WeKnowIt

Emerging, Collective Intelligence for Personal, Organisational and Social Use

FP7-215453

D6.2.1

Definition and implementation of APIs, version 1

Dissemination level	Public
Contractual date of delivery	Month 12, 30.03.2009
Actual date of delivery	
Workpackage	WP6, Architecture and Integration
Task	T6.2 Definition and implementation of APIs
Type	Report
Approval Status	
Version	0.3
Number of pages	110
Filename	

Abstract

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

Firstly, the WKI System prototype is described – its functionality and API – along with client application, which allows to examine the WKI System prototype.

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	12.03.2009	First version	T.Kaczanowski
0.2	06.04.2009	Final version	A.Boruch, R.Janik, T.Kaczanowski, G.Tabor
0.3	10.04.2009	Final version (after internal review)	T.Kaczanowski, J.Kwiecińska

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	J. Kwiecińska	j.kwiecinska@softwaremind.pl
Software Mind	G. Tabor	g.tabor@softwaremind.pl

Executive Summary

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

The WKI System prototype presents how technologies described in D6.1.2 are used to build a functioning system. The prototype presents some simple functionalities in order to prove that the integration and cooperation of services with the WKI Data Storage (WKI DS) is possible. It also proves, that external application can make use of the WKI System API exposed via web services.

Apart from the prototype, this document also presents:

- future evolution of the WKI System prototype,
- list of actually declared services of the WKI System,
- guidelines for services developers.

Abbreviations and Acronyms

API	Application Programming Interface
CSG	Consumers Social Group
CXF	Apache project for service creation and execution
EIP	Enterprise Integration Patterns
ER	Emergency Response
ESB	Enterprise Service Bus
FAQ	Frequently Asked Questions
IDE	Integrated Development Environment
JDK	Java Development Kit
JVM	Java Virtual Machine
HTTP(S)	Hypertext Transfer Protocol (Secure)
JAX-WS	Java API for XML - Web Services
JBI	Java Business Integration
MIME	Multipurpose Internet Mail Extension
OSGi	OSGi Alliance (formerly Open Services Gateway Initiative)
POM	Project Object Model
RMI	Remote Method Invocation
SOA	Service-Oriented Architecture
SQL	Structured Query Language
URI	Uniform Resource Identifier
UDDI	Universal Description, Discovery and Integration
WKI	WeKnowIt
WKI DS	WeKnowIt Data Storage
WP	Work package
WS	Web service
XML	eXtensible Markup Language

Table of Contents

1. Introduction 10
2. The WKI System prototype 11
   2.1. Components of the WKI System prototype 12
      2.1.1. The WKI Data Storage 12
   2.2. The WKI System prototype API 13
   2.3. Installation of the WKI System prototype 13
3. Client application 15
   3.1. Usage of the client application 15
      3.1.1. Upload of file to the WKI System 16
      3.1.2. Add metadata to document 17
      3.1.3. Get document metadata 17
4. The WKI System prototype - summary 19
5. Evolution of the WKI System prototype 21
   5.1. Integration of new services 21
   5.2. Requests of external applications 23
   5.3. Internal changes by WP6 23
6. WKI Services 24
   6.1. Process of services declaration 24
      6.1.1. Services declaration schema 24
      6.1.2. Results of services declaration process 25
   6.2. Services declaration - further steps 26
      6.2.1. Update of services declarations 26
      6.2.2. Workflows specification 26
   6.3. Services and modules of the WKI System 27
   6.4. Services API specification 28
      6.4.1. Service providers and service consumers 29
      6.4.2. Suggested development process 30
      6.4.3. Declaration of services 31
      6.4.4. Exchange of data between services 31
7. WKI development guidelines 33

7.1. Planned guidelines 33
7.2. Development FAQ 34
8. Conclusions 35
9. Attachments 36
10. Annex A – List of declared services 37
10.1. WP1 services 37
10.2. WP2 services 44
10.3. WP3 services 52
10.4. WP4 services 58
10.5. WP5 services 64
10.6. WP6 services 71
11. Annex B – Guidelines 74
11.1. Development Conventions 74
11.1.1. Naming conventions 74
11.1.2. Code conventions 75
11.1.3. Documentation conventions 75
11.2. How to install Java & Maven 75
11.2.1. Introduction 77
11.2.2. Installation of Java 77
11.2.3. Installation of Ant 77
11.2.4. Installation of Maven 78
11.2.5. Links 79
11.3. How to create Maven project 80
11.3.1. Introduction 81
11.3.2. Create a project 81
11.3.3. Development 83
11.3.4. Build your project 85
11.3.5. IDE integration 86
11.3.6. Links 86
11.4. How to install the WKI System 87
11.4.1. Introduction 88
11.4.2. Installation of WKI System 88
11.4.3. Files and folders in WKI_HOME 88

11.4.4. Turn off full-information mode 89
11.4.5. Starting WKI System 89
11.4.6. Stopping WKI System 90
11.4.7. Cleaning of WKI System 90
11.4.8. Troubleshooting 90
11.4.9. Links 91
11.5. How to create OSGi service using Spring Dynamic Modules 92
11.5.1. Introduction 93
11.5.2. How to create a service provider 93
11.5.3. How to create client service 98
11.5.4. Getting some feedback - listeners 102
11.5.5. Spring, OSGi, beans and services 106
11.5.6. References 106
11.6. How to deploy OSGi bundles to the WKI System 108
11.6.1. Introduction 109
11.6.2. Deploying of the bundle 109
11.6.3. Troubleshooting 110

List of Listings

Listing 1 Deploy of the WKI prototype bundles 14
Listing 2 File upload - sample output 17
Listing 3 Adding metadata - sample output 17
Listing 4 Retrieval of metadata - sample output 18

List of Figures

Figure 1 Components of the WKI System prototype 12
Figure 2 Development of the WKI System prototype 21
Figure 3 Apache Camel flow 22
Figure 4 Enhanced Camel flow 23
Figure 5 UML diagram used for workflow description 26
Figure 6 Service consumer / provider interact via service registry (from 20) 29

Figure 7 Vicious circle of requirements 30
Figure 8 Development of service – stubbed dependencies.....31
Figure 10 Direct data exchange 32
Figure 11 Data exchange via the WKI Data Storage 32
Figure 12 Spring, OSGi, beans and services 106

List of Tables

Table 1 Methods of the WKI prototype API.....13
Table 2 Metadata types recognized by the WKI storage prototype 15
Table 3 Client application functionalities and arguments 16
Table 4 Features of the WKI System prototype.....20
Table 5 Service declaration schema 25
Table 6 WKI Modules specification 28
Table 7 WP1_AccountManager 39
Table 8 WP1_LogIn.....39
Table 9 WP1_ManageItem 40
Table 10 WP1_Tag.....40
Table 11 WP1_Comment 41
Table 12 WP1_Rate 42
Table 13 WP1_SearchKB 42
Table 14 WP1_UsersMessaging 43
Table 15 WP1_RSSManager 43
Table 16 WP2_Text_Classification.....45
Table 17 WP2_Text_Clustering 46
Table 18 WP2_Text_Annotation.....47
Table 19 WP2_VisualAnalysis 48
Table 20 WP2_SpeechIndexing 50
Table 21 WP2_SearchInSpeech 52
Table 22 WP3_AnswerSpamDetector.....54
Table 23 WP3_AnswerQualityEvaluator 54
Table 24 WP3_LocalTagCommunityDetector 56
Table 25 WP3_ExpertFinder 57
Table 26 WP3_DetectLatentTopics 58

Table 27 WP4_Community_Design_Language 61
Table 28 WP4_ClosenessCentrality 61
Table 29 WP4_InverseDistanceClosenessCentrality 62
Table 30 WP4_BetweennessCentrality 62
Table 31 WP4_NewmanClustering 63
Table 32 WP4_SNAGraphStatistics.....64
Table 33 WP4_AffiliationNetworkMeasures.....64
Table 34 WP5_GroupManagement 66
Table 35 WP5_TaskManagement 68
Table 36 WP5_IncidentLog 69
Table 37 WP5_LogMerger 71
Table 38 WP6_DataStorage 72
Table 39 WP6_ExternalDataStorageService 73

1. Introduction

This written report of deliverable D6.2.1 "Definition and implementation of API, version 1" accompanies the first WKI System prototype. The prototype is a "proof-of-concept" of the WKI System, showing that selected technologies (described in D6.1.2) can cooperate. Many features of the WKI System are exposed by this prototype, for example: possible cooperation with external applications, communication of services with the WKI Data Storage (WKI DS) service and distribution of services in form of OSGi-fied Java Archive files (JAR[1]) etc.

Apart from the prototype description, this deliverable also presents the actual API of the WKI System. This API constitutes of services declared by all WPs. The process of services declaration is also presented.

Work on the WKI System API will be continued (second version of "Definition and implementation of API" deliverable is scheduled for M18), and this document provides some valuable information for service developers. Firstly, it describes further steps that will be taken in order to make sure that the services provided reflect the needs of the WP7 use-case applications. Then, it describes issues of services API specification, which is strictly connected with cooperation of services. Finally, it includes all the guidelines prepared by WP6, which describes technical details that are important for creation of services compatible with the WKI System.

[1] JAR, Java Archive, file format used to distribute Java classes, based on popular ZIP format.

2. The WKI System prototype

The WKI System prototype aims at proving that the concept of a system, as described in D6.1.2, works in reality and not only on paper. The prototype is a simplified version of the final system, and whilst it is not intended to be complete or fully functional, it does provide an architecture that allows loosely coupled services to be functionally integrated. This meets the requirements for the WKI system of having a coherent, valuable system emerge from a set of independent services.

Whilst the intention is to develop a prototype which meets the current requirements of the WKI System, and can thus be used further as the basis for the real system, the nature of the state of the art research being pursued in the development of the WKI Services means changes may be required as development proceeds. For example, it may prove necessary to implement a less flexibility but more timely interaction between certain services if bottlenecks are encountered. The prototype also utilises a state of the art architectural design intended to handle such unforeseen necessities; as it maintains independence between the elements of the system.

From the technical point of view, the main goal of the WKI System prototype is: to implement the core functionality, to prove that the integration of services is possible, and that a system built in such a way works in practice and provide the actual means of integration. Thus, in summary, the key purposes of the prototype are:

- "proof of concept" for the WKI Data Storage solution,
- foundation on which further versions of the WKI System will be built,
- examination and improvement of development process of WKI services and of development environment,
- final test of guidelines (see section 7) credibility,
- gaining of experience.

This section contains detailed information about the WKI System prototype. Firstly, 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.

2.1. Components of the WKI System prototype

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

Figure 1 presents components of the WKI System prototype:

- Fuse ESB – serves as a Service Oriented Architecture (SOA) and Enterprise Service Bus (ESB) foundation of the WKI System (described in D6.1.2),
- WP6_DataStorage – API of the WKI Data Storage, as described in Table 38,
- WP6_ExternalDataStorageService – web service API of the WKI System; this service is responsible for uploading files to the WKI System, adding metadata and querying data storage, as described in Table 39.

Please notice that the client application presented on Figure 1 is not a part of the prototype.

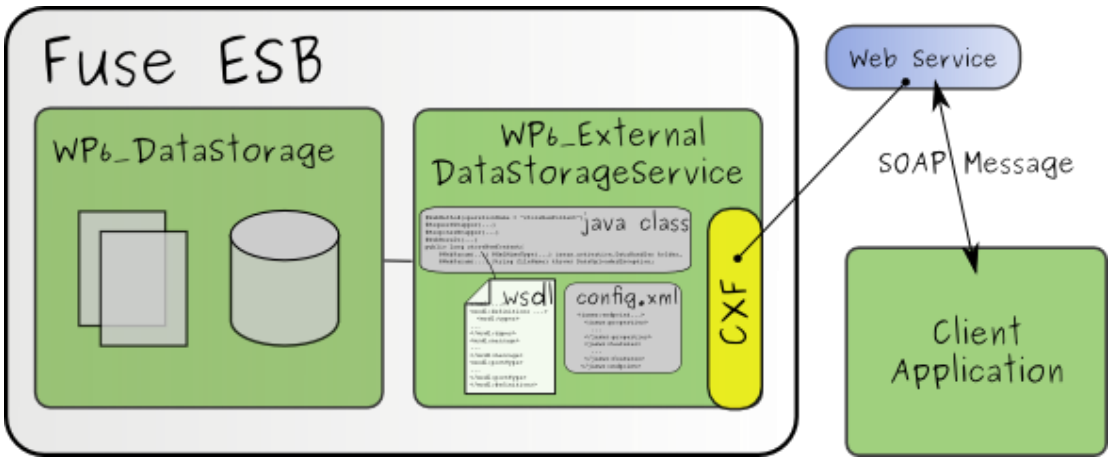

Figure 1 Components of the WKI System prototype

2.1.1. The WKI Data Storage

Full description of the WKI Data Storage and its OSGi bundles can be found in D6.3.

The WKI System prototype uses file system storage (OSGi bundle koda-fscr) to save uploaded documents in file system directory. The directory is specified in configuration file in folder *configs/koda-fscr.properties* of the WKI System. Other kinds of content repositories[2] are also available, but they are not used in the prototype.

[2] For example Apache JackRabbit java content repository in OSGi bundle koda-jcr.

