures provided by other WPs, e.g. uploaded content or analysis results, that can be used for executing the task. Defining tasks explicitly in the system can improve the collaboration of the emergency responders.

Event-based Incident Log services and LogMerger services provide means to organise knowledge and content in terms of events within logs. The events span along multiple dimensions like time and location. Also functions are provided to summarize logs. For emergency personnel in the ER use-case an event log is a tool to integrate all incident relevant information along multiple dimensions. Users in the CSG use-case can use the log to integrate their experiences.

Name:	**WP5_GroupManagement**
Description: General purpose, type, benefit and approach	General Description: Defining, modifying, and deleting organisations and groups, i.e., structures consisting of people and other groups. This groups are can be represented in a distributed manor. 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 D7.1). The overall goal of this service is to facilitate efficient task management in a virtual organisation for ER through the task manager service of WP5 (see below). Type: external service (communicates with other WPs) Interplay with other WPs: Individual users can create groups, assign themselves to groups, etc. (WP1) Cluster of users calculated in WP4 can be represented as a group (WP4) User with a high centrality calculated in WP4 can be represented as user in a group (WP4) Expert for a specific task calculated by WP3 can be represented as user in a group (WP3)
Method sig-	

nature:	• createGroup(URI adminID, name String, Enumeration{OPEN, INVITATION_NEEDED, SINGLE_ADMIN} adminMode) return URI groupID - create a new group • addAgentToGroup(URI agentID, URI groupID) return boolean - adds an agent (user or group) to the group, returns true if successful • removeAgentFromGroup(URI agentID, URI groupID) return boolean - removes the agent (user or group) from the group, returns true if successful • getMembers(URI groupID) return Set<URI> - returns users (without administrators) of the group • getAdministrators(URI groupID) return Set<URI> - returns the administrators of the group • getGroupProfile(URI groupID) return String profile - returns description of the group • setGroupProfile(URI groupID , String profile) - sets description of the group
Types description:	URI = uniform resource identifier
Exceptions:	• all: groupID does not exist • all: insufficient rights to perform this action • createGroup: group with name already exists • addUserToGroup: user is already in the group • removeUserFromGroup: user in not in the group • removeGroupFromGroup: group in not in the group

Table 22 WP5_GroupManagement

Name:	**WP5_LogMerger**
Description:	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 ser-

	vice 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 will provide methods for merging the logs and ordering the respective log entries (based on time) as well as index the log entries, so that they can be searched based on keyword, person, location, ER function, and action.
Required services:	WP2 -> List<String> getLocations(String text): Get list of terms in the input text that denote locations. WP2 -> List<String> getPersons(String text): Get list of terms in the input text that denote persons. WP2 -> List<String> getVerbs(String text): Get list of terms in the input text that denote verbs/actions.
Usage example:	This service is specific to the ER scenario. According to SCC, multiple logs are created during an incident by different organizational users (e.g. person at the control room, FLO, etc.). When reviewing the incident, it would be helpful to have a common log. When all log files are collected, it is possible to call this service in order to merge the logs and then search the merged log for ER-specific entities.
Method signature:	*Log mergeLogs(List<String> logFiles)* - Creates a merged log out of a set of log files (serialized in a predefined word-xml template used by SCC). *List<LogEntry> keywordSearch(String logName, String[] keywords)* - Free-text search in the log. All keywords should be present in a log entry in order for it to be returned. *List<LogEntry> roleSearch(String logName, String[] roleNames)* - Searches for the given roles (ER abbreviations, e.g. FLO) in the input log entries. *List<LogEntry> locationSearch(String logName, String[] locations)* - Searches for the given locations in the input log entries. *List<LogEntry> actionSearch(String logName, String[] actions)* - Searches for the given actions (e.g. notified) in the input log entries.
Types description:	Log (String name, List<LogEntry> entries) LogEntry (DateTime, String fromTo, String message,

	String action) (standard log format used by SCC)
Exceptions:	Unable to open input log file: When the service cannot open for reading one or more of the input logFiles. Incorrect log file format: When the service cannot parse correctly one of the input log files.