2.2. The WKI System prototype API

The WKI System prototype offers API for accessing storages. This API will evolve to meet the needs of services developed by the WPs as described in 4. Table 1 contains list of methods of the actual API. All these methods can be accessed by external applications (e.g. Emergency Response application) via web services[3].

Method signature	Description
`Long storeNewContent(DataHandler holder, String fileName)`	Adds new file content (holder) to WKI content repository using name given in attribute *fileName*. Returns ID of created document.
`void addMetadataToDocument(Long documentId, String mdataKey, String mdataValue)`	Adds new metadata to document with specified ID.
`String executeSparqlQuery(String sparqlQuery)`	Executes SPARQL query against the Knowledge Base and returns obtained results as XML document.

Table 1 Methods of the WKI prototype API

The methods presented in Table 1 were chosen for the API of the prototype for the following reasons:

- they refer to the requirements of services declared by WPs (presented in section 10),
- execution of these methods examines the crucial parts of the WKI System architecture: integration layer (cooperation of services) and WKI DS,
- to implement such functionality, all technologies from the WKI technologies stack (presented in D6.1.2, section 4) were required.

2.3. Installation of the WKI System prototype

The WKI System prototype **requires Linux operating system** and about 200MB of free disk space. More detailed requirements can be found in guideline *How to install the WKI System* (section 11.4). Information on deployment of bundles to the WKI System are included in *How to deploy OSGi bundles to the WKI System (section 11.6)* guideline.

[3] This process is described in details in D6.1.2, section 5.4.1.

Prototype of the WKI System is distributed as `wki_system.tar.gz` archive.

Installation and running of the WKI System requires two steps, and involves two separate shells[4]:

Step1 – start of the system

Open *shell1* and unpack file `wki_system.tar.gz`:

```
tar xzf wki_system.tar.gz
```

A set of folders will be extracted – the purposes of these folders are described in *Files and folders in WKI_HOME* part of *How to install the WKI System* (section 11.4) guideline.

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

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

After ServiceMix starts, status of all installed bundles can be checked using *list* command:

```
osgi list
```

Please note that it may take few minutes to install all bundles (*osgi list* should return 138 active bundles).

Step2 – deployment of bundles

Open *shell2*, proceed to the installation directory and install all the WKI prototype packages using *deploy_wki_bundles.sh* script.

```
chmod 755 ./deploy_wki_bundles.sh
./deploy_wki_bundles.sh
```

It will print some information as presented on Listing 1.

```
START DEPLOYING WKI SYSTEM PROTOTYPE
START DEPLOYING koda-commons
START DEPLOYING koda-fscr
START DEPLOYING koda-jcr
START DEPLOYING koda-cr
START DEPLOYING koda-db
START DEPLOYING wki-commons
START DEPLOYING wki-ds
START DEPLOYING wki-uploader-service
ALL WKI SYSTEM PROTOTYPE BUNDLES WERE DEPLOYED
```

Listing 1 Deploy of the WKI prototype bundles

Now the WKI System is ready to receive requests from client applications which is described in section 3.

[4] Shell is a standard command-line interpreter available in every Unix/Linux distribution.

3. Client application

The general purpose of prototype client application is to test the basic user functionalities (upload and access) of the WKI System prototype. The client is implemented as a standalone application, using the JAX-WS technology to create a web service client. The application provides a command-line interface, which allows the user to interact with the WKI System prototype web services.

In WKI DS content, and metadata relating to this content are virtualized in form of a document. Basically, the prototype distinguishes three types of metadata:

- automatically extracted information like mimetype[5], size or name of file,
- metadata provided by user,
- URI reference pointing to externally located file.

At the current stage of development, five types of metadata are extractable, however, it is planned to extend them with additional ones:

Metadata type	Description
CONTENT_FILE_UPLOAD_TIMESTAMP	file upload time
CONTENT_FILE_SIZE	file size
CONTENT_FILE_MIME	file MIME type
ORIGINAL_FILE_NAME	file name
CONTENT_URI	file URI

Table 2 Metadata types recognized by the WKI storage prototype

3.1. Usage of the client application

Client application is distributed in *wkiClient.tar.gz* archive file. It can be extracted using *tar* command:

```
tar xzf wkiClient.tar.gz
```

Main client's executable file is *wkiClient.sh* script. Before usage, it's access rights must be changed appropriately:

```
chmod 755 wkiClient.sh
```

Script can be executed with one of the following arguments:

[5] MIME Media type is a two-part identifier of file format, e.g. "application/zip" for zip files, "text/csv" for comma separated values, etc.

Short parameter	Long parameter	Arguments	Description
-f	--file	filePath	Upload file to the WKI System.
-m	--metadata	documentId, metadataType, metadataValue	Add metadata to document.
-ad	--alldata	documentId	Returns all metadata of document.

Table 3 Client application functionalities and arguments

Functionalities listed in Table 3 are explained in details in sections 3.1.1 - 3.1.3.

3.1.1. Upload of file to the WKI System

When new document is added to the WKI System, content of the file is placed in one of the available content repositories and metadata are saved in KB.

Usage:

```
./wkiClient.sh -f file_path
```

Example usage:

```
./wkiClient.sh -f ~/testFile.pdf
```

Output:

Information about file upload is displayed on the console. The last line contains the ID (unique identifier) of the newly uploaded document. Listing 2 presents sample output[6].

```
DEBUG [...app.UploaderClientMainApp] - Adding new file to WKI
storage
DEBUG [...JaxWsClient] - Get port
INFO [...JaxWsClient] - http://schemas.xmlsoap.org/wsdl/soap/http
INFO [...JaxWsClient] -
[http://schemas.xmlsoap.org/soap/actor/next]
DEBUG [...JaxWsClient] - Prepare request
DEBUG [...JaxWsClient] - Content type : application/octet-stream
name: wkiClient.sh
DEBUG [...JaxWsClient] - Store new content request
INFO [...JaxWsClient] - Document was successully stored with ID:
1239352146065
File was successfully stored with ID: 1239352146065
```

[6] On this listing, and other listings in this section, some information printed by the WKI System were omitted for the sake of readability.

Listing 2 File upload - sample output

3.1.2. Add metadata to document

To add metadata to document stored in the WKI DS, three parameters are required:

- documentID – same ID as returned by file upload functionality (section 3.1.1),
- metadata type – one of the values presented in Table 2,
- value – arbitrary text value.

Usage:

```
./wkiClient.sh -m document_id,METADATA_TYPE,metadata_value
```

Example usage:

```
./wkiClient.sh -m 1239352146065,CONTENT_FILE_UPLOAD_TIMESTAMP,'12
dec 1970'
```

Output:

Detailed information about operation progress is logged on the screen as presented on Listing 3 Adding metadata - sample output.

```
DEBUG [...app.UploaderClientMainApp] - Adding new metadata
INFO [...app.UploaderClientMainApp] - Try add metadata type:
'CONTENT_FILE_UPLOAD_TIMESTAMP' with value: '12 dec 1970' for
document with ID: '1239352146065'
DEBUG [...JaxWsClient] - Try get service entry
DEBUG [...JaxWsClient] - Get port
INFO [...JaxWsClient] - http://schemas.xmlsoap.org/wsdl/soap/http
INFO [...JaxWsClient] -
[http://schemas.xmlsoap.org/soap/actor/next]
DEBUG [...JaxWsClient] - ADD METADATA
```

Listing 3 Adding metadata - sample output

3.1.3. Get document metadata

To retrieve metadata of a document, it must be referenced by its identifier. This section shows the retrieval of metadata of MsWord document with ID 123720801563.