Table 23 WP5_LogMerger

9.6. WP6 services

WP6 services provides access to data saved in the WKI Data Storage. More detailed description of WP6 services is included in D6.3.

Name:	**WP6_DataStorage**
Description:	The service provides API for storing/retrieving data from the WKI Data Storage. This API covers all components of the WKI Data Storage - triple store(s) and other databases, and file storages. Provided functionalities can be divided into few groups: • model - 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 apply some order to the data model and also can be used to restrict the area of search queries. Each model is uniquely identified by its URI. • common objects - The WKI DataStorage API is designed to interact with the rest of the WKI System mainly through objects from the common model . This approach ensures that the WKI DataStorage is independent from any of the underlying ontologies and can be easily extended to use new sets of them. This is the preferred way of communication with the WKI DataStorage. • raw triples - allows to store whole graphs (sets of triples) into models and retrieve all data from specified models. This part of the API was added on request of the partners, however it is strongly recommended to use methods from 'common object' part instead, because of efficiency reasons and data exchange policy. • file - allows to store and retrieve binary data (files) into the WKI DataStorage • query - various query and search methods
Usage:	This service should be used by other services to store

	different types of data as well as to search for data fulfilling specific requirements.
Required Services:	None
Method signature:	String executeQuery(String modelUri, String sparqlQuery) throws WkiDataStorageException - executes sparql query on KB and returns result as XML. • modelUri - model name where data should be searched • sparqlQuery – query • return - result as XML String executeQuery(String sparqlQuery) throws WkiDataStorageException - executes sparql query in default model on KB and returns result as XML. • sparqlQuery – query • return - result as XML String executeQueryJson(String modelUri, String sparqlQuery) throws WkiDataStorageException - executes sparql query in default model on KB and returns result as JSON (see http://www.w3.org/TR/rdf-sparql-json-res). Only specified model will be used to search data. • modelUri - model name where data should be searched • sparqlQuery – query • return - result as JSON String executeQueryJson(String sparqlQuery) throws WkiDataStorageException - Executes sparql query in default model on KB and returns result as JSON (see **http://www.w3.org/TR/rdf-sparql-json-res**). • sparqlQuery – query • return - result as JSON String storeCommonObject(String modelUri, IWkiCommonObject commonObj) throws WkiDataStorageException - Add new common object to given model. ID will be generated automatically. • modelUri - model URI • commonObj – object from common model to store

	• returns – ID of stored object String storeCommonObject(IWkiCommonObject commonObj) throws WkiDataStorageException - Add new common object to default model. ID will be generated automatically. • commonObj – object from common model to store • returns – ID of stored object IWkiCommonObject getCommonObject(String modelUri, Class<? extends IWkiCommonObject> type, String id) throws WkiDataStorageException - Return common object for given type and ID from specified model. Type must be one of IWkiCommonObject types. Returns null if object with given type and ID doesn't exist. Throws exception if this model doesn't exist. • modelUri – model URI • type - object type • id - object ID • return - IWkiCommonObject if exist or returns null if object with given type and ID doesn't exist. IWkiCommonObject getCommonObject(Class<? extends IWkiCommonObject> type, String id) throws WkiDataStorageException - Return common object for given type and ID from default model. Type must be one of IWkiCommonObject types. Returns null if object with given type and ID doesn't exist. Throws exception if this model doesn't exist. • type - object type • id - object ID • return - IWkiCommonObject if exist or returns null if object with given type and ID doesn't exist. void removeCommonObject(String modelUri, Class<? extends IWkiCommonObject> type, String id) throws WkiDataStorageException – Remove previously stored object form common model. • modelUri – model URI • type - object type • id - object ID void removeCommonObject(Class<? extends IWkiCommonObject> type, String id) throws WkiDataStora-

	geException - Remove previously stored object form common model. • type - object type • id - object ID void createModel(String modelUri) throws WkiDataStorageException - Creates new model in knowledgebase with specified URI. If model already exist then nothing is done. • modelUri - new model URI void startModel(String modelUri) throws WkiDataStorageException - Start given model. • modelUri - model to start void stopModel(String modelUri) throws WkiDataStorageException - Stop given model. • ModelUri – model to stop void storeData(String modelUri, InputStream data, WkiDataFormat dataFormat) throws WkiDataStorageException - Store data (set of triples - graph) in given model. Data is read from InputStream. void storeData(InputStream data, WkiDataFormat dataFormat) throws WkiDataStorageException - Store data (set of triples - graph) in default model. Data is read from InputStream. void getData(String modelUri, OutputStream outStream, WkiDataFormat dataFormat) throws WkiDataStorageException - Writes all data from given model to Output stream. void getData(OutputStream outStream, WkiDataFormat dataFormat) throws WkiDataStorageException - Writes all data from default model to Output stream. InputStream getFileContent(String contentUri) throws WkiDataStorageException - Returns content of file with given URI. • URI - file URI • return - content of file String storeFileContent(InputStream content, String contentID) throws WkiDataStorageException - Stores content of file for access by HTTP. ContentID should be unique and points to the content during access via HTTP request. Important! - files stored with this method will be accessible without security restrictions.

	This method was intended for FOAF profiles which should be freely available. Content stored through this method can also be retrieved using getFileContent and returned uri. • content - content of file • contentID - ID used to access content via http • return - URI of stored file accessible via http void updateFileContent(InputStream content, String contentID) throws WkiDataStorageException - Updates previously stored file content with given content id. Overrides all previously stored data. • Content - new content of file • contentID ID used to access content via http Iterator<IWkiCommonObject> getRecentCommonObjects(String modelUri, Class<? extends IWkiCommonObject> type, Date since, Integer limit) throws WkiDataStorageException - Returns iterator of common objects of given type added to given model or updated since given date. Method can returns less IDs then limit parameter specifies if it is not possible to retrieve given number of IDs, e.g. only two objects of a given type were added to IWkiDataStorage and limit parameter is set to 5. Limit parameter set to null tells that there will be no limitations in number of returned objects. • modelUri – model uri • type - type of common objects to look for • since - only common objects updated/added after this date will be returned • limit - maximum number of IDs to return, if null there will be no limitations in number of returned objects, all found objects will be returned • return - iterator of common objects Iterator<IWkiCommonObject> getRecentCommonObjects(String modelUri, Class<? extends IWkiCommonObject> type, Date since) throws WkiDataStorageException - Returns iterator of common objects of given type added to given model or updated since given date. • modelUri – model uri • type - type of common objects to look for • since - only common objects updated/added after

	this date will be returned • return - iterator of common objects Iterator<IWkiCommonObject> getRecentCommonObjects(Class<? extends IWkiCommonObject> type, Date since) throws WkiDataStorageException - Returns iterator of common objects of given type added or updated since given date in default model. • type - type of common objects to look for • since - only common objects updated/added after this date will be returned • return - iterator of common objects Iterator<IWkiCommonObject> getRecentCommonObjects(String modelUri, Class<? extends IWkiCommonObject> type, Integer limit) throws WkiDataStorageException - Returns iterator of common objects of given type recently added to given model or updated. The number of returned objects is specified by limit parameter. • modelUri – model uri • type - type of common objects to look for • limit - maximum number of IDs to return, if null there will be no limitations in number of returned objects, all found objects will be returned • return - iterator of common objects Iterator<IWkiCommonObject> getRecentCommonObjects(Class<? extends IWkiCommonObject> type, Integer limit) throws WkiDataStorageException - Returns iterator of common objects of given type recently added or updated in default model. The number of returned objects is specified by limit parameter. • modelUri – model uri • type - type of common objects to look for • limit - maximum number of IDs to return, if null there will be no limitations in number of returned objects, all found objects will be returned • return - iterator of common objects void updateCommonObject(String modelUri, IWkiCommonObject commonObj) throws WkiDataStorageException - Updates given common object from given model. • modelUri – model uri