Usage:

```
./wkiClient.sh -ad document_id
```

Example usage:

```
./wkiClient.sh -ad 1237208013563
```

Output:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
```

```
  <head>
    <variable name="key"/>
    <variable name="value"/>
  </head>
  <results>
    <result>
      <binding name="key">
        <uri>http://koda/metadata/types#contentUri</uri>
      </binding>
      <binding name="value">
        <literal>
         file://wkimaster#/home/anbo/wki/fs_cr_repo/PdfNaDoc.doc
        </literal>
      </binding>
    </result>
    <result>
      <binding name="key">
        <uri>http://koda/metadata/types#fileuploadtime</uri>
      </binding>
      <binding name="value">
        <literal>23 feb 2009</literal>
      </binding>
    </result>
  </results>
</sparql>
```

Listing 4 Retrieval of metadata - sample output

4. The WKI System prototype - summary

Previous sections described parts and installation process of the WKI System, as well as a client application that is used to show its capabilities.

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

The prototype of the WKI System offers some basic functionality that will evolve towards the full WKI System (as described in section 5). The aim of the prototype is to use the same set of technologies, bound in the same way, that the final system will use.

In case of the WKI System prototype, full stack of the technologies that are intended to be used in the final system (as described in D6.1.2) have been used.

Table 4 lists some features of the WKI System that are presented by the WKI System prototype:

Feature	Prototype example	Technology
Use of the WKI System by external applications.	Client application accessing web services API of the WKI System.	Web services (Apache CXF)
OSGi bundles packed as JAR files are used to distribute services.	WP6_ExternalDataStorageService and WP6_DataStorage are distributed as JAR files with custom, OSGi-compatible MANIFEST.MF file.	OSGi, Maven-bundle-plugin
The WKI DS as one of many services.	The WKI DS distributed in form of OSGi bundles, and exposing its functionality via OSGi registry.	OSGi, Spring Dynamic Modules
Configurable services.	File system set as a storage for the WKI DS with XML configuration.	Spring Dynamic Modules
Passing of arbitrary data structures between services.	Uploaded file, wrapped as IWKIDocument and passed to the WKI DS service.	Apache Camel
Cooperation of services.	WP6_ExternalDataStorageService exchange data with WP6_DataStorage service.	OSGi
No hardcoded dependencies between services.	WP6_ExternalDataStorageService has dependency on WP6_DataStorage injected.	Spring
It is possible to store files in the	Uploaded file is written to the file system repository (which is part of	

WKI DS.	the WKI DS).	
It is possible to store triples in the WKI DS.	Storage/retrieval of file metadata.	Jena
The WKI DS allows to execute arbitrary SparQL queries.	*executeSparqlQuery* method of the prototype API.	Jena

Table 4 Features of the WKI System prototype

The features of the WKI System prototype listed in Table 4 are strictly connected to the integration part of the project, that is realized by WP6. The WKI System prototype has proved that the selected technologies (described in D6.1.2) are capable of integrating services. The prototype also shows that it is possible to achieve required level of flexibility.

Assuming that all the services are developed in accordance with the process described in D6.1.2, it is possible to integrate them using techniques that were examined during the work on the WKI System prototype.

5. Evolution of the WKI System prototype

As it was stated in section 2, the WKI System prototype will evolve towards the full WKI System. The development will be driven by three main forces:

- new services created by WPs, and updates of existing services,
- requests made by WP7 applications,
- internal changes by WP6.

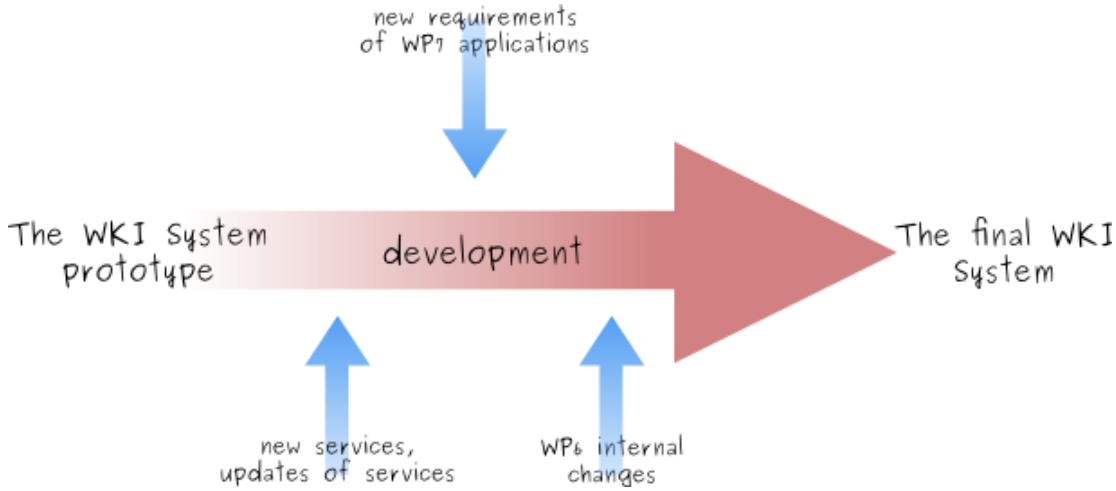

Figure 2 Development of the WKI System prototype

Figure 2 presents the forces which have an influence on the development of the WKI System. It should be stressed, that the development will proceed according to the integration plan described in D6.1.2. This means that many intermediary versions of the WKI System will be created (and tested) before formation of the final system.

5.1. Integration of new services

If a new service is developed[7] it needs to be included into the WKI System.

This process can be best described by example. Let us assume that the WKI System contains four OSGi bundles[8]. Three of them contain services:

[7] Changes of the existing service (especially API changes) may require steps similar to the ones described in this section.

- WP6_DataStorage – API of the WKI Data Storage, as described in Table 38,
- WP6_ExternalDataStorageService – web service API of the WKI System, as described in Table 39,
- WP1_ManageItem – service for file uploads, as described in Table 9.

The fourth bundle – WP6_Integration – contains Camel flow (as described in D6.1.2) which specifies the flow presented on Figure 3. All these four elements, when deployed to the WKI System, provide some basic functionality – a file with metadata can be uploaded by external application to the system and stored in the WKI Data Storage.

Figure 3 Apache Camel flow

Now let us assume that the next service is developed – WPX_SpamDetector, which is able to analyze text (metadata)from the point of spam content. In order to incorporate this service into the WKI System two actions have to be taken:

- WPX_SpamDetector service must be deployed to the WKI System,
- WPX_SpamDetector service must be integrated with other services.

First point is straightforward – WPX_SpamDetector service must be deployed as described in *How to deploy OSGi bundles to the WKI System* (section 11.6) guideline.

The second point might be resolved by updating the existing Camel flow as presented on Figure 4. Thanks to use of the appropriate Enterprise Integration Patterns (EIP), WPX_SpamDetector can act like a "guard," allowing only trusted content to enter the WKI Data Storage.

---

[8] For the sake of simplicity some parts of the WKI System that would be required to realize such scenario, and some technical aspects are omitted.

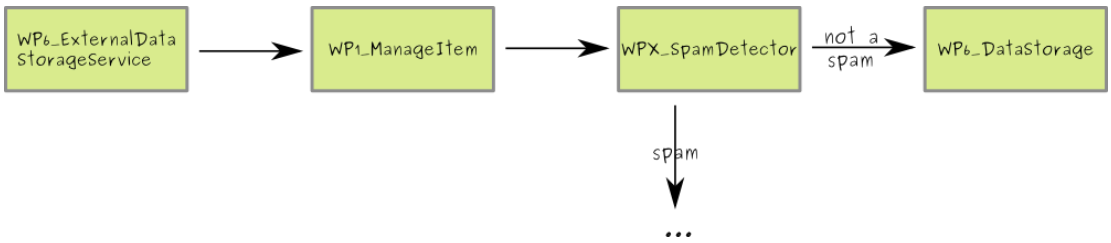

Figure 4 Enhanced Camel flow

After introduction of these two changes the WKI System is now taking advantage of the functionality provided by WPX_SpamDetector service.

5.2. Requests of external applications

WP7 aims at creating two applications that will exploit and present capabilities of the WKI System. Both applications will use the WKI System via it's web services API. This API, as stated in D6.1.2, will expose functionalities provided by:

- single services,
- composition of services (Camel flows).

The development of WP7 applications will be preceded by workflows specification, as described in section 6.2.2. It is expected that during workflows specification and during actual development new requirements regarding the WKI System API will emerge. These requirements can result in:

- creation of new services,
- exposition, or compositions of services that are not yet exposed via web services API,
- creation of new compositions of services.

Task of creation of new service will be appointed to appropriate WP. Other tasks belong rather to integration part of the project, so WP6 will take care of them[9].

5.3. Internal changes by WP6

It is also expected that during work on integration, WP6 will introduce some changes to the WKI System, for example, addition of performance monitoring. Such tasks might result in creation of new services, creation of new compositions of services and updates of existing compositions of services.

[9] In some cases (i.e. exposition of existing service) it might be better if developers of particular service could take care of this – such situations will be resolved between WP6 and responsible WP.

6. WKI Services

As stated in D6.1.2, a WKI service "is a building block that provides well-defined functionality and can be used as element of larger structures". Integrated services provided by WPs constitute the WKI System, and allow to build various applications using functionalities that they provide.

The list of services declared by WPs is available in *Annex A – List of declared services* (section 10).

This section describes the process of services declaration and explains the dependencies between modules of the WKI System (presented in D6.1.2) and the WKI services. It also provides information on further steps and enhancements of the process of service declarations and discusses topic of service integration.

6.1. Process of services declaration

The first step towards integration of services is to identify them. All WPs were requested by WP6 to provide information about the services that they intend to develop for the WKI System.

Achievement of few goals was intended:

1. to identify the services, and make the role which they play in the WKI System clear,
2. to identify the connections between the services,
3. to identify data structures used by services.

It is important to notice that even if this process was initiated by WP6 and with integration issues on mind, all WPs benefit from it by enhancing their knowledge and understanding of the whole system and possible interactions with services created by other WPs.

All WPs have filled prepared wiki pages with declarations of their services. Then, in an iterative process these declarations were reviewed by WP6 and some changes necessary to have a clear and consistent declarations were requested from responsible WP. The whole process resulted in list of consistently described services.

6.1.1. Services declaration schema

The schema for the services declaration (presented in Table 5) was proposed by WP6.

Name	Unique name (serves as identifier), in format WPx_ServiceName, e.g. WP99_StockService

Description	General purpose - what is the general function.
Usage	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).
Required Services	Names of other services this service requires.
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
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 5 Service declaration schema

6.1.2. Results of services declaration process

The process of services declaration resulted in more than 30 declarations of services. All declarations share common structure, which makes them easy to understand.

Information about services provided by WPs reflects progress of work on each of them. Because of this some details (e.g. exceptions thrown) are still missing and will be filled on later date. This omission does not undo the value of services declarations.

During the process some discrepancies between services declared by WPs have been discovered and corrected. For example, some functionalities were provided by more than one service. Such situations were discussed by responsible WPs and duplicates were removed.

6.2. Services declaration - further steps

Despite the services being much better described and understood by all WPs, there is still much work ahead that must be done in order to create a valuable system based on them.

6.2.1. Update of services declarations

First of all, services declaration task is not finished. Information contained on wiki pages should be updated according to the progress of development. WP6 will overview this task requesting all WPs to update the wiki page frequently.

It is also expected that many new services will be declared by WPs.

6.2.2. Workflows specification

Right now it seems that the most important task is to identify the services necessary to fulfil the requirements of both use-cases. This is crucial for the creation of the WKI System.

This process has already been started by WP6 & WP7. It was agreed that UML sequence diagrams will be used to present the flow of control between services. Figure 5 presents a sample workflow which describes the cooperation of services in order to give a user permissions to upload images from his mobile[10].

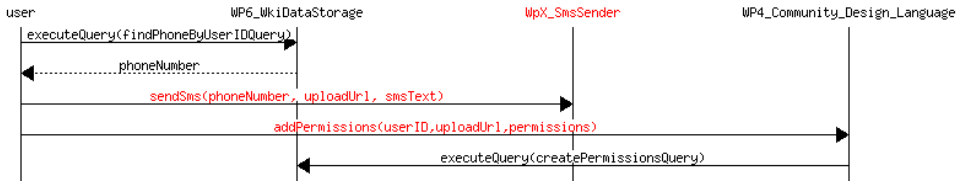

Figure 5 UML diagram used for workflow description

Figure 5 shows how phone number of a user is extracted from the WKI Data Storage with SPARQL query, and then used to send SMS to the user. What is more, cooperation between WP4_Community_Description_Language and WP6_WkiDataStorage is presented. Red colour is used to mark elements that do not exist yet:

- services (e.g. WpX_SmsSender),
- or methods which are expected to be provided by a particular service (e.g. "addPermission" method of WP4_Community_Design_Language).

[10] This sample workflow is roughly based on "Assign access temporary right" issue described in D7.2.

The process of workflow specification will involve all WPs. WP6 and WP7 will provide workflows from the point of view of use-case applications. Each service provider is expected to provide information (in form as presented on Figure 5) on the internal workings of his service. For example, WP6 and WP7 know that to assign permissions to user, WP4_Community_Design_Language service must be used, but they have no knowledge about the internal workings of this service (i.e. about the services that this service uses internally). Thus, "general" workflows prepared by WP6 & WP7 must be detailed by other WPs.

Thanks to workflows specification process it will be clear:

- what services are required,
- what methods are required, and what types are exchanged.

These two pieces of information are crucial for the integration of services.

6.3. Services and modules of the WKI System

Table 4 shows how services declared by WPs fit into WKI modules (described in D6.1.2).

WKI module name	WP services names
User and group management	WP1_AccountManager WP1_LogIn WP1_UsersMessaging WP4_Community_Design_Language WP5_GroupManagement
Data processing	WP2_Text_Classification WP2_Text_Clustering WP2_Text_Annotation WP2_VisualAnalysis WP2_SpeechIndexing WP3_LocalTagCommunityDetector
Data storage	WP6_DataStorage WP1_ManageItem
Search	WP1_SearchKB WP6_DataStorage WP2_SearchInSpeech

	WP2_VisualAnalysis WP3_ExpertFinder
Common services	WP5_TaskManagement WP1_Tag WP1_Comment WP1_Rate
WKI Web service API	WP6_ExternalDataStorageService
Other	WP3_AnswerSpamDetector WP3_AnswerQualityEvaluator WP3_DetectLatentTopics WP4_ClosenessCentrality WP4_InverseDistanceClosenessCentrality WP4_BetweennessCentrality WP4_NewmanClustering WP4_SNAGraphStatistics WP4_AffiliationNetworkMeasures WP5_IncidentLog WP5_LogMerger WP1_RSSManager

Table 6 WKI Modules specification

Set of services belonging to each module presented in Table 6, as well as modules itself, will be updated according to changes described in section 5.

6.4. Services API specification

Deliverable D6.1.2 describes the technical aspects of services development and integration. This section presents ways of creating services API that would allow for integration of the WKI System.

Concepts presented in next sections have been discussed and accepted by all partners during the WKI consortium meeting in Koblenz (October 2008).

6.4.1. Service providers and service consumers

As previously stated, the WKI System is based on the SOA approach. It means that each service can act as service provider and/or[11] service consumer:

- service provider – provides some functionality, and makes it accessible to other services (to service consumers) by publishing information about accessing this service in service registry (e.g. Universal Description, Discovery and Integration (UDDI) registry, OSGi registry),
- service consumer – makes use of functionality provided by service provider; uses service registry to discover and get access to information about the available services.

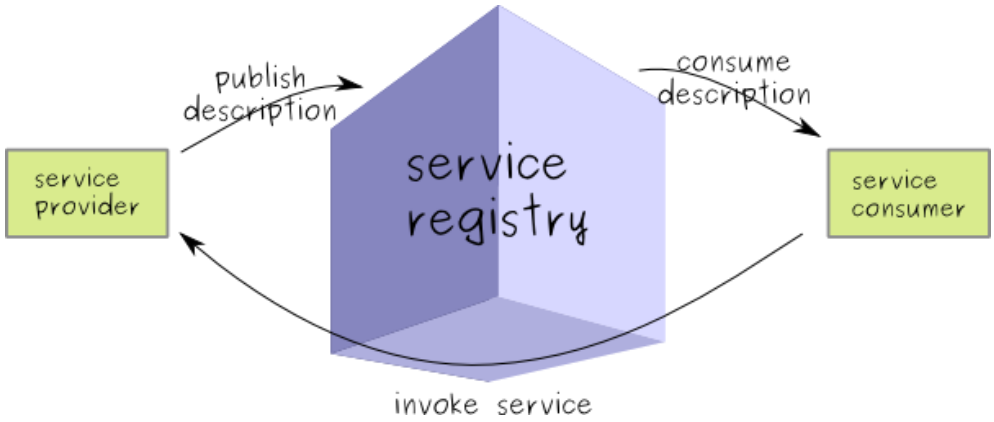

Figure 6 Service consumer / provider interact via service registry (from 20)

In the first phase of the WKI project, a problem with services development occurred. None of WPs knew exactly how API of their services should look like. At the same time, WPs were unable to specify what were their requirements regarding services provided by other WPs. This "vicious circle of requirements" is presented on Figure 7 by example of WP6 (architecture provider) and some other WP (WPx):