	• commonObj – object from common model to update void updateCommonObject(IWkiCommonObject commonObj) throws WkiDataStorageException - Updates given common object from default model. • commonObj – object from common model to update <T extends IWkiCommonObject> Collection<T> findCommonObjects(String modelUri, Class<T> type, T objectPattern) throws WkiDataStorageException - Returns all found common objects which fulfill all conditions passed through object pattern. WARNING! Method returns only those objects which have all fields equal to corresponding non null fields from object pattern. Only non null fields from object pattern and their equivalents from stored objects are compared. Object pattern should have filled only those fields which should be used during searching. The rest of fields must have null values. Wki-ds currently handles fields with following types: all primitive types and corresponding wrapping objects (e.g. int and Integer), String and Date. <T extends IWkiCommonObject> Collection<T> findCommonObjects(Class<T> type, T objectPattern) throws WkiDataStorageException - Returns all found common objects from default model which fulfill all conditions passed through object pattern. WARNING! Method returns only those objects which have all fields equal to corresponding non null fields from object pattern. Only non null fields from object pattern and their equivalents from stored objects are compared. Object pattern should have filled only those fields which should be used during searching. The rest of fields must have null values. Wki-ds currently handles fields with following types: all primitive types and corresponding wrapping objects (e.g. int and Integer), String and Date. Set<ISearchResult> keywordSearch(String[] keywords) throws WkiDataStorageException - Keywords search in wki common objects. • keywords - query words • return - ISearchResult with result id and score URL getFileURL(String documentID) throws WkiDataStorageException - Makes resource connected with document with given ID accessible via http and returns URL to

	String storeFileContent(InputStream inputStream) throws WkiDataStorageException - Storing the file content. • inputStream - file content to store • return - Document id where the contentUri is stored String storeFileContent(String url) throws WkiDataStorageException - Storing the file content • url – url of the file to store • return - Document id where the contentUri is stored InputStream getDocumentContent(String documentId) throws WkiDataStorageException - Retrieving the file content • documentId – id of document pointing to secific content • return - inputStream the file content of specified document
Types description:	IWkiCommonObject – interface common for all objects from common model
Exceptions:	WkiDataStorageException - generic exception wrapper for all exceptions thrown.

Table 24 WP6_DataStorage

10. Appendix B – guidelines

This appendix contains some of the guidelines that were written by Wp6 in order to help developers of WKI services develop code compatible with the WKI architecture. The three presented guidelines might be helpful for installation and running of the WKI System prototype (Sections 5 and 6).

10.1. Guidelines - introduction

Even though technologies selected for the WKI System were examined in terms of ease-of-use (see D6.1.2 for technology evaluation process) it is clear that in order to facilitate the development of WKI services some sort of guidance for developers is required. WP6 has recognized this need in the beginning of the WKI project and started preparation of a set of guidelines that would help developers create components that would be compatible with the WKI System architecture taking some of the learning burden from their shoulders.

This way a set of guidelines was developed. These guidelines provide all the information needed to develop a service that is compatible with the WKI architecture – from information on the use of development infrastructure to detailed tutorial that presents how to deploy implemented services to the WKI System.

The full list of available guidelines is presented below. Titles of guidelines which were added since D6.2.1 are written using **bold font**. Guidelines that were updated since D6.2.1 have their **versions bolded**:

- Development Conventions (ver 0.1),
- How to install Java & Maven (ver **0.6**),
- How to install the WKI System (ver **0.11**),
- 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 **0.3**),
- **How to use Hudson CI** (ver 0.5),
- **SVN project layout** (ver 0.2)
- **How to run Berlin SPARQL Benchmark (BSBM) to test the WKI Data Storage** (ver 0.2)

All guidelines can be downloaded from the project main wiki page. Some of them (those related to installation and basic usage of the WKI System) are also included in Section 10.