[11] One service can play both roles.

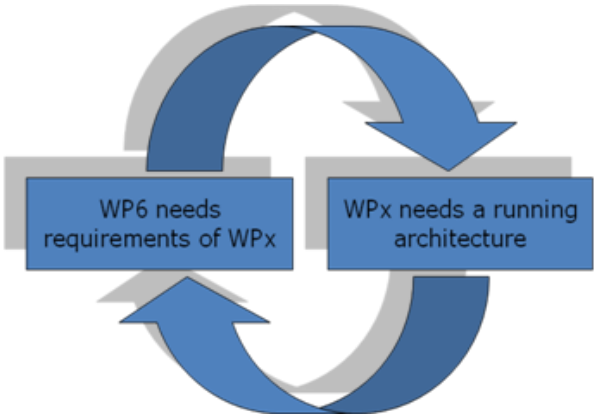

Figure 7 Vicious circle of requirements

To break out of this difficult situation WP6 has proposed a solution described below. The main idea is to share responsibilities of service providers and service consumers, so they can act independently.

Service provider is responsible for:

- definition of signatures of:
  - provided services,
  - input structures
- definition of output structure,
- specification of storage type.

Service consumer is responsible for:

- adoption of required input structure – gathering data from data storage or requesting data from other service.

6.4.2. Suggested development process

In general, developers of the WKI services should develop their services, as if all other services that they require, have already existed. These required services should be simulated (stubbed) during the development process. In this way, services can be developed independently, without delays caused by dependencies on services that donot exist yet. Figure 8 presents this idea. All the services required by "service X" are simulated.

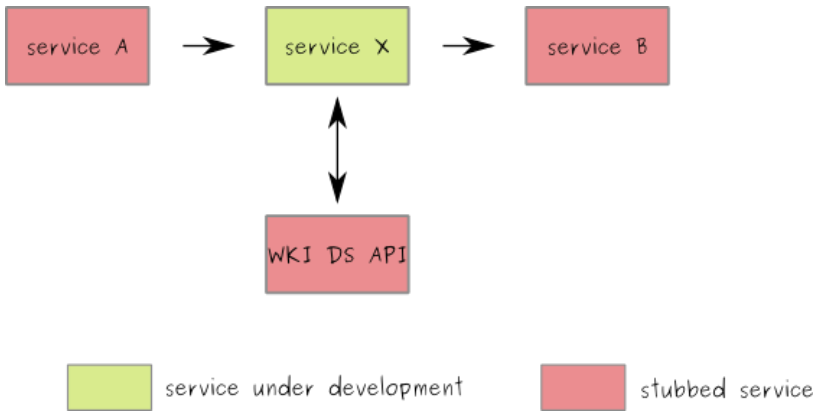

Figure 8 Development of service – stubbed dependencies

During development abstractions of input, output and storage are used. When the real services are available in the WKI System, then these simulated components are replaced by them (by real services provided by other WPs).

6.4.3. Declaration of services

To avoid confusion and misunderstandings between service developers, an effort has been made to identify and describe all the services planned for the WKI System. This task has not been finished yet. The services declared so far were presented in section 10.

6.4.4. Exchange of data between services

Services exchange data with each other. This is challenging for, at least, two reasons:

- API and functionality of both services must be well specified, and well understood,
- no matter the size of the data, it must be exchanged efficiently.

While previous sections described the process of API setting, this section concentrates on actual exchange of data.

In general, data can be delivered in two ways: directly or using a middleman.

Direct data exchange

Figure 10 presents how direct data exchange works. If Service 1 needs to retrieve a relatively small amount of data from the Service 2, than the exchange of data can be performed directly. In such case, data is transported in request and/or response body. This method implies that both services has agreed on some common data structure that is passed between them.

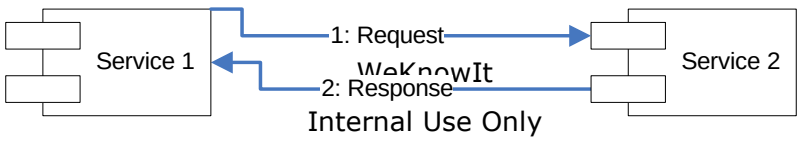

Figure 9 Direct data exchange

Data exchanged via the WKI Data Storage

If the amount of data requested by a service is relatively big, it might be inefficient to transfer it directly between services. Let's consider a following scenario:

1. Service 1 requests Service 2 for a graph that is a result of some computations performed by Service 2.
2. Service 2 performs computations, and than stores the result in the WKI Data Storage.
3. WKI Data Storage returns a unique identifier (URI) that points to the data written in the previous step.
4. Service 2 sends this URI to Service 1.
5. Service 1, in order to get the actual data needs to request it from the WKI Data Storage.
6. WKI Data Storage sends the data to Service 1.

Figure 11 presents such a situation. As seen from the scenario between services 1 and 2, only very small amount of data is exchanged (request, and than URI as a response). The real transfer of data is performed between single service and the WKI Data Storage.

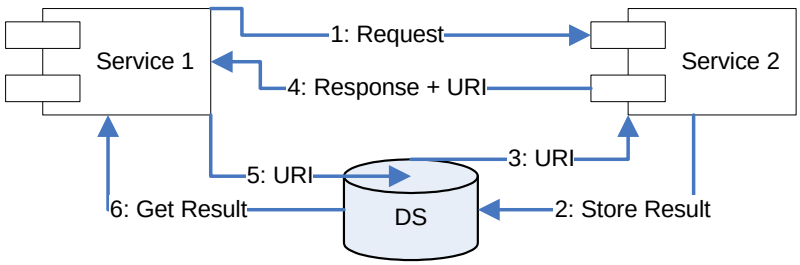

Figure 10 Data exchange via the WKI Data Storage

7. WKI development guidelines

In D6.1.2, the process of choosing technologies for the WKI System is described. One of many factors that were taken into consideration is *ease-of-use*. Even if current technologies come with good documentation and samples, still the amount of reading needed to start the development might be overwhelming. It's especially true for projects like WeKnowIt, where many different frameworks, libraries and tools are used.

Its WP6 responsibility to make sure that every partner involved in the development of the WeKnowIt services is able to provide them in a form, that allows to integrate them in the whole WeKnowIt system. In order to achieve this goal, a set of guidelines has been prepared. **These guidelines provide all the information needed to develop a service that is compatible with the WKI architecture – from development infrastructure setting, to deploying developed service to the WKI System.**

Till now, following guidelines have been released:

- Development Conventions (ver 0.1)
- How to install Java & Maven (ver 0.5)
- How to install the WKI System (ver 0.2)
- How to create Maven project (ver 0.4)
- How to create OSGi service using Spring Dynamic Modules (ver 0.2)
- How to deploy OSGi bundles to the WKI System (ver 0.1)

All guidelines can be downloaded from the main project wiki page. They are also included in the Annex A.

7.1. Planned guidelines

The list of guidelines is not closed. It's expected that during the WKI development phase, WPs will request more information regarding various aspects of technologies used. Right now, at least two more guidelines are planned. The first one will explain how to expose WKI services as web services using Spring Dynamic Modules framework. The second one will explain Maven's dependency issues (it's not yet decided if it will be a separate guideline or an enhancement of one of the existing).

7.2. Development FAQ

Apart from the guidelines, there is also a *Development FAQ* available on main wiki. Its purpose is to deliver fast answers to the urgent questions that arise during development. The questions and answers from this FAQ will be eventually included into guidelines.

8. Conclusions

This document provides information on three topics. Firstly, it describes the content and usage instruction of D6.2.1 prototype. Then, it provides a list of services, which constitutes the API of the WeKnowIt system. Lastly, it describes the guidelines which were prepared in order to help WPs implement these services.

This document is connected with two other WP6 deliverables of M12. D6.1.2 explains the selection of technologies, that are used in this prototype while D6.3 presents the WKI Data Storage, which is part of the prototype.

9. Attachments

Following attachments are available:

- wki_system.tar.gz – archive that contains the WKI System, some shell scripts which helps to install the system,
- wkiClient.tar.gz – archive with example console client application. Contains jar file, directory with some files needed by client and script which helps using client application,
- DataUploadService.wsdl – wsdl file from bundle which exposes Web services.

10. Annex A – List of declared services

This section provides list of services as declared by WPs. Information about services of each WP is preceded with general description.

10.1. 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 four 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 CS 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	The creation/editing/deletion of a user's account. An account is attached to a user profile.
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 of 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 7 WP1_AccountManager

Name	**WP1_LogIn**
Description	User Account Login & Logout
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 8 WP1_LogIn

Name:	**WP1_ManageItem**
Description:	***Purpose***: The uploading/deleting of a captured item (photo, video, document, etc.) to the WeKnowIt KB
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 sig-	*int uploadItem(URI path, Permission perm)* - Returns itemID; negative values could be reserved for

nature:	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 9 WP1_ManageItem

Name:	**WP1_Tag**
Description:	The addition/deletion/modification of tags attached to the uploaded content items
Required Services	WP2: Text Annotation Tool WP2: Visual Analysis
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 10 WP1_Tag

Name:	WP1_Comment
Description:	The addition/deletion/modification of comments to the uploaded content 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 signature:	*int addComment(int itemId, String comment)* - returns 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 11 WP1_Comment

Name:	WP1_Rate
Description:	***Purpose***: The addition/deletion/modification of rating to the uploaded content items
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 12 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 results, 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 13 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 14 WP1_UsersMessaging

Name:	**WP1_RSSManager**
Description:	***Purpose***: managing the subscription to RSS feeds ***Type***: internal ***Approach***: to notify users for example about any new incidents ***Benefit***: the prompt notification of users about any new incidents of their interest
Usage:	-
Method signature:	*subscribeRSS(userId, RSSId, password)* - returns: yes/no *deleteRSS(userId, RSSId, password)* - returns: yes/no
Types description:	password must be at least 8 characters with at least 2 numbers
Exceptions:	Bad userId Bad RSSId Bad password

Table 15 WP1_RSSManager

10.2. 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 off-line. 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 re-

	lating to different categories. The pre-defined 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:	*List<ClassificationModel> getClassificationModels()* - returns the possible models which can be used for classification *Prediction classify(Text text, ClassificationModel modelId)* - classify the specified text against the specified model.
Types description:	Note: this is just a preliminary tentative list of data types that are subject to change! Text: The text object can include Plain Text: in a specific encoding (default to utf8) Metadata: author id, creation time, notations, etc. ClassificationModel: 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. Prediction: The prediction object returns a distribution of likelihoods over the categories in the classification model.
Exceptions:	None.

Table 16 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 normalize 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 Consumer - ?
Method signature:	Note: this is just a preliminary tentative list of methods that are subject to change! *Collection<Cluster> cluster(Collection<Text>, ClusterParameters...)* - returns clusters of texts generated from the specific text collections
Types description:	Note: this is just a preliminary tentative list of data types that are subject to change! Text: The text object can include Plain Text: in a specific encoding (default to utf8) Metadata: author id, creation time, notations, etc. ClusterParameters: This is a set of parameters used by the clustering process, e.g. number of clusters to form or similarity thresholds for clustering. Default parameters are likely to be adopted in most cases. Cluster: A collection of Texts.
Exceptions:	

Table 17 WP2_Text_Clustering

Name:	**WP2_Text_Annotation**
Description:	Automatically annotates entities in the text, using some pre-defined annotation model. For each entity type (e.g. geospatial, temporal) an annotation model is developed (off-line). These entity models are then applied to text to recognize 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:	Note: this is just a preliminary tentative list of methods that are subject to change! *Collection<AnnotationType> getAnnotstionTypes*() - returns the collection of possible Annotation Types *annotate(Collection<Text>, AnnotationParameters...)* - added annotations to a collection of texts, according to the specified annotation parameters
Types description:	Note: this is just a preliminary tentative list of data types that are subject to change! Text: The text object can include Plain Text: in a specific encoding (default to utf8) Metadata: author id, creation time, notations, etc. AnnotationParameters: This is a set of parameters used by the annotation process, e.g. The annotation types (Geo-spatial, temporal, etc.) and related parameters (boundary for geo-spatial area or temporal limits), thresholds for annotation confidence, etc. Default parameters are likely to be adopted in most cases. AnnotationTypes: The collection of possible annotation types (e.g. Geo-spatial, temporal)
Exceptions:	None.

Table 18 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:	Note: this is just a preliminary tentative list of methods that are subject to change! *ViralType visualSearch(String imageURI)* - performs a visual search for images similar to the input, along with their tags and geographic location.
Types description:	Note: this is just a preliminary tentative list of data types that are subject to change! *ViralType* - this type contains the following information: image URIs (Map<String, float> imageURIs), image tags (Collection<String> imageTags) and image geotags (Collection<GeoTag> imageGeoTags). *GeoTag* - this type contains the following information: longitude (double longitude) and latitude (double latitude).
Exceptions:	*URINotFoundException* - If given URI does not exist. *IllegalArgumentException* - If parameter is null.

Table 19 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_RetrieveMetadata
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 ser-

	vice.
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 20 WP2_SpeechIndexing

Name:	**WP2_SearchInSpeech**
Description:	The service evaluates a query on the identified speech resources and returns the most probable hits. The user query is analyzed first. The service checks the format of the given query string, parses it and prepares the actual search. The given list of speechindexURIs is compared to the content of the internal repository and the links to non-existing indices are reported. If the list of speechindexURIs is empty, the service performs the search on the entire

	collection of the processed recordings. Service-specific settings are parsed and the search-related parameters are set accordingly. Based on the given settings and the query, the service retrieves relevant metadata and prepares filtering of the search results. Finally, the set of keywords is searched in a given list of previously indexed speech recordings. The service returns a list of the parts from the relevant files that most probably correspond to the given query. The list is sorted according to the confidence estimation.
Required services:	WP6_RetrieveMetadata
Usage example:	ER: User tools invoke this service to identify the (parts of) recordings containing particular keywords. It is expected that the use will correspond to high user-involvement scenarios in the first phase. It will involve a straightforward presentation of (a subset of) the resulting list of hits, the possibility to play the particular part of the audio file and the user's decision what to do with the findings. In future, more advanced services can be built on top of this service that will put together the evidences from various media analyses and base further processing on their combination.
Method signature:	*Hits searchInSpeech(QueryString query, List<String> speechindexURIs, ParameterSpecification settings)* - based on settings - service-specific parameters of the search process - the service evaluates the given query on the specified indexes and returns the most probable hits
Types description:	Hits - a list of probable hits - the triples (speechURI, file_offset, confidence) ordered by the descending confidence. QueryString - a string specifying the set of keywords to be searched and their relative positions (such as the fact that they should form a continuous phrase). ParameterSpecification - a structured format defining values of attributes that specify the settings under which the search should operate.
Exceptions:	InsufficientPermissionException - user does not have the right to perform this operation EmptyQueryException - the parameter is null or

	empty WrongQueryFormatException - the given query cannot be parsed (the location of the probable error is indicated) MetadataNotAvailableException - to evaluate the query, the system needs a metadata that is not available (the type of metadata that cause the problem is reported) SpeechFileNotIndexedException - the speech file specified in the query has not been indexed (the particular URI is reported) SpeechIndexDoesNotExistException - the speech index corresponding to one of the speechindexURIs provided in the input cannot be found (the particular URI is reported) SettingsParseException - the settings parameter does not correspond to the expected format (the location of the probable error is indicated) ServiceInternalErrorException - any other error (that usually needs an action by the service administrator). if any of the above exception occurs, the speech index will not be created

Table 21 WP2_SearchInSpeech

10.3. 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_AnswerSpamDetector**
Description:	In the context of a Questions & Answers (QA) system, this service can flag an answer to a question as spam.
Required services:	None.
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 judgment would be necessary to reach the final decision) *SpamType isSpam(Question q, Answer answer*) - checks whether the input answer is spam with reference to the input question, e.g. it is ironic, vulgar, etc. (output semantics: SPAM -> spam with high confidence, NO_SPAM -> not spam with high confidence, SPAM_CANDIDATE -> Potentially spam, but human judgment would be necessary to reach the final decision)
Types description:	SpamType: enumeration {NO_SPAM, SPAM, SPAM_CANDIDATE} Question (id, ownerId, headline, text, timeOpened, timeClosed). Also see: Question

	Answer (id, ownerId, questionId, text, createTime). Also see: Answer
Exceptions:	No exceptions identified so far.

Table 22 WP3_AnswerSpamDetector

Name:	**WP3_AnswerQualityEvaluator**
Description:	In the context of a QA system, this service provides a measure of quality for an answer (as to how valuable this answer is in answering the given question). This implementation finds topics covered by question and topics covered by answer. Quality is calculated by comparing how many topics from question are covered within answer, and to what extend (e.g. if the topic defined as important for question is also important in answer).
Required services:	None.
Usage example:	In order to determine the order of presentation of answers to a question, the system should call this service for each question-answer pair. Question and answer structures *must* have filled appropriate text fields, so service can return correct results (otherwise analysis will be not possible).
Method signature:	*double evaluateAnswerQuality(Question q, Answer a)* - provide a measure of the quality of the input answer with reference to the input question (output: 0.0 = bad quality, no coverage of topics from question in the answer, 1.0 = top quality; all topics from question are covered in answer)
Types description:	*Question.* Important fields that are used in analysis: Question.headline and Question.text *Answer.* Important field that is used in analysis: Answer.text
Exceptions:	No exception defined. In case of incorrect, empty or null input, zero value is returned (indicates very bad quality).

Table 23 WP3_AnswerQualityEvaluator

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:	None.
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<Tag> getRelatedTags(String tag)* - find a list of tags that are related to the input tag *List<Tag> getRelatedTags(String[] tags)* - find a list of tags that are related to all input tags
Types description:	Tag (name (acts as id), dbId, tag-frequency). Also see: *Tag*

Exceptions:	No exception has been identified yet.

Table 24 WP3_LocalTagCommunityDetector

Name:	**WP3_ExpertFinder**
Description:	Locate expert(s) for answering given question based on question content and assigned tags (if any). In general: find users that are expert in given topic. Topic is determined by analysis of input text (can be question, tags, answer or mixture of the above) Expert - (1) user that is associated with specific topic group, previously discovered by categorizing question/answer; and (2) that has the most significant contribution (in terms of documents and their importance) to the group.
Usage:	Final user putting a question to a system User enters text (can be question concatenated with tags, or other free text), system classifies it to find most applicable topic groups and generates a list of relevant experts for detected topics.
Input:	In the most general form method accepts text for analysis: *SortedSet<UserWithRank> findExperts(String text)* Following signatures are additional interfaces that allow to accept Lycos-related structures and extract appropriate text from then for further analysis *SortedSet<UserWithRank> findExperts(Question q)* *SortedSet<UserWithRank> findExperts(Question q, Set<Tag> tags)* *SortedSet<UserWithRank> findExperts(Answer a)*
Output:	SortedSet<UserWithRank> - sorted set of users with their rank. Set is sorted in descending order (most relevant users appear at the beginning) Rank is [0-1] - (1.0 = top expert, 0.0 = bad expert) In case no experts are found, empty list is returned. In case parameters are empty or null, an empty list is returned.
Exceptions:	No exceptions defined. In case if empty, null or incomplete input, an empty set it returned.

Table 25 WP3_ExpertFinder

Name:	**WP3_DetectLatentTopics**
Description:	Find relevant latent topics for given document: question or answer. Each latent topic has a list of related questions and answers, so indirectly, service allows to find documents in the system that are related to provided input.
Usage:	Given input (text in question, question and tags, or answer) system classifies it into previously calculated latent topics. Service uses text values provided within structures to perform classification. User should provide information which model should be used in calculations, otherwise a default model will be used. Multiple method signatures are provided to handle several structural types. Implementation of all calculations is in method with following signature: *detectLatentTopics(String modelName, String text)*
Input:	*SortedSet<LatentTopic> detectLatentTopics(String modelName, String text)* *Alternative method signatures:* *SortedSet<LatentTopic> detectLatentTopics(String modelName, Question q)* *SortedSet<LatentTopic> detectLatentTopics(String modelName, Question q, Set<Tag> tags)* *SortedSet<LatentTopic> detectLatentTopics(String modelName, Answer a)* *SortedSet<LatentTopic> detectLatentTopics(String text)* *SortedSet<LatentTopic> detectLatentTopics(Question q)* *SortedSet<LatentTopic> detectLatentTopics(Question q, Set<Tag> tags)* *SortedSet<LatentTopic> detectLatentTopics(Answer a)*
Types description:	*Question.* Important fields that are used in analysis: Question.headline and Question.text *Tag.* Important field that is used in analysis:

	Tag.name *Answer.* Important field that is used in analysis: Answer.text
Output:	SortedSet<LatentTopic> - sorted set of relevant latent topics to given input (question, answer, text, ...) Each latent topic is ranked: (1.0 = highly related, 0.0 = not related). Topics are ordered by their rank values in decreasing order In case system cannot find related question/answer, an empty set is returned. In case methods without a modelName are used, the default model is assigned internally. In case user provides incorrect model (empty, null or not existing), the default model will be used for calculations.
Exceptions:	No exceptions defined.

Table 26 WP3_DetectLatentTopics

10.4. 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:	CDL is a formal language enabling users to manage access rights in a flexible way. It consists of the following main structural elements: Authorization Basic Sets: A set of elements identifying an object relevant for identification. E.g. the users, the permission, the object, a timeslot, ...

	Authorization Property Set: A set of elements representing any property of an Auth Basic Set. E.g. a user group, a user belongs to. The last user login date (belonging to the basic set user). A creation date of a file (belonging to the basic auth set object). Explicit property assignments: The explicit assignment of a basic set element to a property set element. (E.g. assign user Alice to the user property element Group a). Tests on Property Sets: Does an element of a property set pass a test or not? (E.g. all elements of creation dates of property creation date earlier than 1.1.2009). Authorization Conditions: Logical AND-combinations of Tests on property sets. The CDL offers a formal language (like SQL) to control the above mechanisms. The CDL service is a wrapper service to enable direct method calls, which will be translated to CDL language and executed.
Required services:	Only CDL internal services: CDL Persistence Service, CDL Cache Service, CDL Core Service, CDL Parser Service, CDL Wrapper Service
Usage example:	ER: A
Method signature:	BasicAuthSet *static void CreateBasicAuthSet(String name);* *static void DeleteBasicAuthSet(String name);* *static void CreateBasicAuthSetElement(String basicauthsetname, String authsetelement);* *static void DeleteBasicAuthSetElement(String basicauthsetname, String authsetelement);* PropertyAuthSet *static void CreatePropertyAuthSet(String name);* *static void DeletePropertyAuthSet(String name);* *static void CreatePropertyAuthSetElement(String propertyauthset, String name);* *static void DeletePropertyAuthSetElement(String propertyauthset, String name);*

	BasicAuthSet - PropertyAuthSet Assignment *static void LinkAuthSet(String basicauthset, String propertyauthset);* *static void UnlinkAuthSet(String basicauthset, String propertyauthset);* *static void LinkAuthSetElement(String basicauthset, String basicauthsetelement, String propertyauthset, String propertyauthsetelement);* *static void UnlinkAuthSetElement(String basicauthset, String basicauthsetelement, String propertyauthset, String propertyauthsetelement);* Tests *static void CreateAuthSetTest(String name, String authsetname, String operator, String relator);* *static void DeleteAuthSetTest(String name);* Authorization Condition *static void CreateAuthCondition(String name, Tests [] tests);* *static void DeleteAuthCondition(String name);*
Types description:	BasicAuthSet: Set of BasicAuthSetElements of one BasicAuthSet type, e.g. Users, Permissions, Objects, Timeslots. BasicAuthSetElement: Element of one BasicAuthSet type, e.g. user Alice, permission read, timeslot "on all mondays". PropertyAuthSet: Set of PropertyAuthSetElements defining representing a property of a BasicAuthSet. E.g. usergroups, object creation dates. PropertyAuthSetElement: Element of a PropertyAuthSet, e.g. usergroup A, creation date 1.1.2009 AuthSetTest: Test on BasicAuthSets and PropertyAuthSets, mapping an element to TRUE or FALSE. E.g. usergroup=A?, object creation date < 1.1.2009?. It consists of an BasicAuthSet or PropertyAuthSet, an operator and a relator. Authorization Condition: A combination of tests, allowing access. E.g. (usergroup=A, object creation date<1.1.2009, permission=read). --

Exceptions:	tbd.

Table 27 WP4_Community_Design_Language

Community Analysis Tool

The classes of the Community Analysis Tool make use of Java generics.

- O: class to identify a vertex (String, Integer)
- V: class to represent a vertex, usually must be derived from SNAVertex
- E: class to represent an edge

Name:	**WP4_ClosenessCentrality**
Description:	Calculates closeness centrality for (un)weighted, (un)directed social networks
Required services:	WP4_AllPairsSPData (internal WP4 service)
Usage example:	estimate members reachability in network, *ER*: high closeness centrality users can reach other user very fast
Method signature:	*ClosenessCentralityScorer(SNAGraph graph, AllPairsSPData spData, boolean averaging)* - create new instance of ClosenessCentrality *double getVertexScore(V vertex)* - return closeness centrality of specified vertex
Types description:	*SNAGraph* - graph implementing the interfaces SNAGraph, UndirectedGraph (UndirectedSNAGraph, DBUndirectedSNAGraph) *AllPairsSPData* - class that calculates the shortest paths between all pairs
Exceptions:	None.

Table 28 WP4_ClosenessCentrality

Name:	**WP4_InverseDistanceClosenessCentrality**
Description:	Calculates inverse distance closeness centrality for (un)weighted, (un)directed social networks
Required	WP4_AllPairsSPData (internal WP4 service)

services:	
Usage example:	estimate members reachability in network, ER: high id closeness centrality users can reach other user very fast
Method signature:	*InverseDistanceClosenessCentrality(SNAGraph<O,V,E> graph, Distance<V> distance, boolean averaging)* - create new instance of InverseDistanceClosenessCentrality *double getVertexScore(V vertex)* - return inverse distance closeness centrality of specified vertex
Types description:	*Distance* - class that calculates the geodesic distance of two vertices, AllPairsSPData implements Distance interface
Exceptions:	None.

Table 29 WP4_InverseDistanceClosenessCentrality

Name:	**WP4_BetweennessCentrality**
Description:	Calculates between's centrality for (un)weighted, (un)directed social networks
Required services:	WP4_AllPairsSPData (internal WP4 service)
Usage example:	estimate members influence on information flow in network
Method signature:	*SNABetweennessCentrality(SNAGraph graph, AllPairsSPData spData, boolean averaging)* - create new instance of BetweenessCentrality *evalute()* - starts calculation *Map<SNAVertex<Integer>,Number> getVertexRankScores()* - retrieves Map<SNAVertex<Integer>,Number> which maps vertices to centrality values
Types description:	*AllPairsSPData* - class that calculates the shortest paths between all pairs
Exceptions:	None.

Table 30 WP4_BetweennessCentrality

Name:	WP4_NewmanClustering
Description:	Clusters a unweighted, undirected social network
Required services:	none
Usage example:	Cluster ego-network of a community member to group his/her contacts
Method signature:	*List<List<V>> cluster(SNAGraph<O,V,E> graph, Set<V> vertices)* - clusters the subgraph of the specified set of vertices and returns a list of clusters (cluster = list of vertices)
Types description:	*SNAGraph* - graph implementing the interfaces SNAGraph, UndirectedGraph (UndirectedSNAGraph, DBUndirectedSNAGraph) *Set<V>* - V is generic: V extends SNAVertex, SNAVertex parent class for all vertices, provides access to ID and connected component info
Exceptions:	IllegalArgumentException: if graph is directed

Table 31 WP4_NewmanClustering

Name:	WP4_SNAGraphStatistics
Description:	Provides network-level measures
Required services:	none
Usage example:	measures for community administrators to evaluate community
Method signature:	double getDiameter(SNAGraph<O,V,E> graph, Distance<V> distance) - *returns the diameter of the network* *double getAverageDegree(SNAGraph<O,V,E> graph, Transformer<E,Double> transformer)* - returns the average degree of all vertices *double getDensity(SNAGraph<O,V,E> graph)* - returns density of the graph
Types description:	Transformer: Interface from Apache Commons, in this case assigns an edge weight to an edge (usually the internal weight value of an edge object)

Exceptions:	---

Table 32 WP4_SNAGraphStatistics

Name:	**WP4_AffiliationNetworkMeasures**
Description:	Provides a set of measures for affiliation networks
Required services:	None.
Usage example:	provide search for similar users based on connections to event (that could be tags, images etc.)