Apart from the guidelines, there is also a *Development FAQ* available on main wiki. It's purpose is to deliver fast answers to the urgent questions

that arise during development. The questions and answers from this FAQ are gradually included into guidelines.

The list of guidelines is not closed, yet all the technologies are used by development teams for a long time, so it seems unlikely that completely new requests for guidelines will emerge. Nevertheless, continuous work on the WKI System architecture (i.e. upgrade of internal libraries) causes the need to update the existing guidelines so they reflect the current state of the system.

10.2. How to install Java and Maven

10.2.1. Introduction

Rationale

Java, Ant and Maven are the main tools used for development of WKI services. They must be installed and configured. This guideline describes this process.

Prerequisites

None known.

Outcome

Java, Ant and Maven installed and configured.

10.2.2. Installation of Java

If you have already Java JDK (Java Development Kit) installed (minimum version 1.6 is required) you can skip this section. To see what Java version is installed type:

```
java -version
```

Java JDK downloads can be found at **http://java.sun.com/javase/downloads/index.jsp**. Installation process is platform-dependent, so please make sure you follow the instructions suitable for you operating system (the installation instruction can be found on the same page - search for 'Installation Instructions' link). Please install latest version of Java JDK (at the time of writing of this guideline its '**Java SE Development Kit (JDK) 6 Update 12**').

Please follow ALL the instructions of the installation process as described there.

Make sure that JAVA_HOME property is set.

10.2.3. Installation of Ant

If you have already Ant 1.7.1 installed, you can skip this section. To see what version of Ant is installed type:

```
ant -version
```

Ant downloads and installation instructions can be found at **http://ant.apache.org/**. The installation process is platform-dependent, so

please make sure that you follow the instructions suitable for you operating system. Please install version **1.7.1** of Ant.

Please follow ALL the instructions of the installation process as described there.

10.2.4. Installation of Maven

If you have already Maven 2.0.9 installed you can skip this section (but you should read the next ones !). To see what version of Maven is installed type:

```
mvn -v
```

Maven downloads and installation instructions can be found at **http://maven.apache.org/download.html**. The installation process is platform-dependent, so please make sure that you follow the instructions suitable for you operating system. Please install version **2.0.9** of Maven.

Please follow ALL the instructions of the installation process as described there.

Local repository

When you build a project, Maven downloads JAR files from the WKI Maven repository (Nexus) to you local disk[19]. You need to specify where this downloaded libraries (JAR files) will be stored. You use `settings.xml` file to specify the location of your local repository.

IMPORTANT – it is forbidden to use folder with whitespaces for you local Maven repository. It will not work, and you will get very strange errors !!!

settings.xml file

Please put `settings.xml` file into your `.m2` folder. Depending on your operating system `.m2` folder will be placed in:

1. `c:\Documents and Settings\YOUR_USERNAME\.m2` on Windows XP,
2. `c:\User\YOUR_USERNAME\.m2` on Windows Vista,
3. `~/.m2` on Linux.

You need to edit this file. And this file should contain the following information:

```
<settings xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/settings-1.0.0.xsd">

    <!-- uncomment the line appropriate to you operating system-->
    <!-- windows example →
    <!-- please avoid folders with whitespaces !!! for example DON'T USE “Documents and Settings” or Maven will crash with cryptic error message -->
    <!-- <localRepository>c:\.m2\repository</localRepository> -->
```

[19] Please refer to D6.1.2 for further expalanations.

```
  <!-- linux example →
  <!-- <localRepository>/home/username/repo</localRepository> -->

 <mirrors>
  <mirror>
   <id>WKI</id>
   <name>Nexus WKI Mirror</name>
   <url>http://mvn.weknowit.eu/content/groups/public</url>
   <mirrorOf>central</mirrorOf>
  </mirror>
 </mirrors>
 <servers>
  <server>
   <id>WKI</id>
   <!-- put your credentials here -->
   <username>{your_username}</username>
   <password>{your_password}</password>
  </server>
 </servers>

</settings>
```

There are two parts of this file that need to be customized (**bolded** - see also comments in the file above):

- location of the local Maven repository:
    1. the content of this file is platform-dependent - make sure you (un)comment appropriate lines
- credentials for the WKI Maven repository:
    1. to receive your account credentials (login/pass), please ask **SMIND**.