Method signature:	*int getSharedEventsCount(SNAAffiliationGraph<O,V,E> graph, V actor1, V actor2)* - returns the number of events two actors have in common *List<V> getSharedEvents(SNAAffiliationGraph<O,V,E> graph, V actor1, V actor2)* - return the events two actors have in common *double eventSimilarity(SNAAffiliationGraph<O,V,E> graph, V actor1, V actor2)* - calculates the similarity of two actors based on their shared events.
Types description:	SNAAffiliationGraph: bipartite graph with the two distinct set of vertices actors and events
Exceptions:	---

Table 33 WP4_AffiliationNetworkMeasures

10.5. 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:

The services of Group Management allow for maintenance and use of distributed defined groups within the system. This enables the inclusion not only of members of other professional organizations or authorities, but also of private organisations into the virtual organisation created in order to respond to the emergency. In the CSG use case the provided services allow for reuse of social structures previously defined in other contexts.

Task Management services provide functionality necessary to define tasks that, later on, 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. What is more, they include services and data structures provided by other WPs, e.g. uploaded content or analysis results, that can be used for executing the task. Explicit definition of tasks in the system can improve the collaboration of the emergency responders.

Event-based Incident Log services and LogMerger services provide means for organizing knowledge and content, in terms of events within logs. The events span along multiple dimensions like time and location. Moreover, functions are provided to summarize logs. In case of 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 can be represented in a distributed manor. In addition to this it 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 repre-

	sented 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 signature:	*createGroup(URI adminGroupID, String name, Enumeration{OPEN, INVITATION_NEEDED, SINGLE_ADMIN} adminMode)* - return URI groupID - create a new group *addUserToGroup(URI userID, URI groupID)* - return boolean - adds user to group, returns true if successful *addGroupToGroup(URI childGroupID, URI parentGroupID)* - return boolean - adds the child group to the parent group, returns true if successful *removeUserFromGroup(URI userID, URI groupID)* - return boolean - removes user from group, returns true if successful *removeGroupFromGroup(URI childGroupID, URI parentGroupID)* - return boolean - removes child group from parent 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 34 WP5_GroupManagement

Name:	WP5_TaskManagement
Description: General purpose, type, benefit and approach	General Description: This service provides managing of tasks, i.e., defining, sharing, assigning, and deleting them. It allows the users to interact and collaboratively process the tasks, observing task progress, etc. Type: external service (communicates with other WPs) Interplay with other WPs: Individual users use task management for ER and CSG use case (WP1) Interplay with WP3 and WP4 by assignment of tasks to users and groups supported by GroupManagement service above Resources such as media assets analysed in WP2 can be assigned to tasks (WP2)
Method signature:	*createTask(String taskDescription)* - return URI taskID - creates a new task *assignTaskToUserOrGroup(URI taskID, URI groupOrUserId)* - return boolean - assigns a task to a user or group *removeTaskFromUserOrGroup(URI taskID, URI groupOrUserId)* - return boolean - removes the assignment of a task to a user or group *getTaskExecutors(URI taskID)* - return Set<URI> executors - returns a list of users and groups assigned to task *addResourceToTask(URI resourceID, URI taskID)* - return boolean - assigns a resource such as an image to a task *getResources(URI taskID)* - return Set<URI> resourceIDs - returns the resources assigned to a task *removeResources(URI resourceID, URI taskID)* - return boolean - removes a resource from a task *addSubtask(URI childTaskID, URI parentTaskID)* - return boolean - adds a subtask *removeSubtask(URI childTaskID, URI parentTaskID)* - return boolean - removes a subtask *getSubtasks(URI parentTaskID)* - return Set<URI>

	childTaskIDs - returns a list of subtasks *deleteTask(URI taskID)* - return boolean - deletes a task *getTaskDescription(URI taskID)* - return String taskDescription - returns the task description *setTaskDescription(URI taskID, String taskDescription)* - return boolean - sets the task description *getTaskStatus (URI taskID)* - return Enumeration{NOT_STARTED,IN_PROCESS,FINISHED} taskStatus - returns the current task status *setTaskStatus (URI taskID, Enumeration{NOT_STARTED,IN_PROCESS,FINISHED} taskStatus)* - return boolean - set the current task status *duplicateTask(URI inputTaskID)* - return URI outputTaskID - creates a duplicate of a task together with all subtasks
Types description:	URI = uniform resource identifier TaskTemplate = A taskTemplate is a list of tasks and is provided by WP5. It is created as follows: 1. a super-task is created, 2. a list of subtasks to this task is created and which is actually the task template. It is provided implicitly by the different methods defined above.
Exceptions:	all: task with taskID does not exist all: insufficient rights to perform this action createTask: task with taskDescription already exists assignTaskToUserOrGroup: user/group does not exist removeTaskFromUserOrGroup: user/group does not exist addSubtask: subtask does not exist addSubtask: subtask already added before removeSubtask: subtask does not exist addResourceToTask: resource does not exist addResourceToTask: resource already added before

Table 35 WP5_TaskManagement

Name:	**WP5_IncidentLog**

Description: General purpose, type, benefit and approach	General Description: This service provides for managing events that can happen in an emergency incident by writing them down in an incident log. The service allows for adding events and searching them. Type: external service (communicates with other WPs) Interplay with other WPs: Adding entries to incident log from professional users from WP1. Communicating these entries with ER entities (via WP1). Analysis of entered event descriptions via textual analysis (WP2) for extracting event information such as time, space, persons, causes, correlations, etc. (WP2)
Method signature:	*createIncidentLog(String incidentName)* - return URI incidentID - creates a new incident and returns its identifier *deleteIncidentLog(URI incidentLogID)* - return boolean - deletes an incident *addEntryToIncidentLog(Event event, Date timeWhenAdded, String entryDescription, URI userOrGroupID, URI incidentLogID)* - return boolean - adds an entry to the log *getEntries(URI incidentLogID)* - return Set<Entries> entries - returns the entries of the incident log *getIncidentLogs()* - return Set<URI> entries - returns list of existing incident logs
Types description:	URI = immutable universally unique identifier Entry = (Event event, Date timeWhenAdded, String entryDescription, URI userOrGroupID) Event = as defined by the WP5 event model
Exceptions:	all: insufficient rights to perform this action deleteIncidentLog: log with incidentLogID does not exist

Table 36 WP5_IncidentLog

Name:	**WP5_LogMerger**

Description:	The service merges a set of event logs referring to the same incident into a single log and allows targeted searching within the merged log. In reality, this service is useful in the post-event stage of the ER use case, when the logs which have been created by the ER personnel are collected and controlled (for review and audit purposes). Frequently, there are multiple log files created by different people that refer to the same emergency event. This service 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 log-Files. Incorrect log file format: When the service cannot parse correctly one of the input log files.

Table 37 WP5_LogMerger

10.6. WP6 services

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

Name:	**WP6_DataStorage**
Description:	Provides access to the WKI Data Storage. Provides storage and retrieval functionalities (of both meta-data and content) also in form of SparQL queries. This is an internal service.
Usage:	This service should be used by other services. For example: • WP2_SpeachIndex service after examining an audio file calls WP6_DataStorage.addMetadataToDucoment(...) in order to save the created speechIndexURI to the WKI DS, • WP1_SearchKB executes a SparQL query using WP6_DataStorage.executeQuery(...) to retrieve required data. • A service responsible for metadata extraction from text files executes SparQL query using WP6_DataStorage.executeQuery(...) to find all text documents with certain properties.
Required	None

Services:	
Method signature:	*String executeQuery(String sparqlQuery)* - executes sparql query on KB and returns result as XML *Map<String, List<String>> getWkiDocumentMetadata(Long documentId)* - returns document metadata. Result is a map as described in "Types description". *void storeDocument(IWkiDocument document)* – adds new document (content + metadata) to storage. The IWkiDocument is created by other internal WP6 service (during creation of IWkiDocument its ID is generated). *InputStream getDocumentContent(Long documentId)* - returns content (InputStream) of a document with given ID. *void addMetadataToDocument(Long documentId, Map<String, List<String>> metadata)* - adds metadata to specified document. Doesn't change the existing metadata, simply adds new. Metadata is a map as described in "Types description".
Types description:	IWkiDocument document - represents content (file) and its metadata. IWkiDocument contains: • ID, • metadata map (where key is a metadata type - predicate in RDF triple - and value is a list of all values), • list of URIs which points to the content.
Exceptions:	WkiDataStorageException - generic exception wrapper for all exceptions thrown.

Table 38 WP6_DataStorage

Name:	**WP6_ExternalDataStorageService**
Description:	Provides access to the WKI Data Storage via web services.
Usage:	• Emergency Response application uploads file via Web service.

	• Consumer Group application retrieves meta-data of a document with given ID via Web service.
Required Services:	WP6_DataStorage (internal service).
Method signature:	*String executeQuery(String sparqlQuery)* - executes sparql query on WKI DS and returns result as XML. *Long storeNewContent(DataHandler holder, String fileName)* - adds new content (file) with specified name to the WKI DS. Returns ID of a created document. *void addMetadataToDocument(Long documentId, String metadataType, String metadataValue)* - adds metadata represented by meta-dataType,metadataValue pair to document with specified ID.
Types description:	DataHandler - java type javax.activation.DataHandler
Exceptions:	DataStorageException- generic exception wrapper for all exceptions thrown.

Table 39 WP6_ExternalDataStorageService

11. Annex B – Guidelines

11.1. Development Conventions

Date: 23.03.2009
Author(s): Rene Wagner, Andrzej Boruch
Organisation(s): SMIND, LYC
WP/Task/Subtask: WP 6/T6.2
Version: 0.1

History

Ver	Author	Description
0.1	Rene Wagner, Andrzej Boruch	First version

11.1.1. Naming conventions

Uniform naming is a very important aspect of clear project integration and consolidation. In WKI System all sources should be bound to the namespace:

eu.weknowit

Each workpackage should bind all its services to the subspace:

eu.weknowit.{Workpackagename}

Every workpackage is using its userfriendly fulltextname as mapped in the following:

- WP1: personal
- WP2: media
- WP3: mass
- WP4: social
- WP5: organisational

Every service that does not fit into a certain workpackage domain will be bound to:

eu.weknowit.common

This leads to the following namespaces:

- eu.weknowit.common
- eu.weknowit.personal
- eu.weknowit.media
- eu.weknowit.mass
- eu.weknowit.social
- eu.weknowit.organisational

Any service gets its own package, i.e.

eu.weknowit.mass.communitydetector

11.1.2. Code conventions

We will follow the Java code conventions as documented here:

http://java.sun.com/docs/codeconv/

11.1.3. Documentation conventions

For documentation purposes please use code comment documentation like explained in the section 5.2 of the code conventions listed above or here:
http://java.sun.com/javadoc/writingdoccomments.
If the documentation has to be published i.e. on wikipages use the **javadoc** tool for transformation.

11.2. How to install Java & Maven

Date:	23.03.2009
Author(s):	Tomasz Kaczanowski, Andre Skusa
Organisation(s):	SMIND, LYC
WP/Task/Subtask:	WP 6/T6.2
Version:	0.5

History

Ver	Author	Description

0.1	Tomasz Kaczanowski, Andre Skusa	First version
0.2	Tomasz Kaczanowski	Ant installation described. More links added. Code format changed.
0.3	Tomasz Kaczanowski	JAVA_HOME
0.4	Tomasz Kaczanowski	Java version updated to 1.6. Information on .m2 folder on Windows Vista added
0.5	Tomasz Kaczanowski	Nexus repository address change.

11.2.1. 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.

11.2.2. 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.

11.2.3. 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.

11.2.4. Installation of Maven

If you have already Maven 2.0.9 installed you can skip this section (but you should read the next section - `settings.xml` file!). 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.

settings.xml file

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

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

And this file should contain this 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">

  <!-- your local repository - please avoid folder with
whitespaces !!! -->
    <!-- uncomment the line appropriate to you operating system-->
    <!-- windows example -->
    <!-- <localRepository>c:\.m2\repository</localRepository> -->
    <!-- linux example -->
    <!-- <localRepository>/home/mynickname/repo</localRepository>
-->

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

<url>http://mvn.weknowit.eu/nexus/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 (see also comments in the file above):

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

After the changes, the `settings.xml` file 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:\.m2\repository</localRepository>

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

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

11.2.5. 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

11.3. How to create Maven project

Date: 16.02.2009

Author(s): Łukasz Baran, Tomasz Kaczanowski, Andre Skusa

Organisation(s): SMIND, LYC

WP/Task/Subtask: WP 6/T6.2

Version: 0.4

History

Ver	Author	Description
0.1	Łukasz Baran, Tomasz Kaczanowski, Andre Skusa	First version
0.2	Łukasz Baran, Tomasz Kaczanowski	Code format changed. Xml configuration files moved to META-INF/spring (picture updated). Archetype version updated to 1.5. Changes in chapter "Build your project" section.
0.3	Tomasz Kaczanowski	Changes in chapter "Build your project" section - `install` explained.
0.4	Tomasz Kaczanowski	Section 5 - "IDE Integration" - enhanced.

11.3.1. Introduction

Rationale

The services developed by the WPs, must be prepared, in order to be used in the WKI System. The creation of a Maven project is the first step within this preparation. This guideline describes this process.

There are many ways to create a Maven project. The one described here generates a new project which have some configuration files created and filled with values suitable for the creation of WeKnowIt services.

Prerequisites

- Java & Maven installed and configured - as described in *How to install Java & Maven* guideline.
- Maven repository and Nexus login/pass - as described in *How to install Java & Maven* guideline.
- Optionally: you have some source code of your service.

Outcome

Project that can be compiled by using Maven.

11.3.2. Create a project

Let us start with creating an empty Maven project. To do it you should execute the command printed below in your shell (one line !).

```
mvn archetype:generate -DarchetypeGroupId=eu.weknowit.common -
DarchetypeArtifactId=weknowit-bundle-pom -DarchetypeVersion=1.5
```

You'll be asked few questions:

```
Define value for groupId:
```

answer with `eu.weknowit.YOUR_WP_NAME` for example `eu.weknowit.social`

```
Define value for artifactId:
```

answer with any name you like (please don't use spaces or any other weird characters), for example - `graph-service`

```
Define value for version:  1.0-SNAPSHOT: :
```

simply press ENTER (the default value will be used)

```
Define value for package:  eu.weknowit.social: :
```

by default it will have the same value as groupId, which you provided earlier. If this value conforms to the WKI code conventions,

than simply press ENTER, if not - enter a valid package name here. In our case `eu.weknowit.social.graph` should be entered.

Now you'll be asked to confirm the entered values:

```
Confirm properties configuration:
groupId: eu.weknowit.social
artifactId: graph-service
version: 1.0-SNAPSHOT
package: eu.weknowit.social.graph
Y: :
```

Type `Y` and press ENTER.

The rest of this text assumes that you entered values as stated below (no value means that nothing was entered):

```
Define value for groupId: eu.weknowit.social
Define value for artifactId: graph-service
Define value for version:  1.0-SNAPSHOT: :
Define value for package:  eu.weknowit.social: :
eu.weknowit.social.graph
Confirm properties configuration:
groupId: eu.weknowit.social
artifactId: graph-service
version: 1.0-SNAPSHOT
package: eu.weknowit.social.graph
Y: : y
```

After Maven finishes, you will find a new folder `graph-service` generated.

Now go into the root folder of your project (i.e., execute the command "cd graph-service"). In the rest of this text it will be assumed that you execute all the commands from this root folder.

The structure of folders and files of graph-service project should look like this:

```
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- eu
    |   |       `-- weknowit
    |   |           `-- social
    |   |               `-- graph
    |   |                   `-- readme.txt
    |   `-- resources
    |       |-- META-INF
    |       |   `-- spring
    |       |       |-- bundle-context-osgi.xml
    |       |       `-- bundle-context.xml
    |       `-- log4j.properties
    `-- test
        `-- java
            `-- eu
                `-- weknowit
                    `-- social
```

```
                        `-- graph
                            `-- readme.txt
```

The purpose of `bundle-context-osgi.xml` and `bundle-context.xml` files will be explained in successive guidelines.

11.3.3. Development

Add your own code

If you already have some source code, simply copy it into `src/main/java/eu/weknowit/social/graph`.

If you don't have any source code yet, start creating it in `src/main/java/eu/weknowit/social/graph` folder.

Add dependencies to pom.xml

If your code uses any libraries you need to tell Maven which ones these are. For every library your code is using add `<dependency>` to the `<dependencies>` section in the project's `pom.xml`, e.g.:

```
<dependency>
   <groupId>x</groupId>
   <artifactId>y</artifactId>
   <version>z</version>
</dependency>
```

The next sections will help you to choose the right values for groupId, artifactId and version.

GroupId, artifactId and version

If you know exactly what library you use (because its name contains all necessary information, as e.g. "hibernate-3.2.6.ga.jar") you should follow these steps:

- visit **http://mvnrepository.com**
- search for "hibernate" - you'll get a list of names of artifacts
- choose the one which name resembles the name of your jar file most (in this case it will be "org.hibernate » hibernate")
- now you will be presented with a list of various versions of this library - once again choose the one that resembles the name of your jar file most (in this case it will be "3.2.6.ga")
- under section "POM Dependency" you'll see a snippet of xml which looks like this:

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate</artifactId>
   <version>3.2.6.ga</version>
</dependency>
```

- now simply copy it and paste it into `<dependencies>` section of your `pom.xml`

Example

Let's assume that your service uses two jars - commons-lang-2.4.jar and hibernate-3.2.6.ga.jar. Using **http://mvnrepository.com** you will discover groupIds and artifactIds of both. The whole `<dependencies>` section of your `pom.xml` should look like this:

```
<dependencies>
    <dependency>
        <groupId>commons-lang</groupId>
        <artifactId>commons-lang</artifactId>
        <version>2.4</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate</artifactId>
        <version>3.2.6.ga</version>
    </dependency>
<dependencies>
```

Library not found

If your code uses some .jar file that is impossible to identify or that can be identified but it can not be found in any Maven repository, than you need:

- choose some arbitrary groupId, artifactId and version for it
- change the name of the jar file so that it reflects the chosen values in a standard Maven way (artifactId-version.jar)
- upload it to Lyc Maven repository
- add `<dependency>` section with exactly the same groupId, artifactId and version that you have chosen for this jar

Example

Let's assume that your service uses a jar file `math.jar` for math calculations, which is immposible to identify. Following the previously described steps you should:

- choose some arbitrary values: groupId=org.math, artifactId=math, version=1.0
- rename `math.jar` it to `math-1.0.jar`
- upload it to Lyc Maven repository
- add this `<dependency>` section to `<dependencies>` section of your `pom.xml`:

```
    <dependency>
        <groupId>org.math</groupId>
        <artifactId>math</artifactId>
```

```
        <version>1.0</version>
    </dependency>
```

11.3.4. Build your project

Execute the command:

```
mvn clean package
```

The artifact (compiled project in form of a `.jar` file) will be created in the `target` directory.

The first execution of this command may take some time, because Maven will download all the plugins and libraries needed - the good thing is, that it happens only once.

If you need the artifact not only to be created, but also to be placed in your local Maven repository (which is a common requirement, if you want other projects to use this `.jar`), execute command:

```
mvn clean install
```

Warnings

You will see many warnings during the execution, e.g.:

```
[WARNING] Warning building bundle eu.weknowit.social:graph-
service:bundle:1.0-SNAPSHOT : Neither Export-Package nor Private-
Package is set, therefore no packages will be included
[WARNING] Warning building bundle eu.weknowit.social:graph-
service:bundle:1.0-SNAPSHOT : Did not find matching referal for !*
[WARNING] Warning building bundle eu.weknowit.social:graph-
service:bundle:1.0-SNAPSHOT : Did not find matching referal for
javax.jws.*
...
```

You can safely ignore these warnings - they are here, because the `maven-bundle-plugin` declared in `pom.xml` file is not properly configured yet. The configuration of this plugin will be explained in further guidelines.

Outcome

The target folder after build should look like this now:

```
`-- target
    |-- classes
    |   |-- META-INF
    |   |   |-- MANIFEST.MF
    |   |   `-- spring
    |   |       |-- bundle-context-osgi.xml
    |   |       `-- bundle-context.xml
    |   `-- log4j.properties
    |-- graph-service-1.0-SNAPSHOT-sources.jar
    `-- graph-service-1.0-SNAPSHOT.jar
```

11.3.5. IDE integration

Eclipse

If you are going to develop the project under Eclipse, you can import this project as it is described in this document: **http://maven.apache.org/guides/mini/guide-ide-eclipse.html#Maven_2_repository** (see sections "Maven2 repository" and "Simple projects" - the rest is not important).

You might also be interested in m2eclipse plugin (**http://m2eclipse.codehaus.org/**) which helps to integrate Maven with Eclipse IDE.

More on Eclipse integration can be found in the "Maven - The Definitive Guide" book - **http://www.sonatype.com/books/m2eclipse-book/reference/**.

Other IDEs

Please check documentation of your IDE.

11.3.6. Links

Further information can be found here:

- **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://www.javaworld.com/javaworld/jw-12-2005/jw-1205-maven.html** - An introduction to Maven 2 - JavaWorld

11.4. How to install the WKI System

Date: 18.03.2009
Author(s): Rafał Janik, Tomasz Kaczanowski
Organisation(s): SMIND
WP/Task/Subtask: WP 6/T6.2
Version: 0.2

History

Ver	Author	Description
0.1	Rafał Janik, Tomasz Kaczanowski	First version
0.2	Rafał Janik, Tomasz Kaczanowski	Turning off debug mode section. Troubleshooting section added.

11.4.1. Introduction

Rationale

The WKI System must be installed on local machine, so developers are able to verify if their services are compatible with WKI architecture.

Prerequisites

Java & Maven installed and configured - as it is described in *How to install Java & Maven* guideline.

Outcome

The WKI System installed.

11.4.2. 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 appriopriate for your operating system:

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

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

11.4.3. Files and folders in WKI_HOME

This is how `WKI_HOME` looks like:

```
|-- bin
|-- configs
|-- data
|-- demos
|-- deploy
|-- etc
|-- examples
|-- lib
|-- licenses
|-- system
|-- LICENSE.txt
|-- NOTICE.txt
|-- README.txt
|-- RELEASE-NOTES.txt
|-- wki-readme.txt
|-- wki-release-notes.txt
```

The most important files and directories are:

- `bin` - startup/shutdown scripts
- `configs` – in this directory will be copied bundle's properties files
- `data` - all runtime data (including deployed bundles and log files) can be found here
- `deploy` - every bundle copied to this directory will be deployed to the WKI System
- `wki-release-notes.txt` - information about this particular release

11.4.4. Turn off full-information mode

By default, WKI System prints out many informations and logs which are useful for development. If you don't need them, you can turn them off by editing `config.properties` file in `etc` directory:

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

Open this file with text editor. At the end of it there is line:

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

Now comment this line using '#':

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

After this all additional informations will be hidden.

11.4.5. Starting WKI System

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

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

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

  - then run the startup script

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

- Windows

```
wki-start.bat
```

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

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

```
 ServiceMix (1.0.2.0-fuse)

Type 'help' for more information.
------------------------------------------------------------
servicemix>
```

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

11.4.6. Stopping WKI System

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

```
exit
```

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

11.4.7. Cleaning of WKI System

If you develop new services, you will likely need to remove the older versions before deploying new ones to the WKI System. In order to do this, some manual cleaning will be needed.

IMPORTANT - first stop the WKI System before performing these steps !

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

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

11.4.8. Troubleshooting

Address already in use

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

To resolve this problem port 1099 must be freed up before starting proper instance.

If it is impossible the ServiceMix RMI port number must be changed in the etc/org.apache.servicemix.management.properties file by changing the rmiRegistryPort property to a port number that is available.

11.4.9. Links

The latest version of the WKI System:

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

Further information can be found here:

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

11.5. How to create OSGi service using Spring Dynamic Modules

Date: 18.03.2009

Author(s): Łukasz Baran, Rafał Janik, Tomasz Kaczanowski, Andre Skusa

Organisation(s): SMIND, LYC

WP/Task/Subtask: WP 6/T6.2

Version: 0.2

History

Ver	Author	Description
0.1	Rafał Janik, Tomasz Kaczanowski, Łukasz Baran, Andre Skusa	First version
0.2	Rafał Janik	ApplicationContextListenerImpl code changed.

Abbreviations and Acronyms

IOC	Inversion of Control
OSGi	a Java-based service platform
POJO	Plain Old Java Object
Spring DM	Spring Dynamic Modules

11.5.1. Introduction

Rationale

Services working in the WKI System will play two roles:

- service providers - which allows other services use their functionalities,
- service clients - which use functionalities provided by other services.

This guideline explains how to develop both kind of services using Spring Dynamic Modules.

Prerequisites

- Java and Maven installed - see *How to install Java & Maven* guideline,
- The WKI System installed and ready to be started - see *How to install the WKI System* guideline,
- basic understanding of IOC/Dependency Injection frameworks in general and Spring injection mechanism.

Outcome

Two services will be created:

- service provider, which exposes some simple functionality,
- client service, which uses functionality exposed by service provider.

The services will be deployed to the WKI System.

11.5.2. How to create a service provider

Service provider exposes some of its functionalities via OSGi registry, so other services can use these functionalities.

Following tasks needs to be performed:

- create a Maven project,
- write code (functionalities that this service provider will expose),
- configure service,
- build service and deploy it.