Examples

Linux

Your `settings.xml` is in `~/.m2` folder, and you plan to keep you local Maven repository in `~/myLocalRepo` folder.

Your `settings.xml` should look like this:

```
<settings xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/settings-1.0.0.xsd">

 <localRepository>/home/MYUSERNAME/myLocalRepo</localRepository>

 <mirrors>
  <mirror>
   <id>WKI</id>
   <name>Nexus WKI Mirror</name>
```

```
    <url>http://mvn.weknowit.eu/content/groups/public</url>
    <mirrorOf>central</mirrorOf>
  </mirror>
 </mirrors>
 <servers>
  <server>
   <id>WKI</id>
   <username>John</username>
   <password>MySecretPassword</password>
  </server>
 </servers>

</settings>
```

Windows XP

Your `settings.xml` is in `c:\Documents and Settings\YOUR_USERNAME\.m2` folder, and you plan to keep you local Maven repository in `c:\myLocalRepo` folder.

Your `settings.xml` should look like this:

```
<settings xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <localRepository>c:\myLocalRepo</localRepository>

 <mirrors>
  <mirror>
   <id>WKI</id>
   <name>Nexus WKI Mirror</name>
   <url>http://mvn.weknowit.eu/content/groups/public</url>
   <mirrorOf>central</mirrorOf>
  </mirror>
 </mirrors>
 <servers>
  <server>
   <id>WKI</id>
   <username>John</username>
   <password>MySecretPassword</password>
  </server>
 </servers>

</settings>
```

10.2.5. Links

Further information can be found here:

- **http://java.sun.com** - Java homepage

- **http://ant.apache.org** - Ant homepage
- **http://maven.apache.org** - Maven homepage
- **http://maven.apache.org/ref/2.0.9/maven-settings/settings.html** - Maven, more information on settings.xml file
- **http://books.sonatype.com/maven-book/index.html** - Maven: The Definitive Guide
- **http://nexus.sonatype.org/** - Nexus homepage

10.3. How to install the WKI System

10.3.1. Introduction

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.

Prerequisites

- Java & Maven installed and configured - as it is described in 'How to install Java & Maven' guideline.
- Apache Ant 1.7.1 installed - available for download at **http://ant.apache.org**.

Please verify that proper version is installed using command:

```
ant -version
```

- Working network connection.

Outcome

The WKI System installed.

10.3.2. Installation of 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:

- `ki-X.Y.zip` for windows
- `ki-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.

Files and folders in WKI_HOME

This is how `WKI_HOME` looks like:

```
|-- ant
|-- bin
|-- configs
|-- data
|-- demos
|-- deploy
|-- etc
|-- fs-cr-repo
|-- http-resources
|-- 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:

- `in` - startup/shutdown scripts
- `onfigs` – 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
```

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

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 1.

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

Table 25: Log levels

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

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

PostgreSQL

PostgreSQL server 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.

1. Download PostgreSQL package appropriate for your operating system from **https://cr.weknowit.eu/wp4/postgres-install/**

1. 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.

2. Install PostgreSQL on your local machine (same as the WKI System) according to instructions described on **http://www.postgresql.org** page using packages downloaded in previous step.

A. 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)

Databases creation

After installation is completed, it is necessary to create required data structures. In order to do that following steps should be performed:

1. start PostgreSQL Server

- check if current user is able to connect to PostgreSQL Server using following command:

```
psql -U postgres postgres
```

It is essential for the rest of the process to successfully execute above command. If it can not be achieved with the current user please switch to the user with greater rights or expand the rights of the current user.

- go to `WKI_HOME/configs` directory
- run command:

```
ant
```

WARNING – this script cleans and rebuilds WKI databases completely. All data will be lost, and initial data structures will be restored.

Every time you want to have a clean WKI System with "default" data structures in database, simply run this script.

10.3.3. Starting the WKI System

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

- inux:

1. first make the startup script executable (you have to do this only once):

```
chmod 755 bin/start
chmod 755 bin/servicemix
```

1. then run the startup script

```
.bin/start
```

- indows

```
bin\start
```

WKI System is working now, to start the console run the client:

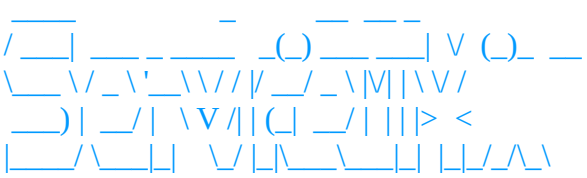
- inux

```
java -jar lib/karaf-client.jar
```

- indows

```
java -jar lib\karaf-client.jar
```

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



Apache ServiceMix (4.2.0-fuse-01-00)

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

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

10.3.4. Stopping the WKI System

To stop the WKI System type the following command in ServiceMix console:

shutdown
To close the console (WKI System will be still working) type:

logout

10.3.5. 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):

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

10.3.6. WKI DS Components configuration

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.

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.

10.3.7. Troubleshooting

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 solve this problem port 1099 must be freed up before starting s 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.felix.karaf.management.cfg` file by changing the `rmiRegistryPort` and `serviceUrl` property to a port number that is available:

```
rmiRegistryPort = port
jmxRealm = karaf
serviceUrl = service:jmx:rmi:///jndi/rmi://localhost:${rmiRegistryPort}/karaf-${karaf.name}
daemon = true
threaded = true
objectName = connector:name=rmi
```

10.3.8. Links

The latest version of the WKI System:

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

Further information can be found here:

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

Information about log levels:

[1] **ttp://fusesource.com/docs/esb/4.0/runtime/DeployESBLogConfig.html**

10.4. How to deploy OSGi bundles to the WKI system

10.4.1. Introduction

Rationale

Services, created in form of the OSGi bundles, must be deployed to the WKI system. This tutorial shows how it may be achieved.

Prerequisites

- Service bundle(-s) created as described in ‘How to create OSGi service using Spring Dynamic Modules’ guideline,
- he WKI system installed and running - see ‘How to install the WKI system’ guideline.

T

Outcome

The bundle(-s) is deployed to the WKI system and is running (ready to perform services).

10.4.2. Deploying of the bundle

For the sake of simplicity, it is assumed that there is only one bundle `osgi-wki-greetingservice-1.0-SNAPSHOT.jar` in the `target` directory of the Maven project.

Copy `osgi-wki-greetingservice-1.0-SNAPSHOT.jar` to `deploy` directory of your OSGi environment (Hint: it is not required to stop the WKI system to deploy a new bundle - OSGi technology is capable of deploying new bundles in runtime).

To check if it was properly recognized and installed execute `list` command from OSGi shell.

```
karaf@root> list
START LEVEL 100
   ID   State         Blueprint      Spring    Level  Name
[   0] [Active     ] [            ] [       ] [    0] OSGi System Bundle (3.5.1.R35x_v20090827)
[   1] [Active     ] [            ] [       ] [    5] OPS4J Pax Url - mvn: (1.1.2)

...
[ 178] [Active     ] [            ] [       ] [   60] osgi-wki-greetingservice (1.0.0.SNAPSHOT)
```

The status of `osgi-wki-greetingservice` is `Active` which means that the bundle is ready to perform services.

Now this service can be used by other services (it's exported methods are visible and can be executed). Using servicemix console it's possible to stop it (make it unaccessible) or uninstall it (remove it completely). Please refer to the servicemix console help, which can be accessed by typing 'tab' and then 'y':

```
Display all 101 possibilities? (y or n)
```

10.4.3. Troubleshooting

It's not possible to describe all the errors that can occur during deployment of bundles. A general hint that can be given here is, that if after deployment bundle has some other status than `Active` you should check logs. You can type

```
karaf@root> log:display-exception
```

to see the exceptions that occured and act depending on the errors you see.

In general, it's always a good idea to have logs open in another console window - on *nix systems, you can easily achive it by typing in `WKI_HOME`.

```
$ tail -f data/log/karaf.log
```

On Windows systems you can read the `WKI_HOME/data/log/servicemix.log` with any text viewer.

The other way to get logs is to type 'log:display' in WKI-SYSTEM:

```
karaf@root> log:display
```

11. Appendix C - Development Infrastructure

Apart from the implemented elements that constitute the WKI System, there are also other activities of the WP6 that are essential for the development and maintenance of the system. WP6 set up and maintains development infrastructure which allows for effective work of many geographically dispersed teams. The development infrastructure of the WKI project, which is up to standards of current commercial applications development, helps to acquire high quality of all implemented elements.

The main elements of this infrastructure are the following:

- CI server - Hudson
- bug tracker - Mantis
- artifacts repository - Nexus

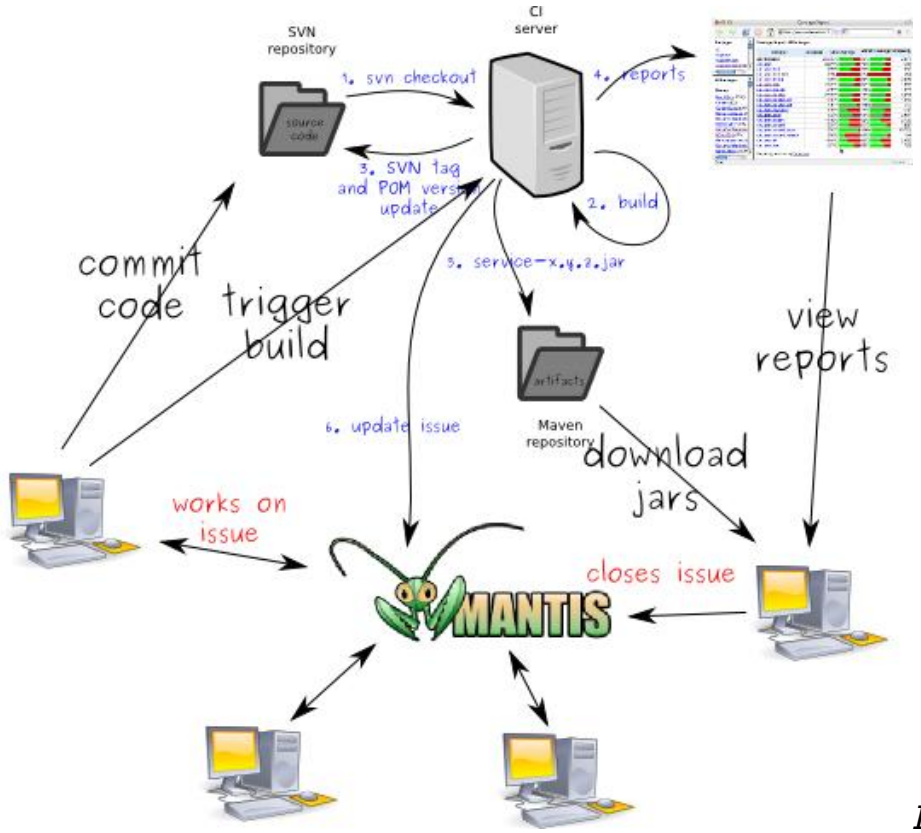


Illustration 10: Development infrastructure

For the description of provided infrastructure please refer to D6.1.2 "Identification of architecture elements and relations, version 2".