Create a Maven project

Firstly, we should create a Maven project. Follow the 'How to create Maven project' guideline. When creating project, use following parameters (bolded):

- `groupId=`**`eu.weknowit.tutorial.osgi`**
- `artifactId=`**`osgi-wki-greetingservice`**

Before we go any further, let us take a look at how the project will look like **at the very end of our work** - this picture will help you to see where exactly each file should be placed:

```
|   pom.xml
|
+---src
   +---main
   |   +---java
   |   |   \---eu
   |   |       \---weknowit
   |   |           \---tutorial
   |   |               \---osgi
   |   |                   |   IGreetingService.java
   |   |                   \---impl
   |   |                           GreetingServiceImpl.java
   |   \---resources
   |       \---META-INF
   |           \---spring
   |                   greeting-service-context.xml
   |                   greeting-service-osgi-context.xml
   \---test
        \---java
```

Project files description:

- `greeting-service-context.xml`, `greeting-service-osgi-context.xml` - Spring DM configurations files
- `IGreetingService.java` - greeting service interface
- `GreetingServiceImpl.java` - greeting service implementation

Write service provider code

Provider contains simply Java interfaces and class files (POJOs). Let us start with creating provider interface and than an implementation of this interface.

The interface and it's implementation will be created in different packages. This will allow to expose only the interface and not the implementation - which is a good, object-oriented way of doing things.

Create service interface

Create `IGreetingService.java` interface in `eu.weknowit.tutorial.osgi` package. It's a very simple interface, with only one method.

```
package eu.weknowit.tutorial.osgi;
public interface IGreetingService {
	public String sayGreeting(String s);
}
```

Create service implementation

Create new package `eu.weknowit.tutorial.osgi.impl` and inside it create `GreetingServiceImpl.java` class which implements `IGreetingService` interface:

```
package eu.weknowit.tutorial.osgi.impl;
import eu.weknowit.tutorial.osgi.IGreetingService;
public class GreetingServiceImpl implements IGreetingService {
	public String sayGreeting(String s) {
		return "Welcome " + s;
	}
}
```

Configure service

To properly expose created classes following steps should be performed:

- creation of Spring configuration files inside *`META-INF/spring`* directory,
- update of `pom.xml` file by configuring `maven-bundle-plugin` and adding required dependencies.

Create Spring DM configurations

We want to enable other services to use method `sayGreeting`. We can do this by exposing this service method via the OSGi registry.

The simplest way to do this is to provide configuration with Spring DM configuration files. Changes will be done in *`META-INF/spring`*.

During the creation of Maven project, two files were generated in `META-INF/spring` directory - `bundle-context.xml` and `bundle-context-osgi.xml`. The first file - `bundle-context.xml` - will hold the configuration of Spring Beans. The second file - `bundle-context-osgi.xml` - will register these beans as OSGi services.

What is required to configure bundle properly is to:

- rename `bundle-context.xml` file to `greeting-service-context.xml` and place the following content in it (this file will contain only regular Spring beans definitions):

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:osgi="http://www.springframework.org/schema/osgi"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd">

<bean name="greetingService"
class="eu.weknowit.tutorial.osgi.impl.GreetingServiceImpl" />
</beans>
```

The file starts with definition of required schemas and namespaces. Then, `greetingService` bean is declared (which represents implementation of `IGreetingService` interface). The important thing here is that the value of `class` attribute is equal to the fully qualified class name of `GreetingServiceImpl` class.

- rename `bundle-context-osgi.xml` file to `greeting-service-osgi-context.xml`. It should contain OSGi schema beans:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd">
  <osgi:service id="greetingServiceOsgi"
                ref="greetingService"

interface="eu.weknowit.tutorial.osgi.IGreetingService" />
</beans>
```

After schemas and namespaces definitions `osgi:service` component is declared. This component will register a given bean (`greetingService` in this example) in OSGi registry. As we can see this clause has three attributes:

- `id` - unique bean ID
- `ref` – a reference to the Spring Bean that is intended to be exported (in this example it refers to `greetingService` bean defined earlier in `greeting-service-context.xml` file),
- `interface` - fully qualified interface name.

Configure pom

Next step is to configure `maven-bundle-plugin` and add required dependencies. Edit `pom.xml` file and put the following configuration of `maven-bundle-plugin` to `plugins` section:

```
<plugins>
 <plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <version>${felix-plugin-version}</version>
  <extensions>true</extensions>
  <configuration>
   <instructions>
```

```
     <Bundle-SymbolicName>greeting-provider-service</Bundle-
SymbolicName>
     <Import-Package>
         javax.activation.*,
         org.apache.servicemix.common,
         org.apache.servicemix.common.osgi,
         org.osgi.framework.*,
         org.springframework.osgi.config,
         org.springframework.osgi.context.event,
         org.springframework.osgi.service.exporter,
         org.springframework.osgi.service.importer,
         org.springframework.beans.factory.config,
         org.springframework.context,
         org.springframework.util,
          !*
      </Import-Package>
      <Private-Package>eu.weknowit.tutorial.osgi.impl</Private-
Package>
      <Export-Package>eu.weknowit.tutorial.osgi</Export-Package>
      <Spring-Context>*;create-asynchronously:=true;wait-for-
dependencies:=true;timeout:=120;publish-context:=true</Spring-
Context>
   </instructions>
  </configuration>
 </plugin>
</plugins>
```

Declarations in italics type is required by Spring DM. Tags description:

- `Bundle-SymbolicName` - bundle name used by OSGi container
- `Import-Package` - all required packages separated by colons
- `Private-Package` - list of bundle private packages which shouldn't be exported to OSGi registry
- `Export-Package` - list of all bundle packages which will be exported to OSGi registry
- `Spring-Context` - some additional information for Spring DM

Two important things to notice here:

1. The `Export-Package` directive exports `eu.weknowit.tutorial.osgi` package (this is the package that the `IGreetingService` interface belongs to). It will make it visible to other packages via the OSGi registry.
2. The `eu.weknowit.tutorial.osgi.impl` is not exported, which means, that the implementation of the greeting service (`GreetingServiceImpl` class) will not be visible to other services.

Build bundle

In order to properly build project, invoke the following command:

```
mvn clean install
```

in a directory where `pom.xml` file is located.

The bundle - `osgi-wki-greetingservice-1.0-SNAPSHOT.jar` - will be generated in `target` directory of the project and installed in your local Maven repository.

For more information on building Maven projects please refer to *How to create a Maven project* guideline (*Build your project* section).

Deploy bundle to the WKI System

Please follow the instructions from *How to deploy OSGi bundle to the WKI System* guideline.

Result

The result of this guideline so far, is that the greeting service is deployed to the WKI System and is ready to perform it's duty if any client asks for it.

11.5.3. How to create client service

Client service makes use of the functionalities provided by service provider. In this case, we will build a simple client bundle that will invoke a method exposed by service provider.

The procedure of creating a client service is very similar to the creation of service provider.

Create a Maven project

First step is to create a separate Maven project for client service. Follow the 'How to create Maven project'. When creating project, use the following parameters:

- `groupId=`**`eu.weknowit.tutorial.client.osgi`**
- `artifactId=`**`osgi-wki-client-greetingservice`**

Before we go any further, let us take a look at how the project will look like **at the very end of our work** - this picture will help you to see where exactly each file should be placed:

```
|  - pom.xml
|
+---src
   +---main
   |   +---java
   |   |   \---eu
   |   |       \---weknowit
   |   |           \---tutorial
   |   |               \---client
   |   |                   \---osgi
   |   |                       |   - IGreetingClient.java
   |   |                       +---impl
   |   |                       |      - GreetingClientImpl.java
   |   \---resources
   |       \---META-INF
```

```
|           \---spring
|                   - greeting-client-service-context.xml
|                   - greeting-client-service-osgi-context.xml
\---test
    \---java
```

Create client code

Create service interface

Create client interface `IGreetingClient.java` in `eu.weknowit.tutorial.client.osgi` package.

The client needs to use some implementation of `IGreetingService`.

```
package eu.weknowit.tutorial.client.osgi;
import eu.weknowit.tutorial.osgi.IGreetingService;
public interface IGreetingClient {
     public abstract void setGreetingService(IGreetingService
greetingService);
     public abstract String askForGreeting(String name);
}
```

Create service implementation

Create implementation class (`GreetingClientImpl.java`) of this interface in package `eu.weknowit.tutorial.client.osgi.impl`:

```
package eu.weknowit.tutorial.client.osgi.impl;
import eu.weknowit.tutorial.osgi.IGreetingService;
import eu.weknowit.tutorial.client.osgi.IGreetingClient;
public class GreetingClientImpl implements IGreetingClient {
        private IGreetingService greetingService;
        public void setGreetingService(IGreetingService
greetingService) {
             this.greetingService = greetingService;
        }
        public String askForGreeting(String name) {
             return greetingService.sayGreeting(name);
        }
}
```

The purpose of the setter method - `setGreetingService` - is to receive reference to the implementation of `IGreetingService`. The `IGreetingService` implementation will be injected in a typical Spring way by xml configuration files (which will be described later).

The `askForGreetingMethod` simply makes use of the greeting service.

Spring DM configuration

Spring DM configuration files (in `META-INF/spring` directory) should also be prepared in a similar way to the service provider. This time we will start with OSGi configuration file:

- Rename `bundle-context-osgi.xml` file to `greeting-client-service-osgi-context.xml` and modify it, so that it would contain the following OSGi services configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd">
   <osgi:reference id="greetingServiceReferenceOsgi"

interface="eu.weknowit.tutorial.osgi.IGreetingService" />
</beans>
```

The `osgi:reference` directive tells OSGi registry to look for a service registered with the given interface. In this case, OSGi registry will look for a service implementing `IGreetingService` interface. OSGi registry will find such service, because we have already registered one (please refer to previous sections, where `greeting-service-osgi-context.xml` configuration file of service provider was described).

The other thing that will happen, is that `osgi:reference` will also create a normal Spring bean with the given ID (`greetingServiceReferenceOsgi`). We will use this bean soon.

- Rename `bundle-context.xml` file to `greeting-client-service-context.xml` and put the following Spring configuration in it:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd">
   <bean name="greetingClient"
class="eu.weknowit.tutorial.client.osgi.impl.GreetingClientImpl" >
       <property name="greetingService"
ref="greetingServiceReferenceOsgi" />
   </bean>
</beans>
```

There is only one bean defined:

- `greetingClient` – this bean represents client implementation code, we inject into this client the reference to `greetingServiceReferenceOsgi` bean using common Spring injection mechanism.

Configuration POM

The dependency on service provider should be added to dependencies section:

```
<dependency>
   <groupId>eu.weknowit.tutorial.osgi</groupId>
   <artifactId>osgi-wki-greetingservice</artifactId>
   <version>1.0-SNAPSHOT</version>
</dependency>
```

Configuration of `maven-bundle-plugin` is similar to the previous configuration in provider except a few differences:

- package `eu.weknowit.tutorial.osgi` should be imported (this package was exported in provider bundle - now we import it, to use methods exported by provider service),
- private and export packages sections will be different.

The configuration of `maven-bundle-plugin` should now look like:

```
<configuration>
 <instructions>
   <Bundle-SymbolicName>greeting-client</Bundle-SymbolicName>
     <Import-Package>
         eu.weknowit.tutorial.osgi,
         org.apache.log4j,
         org.apache.servicemix.common,
         org.apache.servicemix.common.osgi,
         org.osgi.framework.*,
         org.springframework.osgi.config,
         org.springframework.osgi.context.event,
         org.springframework.osgi.service.exporter,
         org.springframework.osgi.service.exporter.support,
         org.springframework.osgi.service.importer,
         org.springframework.osgi.service.importer.support,
         org.springframework.beans.factory.config,
         org.springframework.context,
         org.springframework.util,
         org.springframework.core,
         org.springframework.aop,
         !*
     </Import-Package>
     <Private-
Package>eu.weknowit.tutorial.client.osgi.impl</Private-Package>
     <Export-Package>eu.weknowit.tutorial.client.osgi</Export-
Package>
     <Spring-Context>*;create-asynchronously:=true;wait-for-
dependencies:=true;timeout:=120;publish-context:=true</Spring-
Context>
 </instructions>
</configuration>
```

Deploy bundle to the WKI System

Please follow the instructions from ‘How to deploy OSGi bundle to the WKI System’ guideline.

Result

After deploying of service provider and service client to the WKI System ...nothing will happen. All you can see is that the status of both bundles is set to `ACTIVE` (which is encouraging, but it doesn't really tell you if everything is fine). Both services are ready to work, yet there is no one which execute their methods - so nothing happens.

11.5.4. Getting some feedback - listeners

In the previous sections two services were created and deployed to the WKI System, with no visible results of their work. The client service, which was prepared to use the greeting service, was never invoked - thus, there was no visible output that would help developer to make sure that both services really work.

In this section we will find a way to invoke the client service so the output can be verified.

The idea listeners

During development, it's always good to have some “feedback”, some way to find out if the developed service works or not. Such a feedback can be achieved with the help of listeners.

The idea of the listeners is quite simple - they are registered during the deployment of the service, and they listen to the environment changes (thus the name) and are activated when some specific change in environment occurs. If is sounds cryptic, the following example will clarify the situation.

Some changes to the code created in sections 2 and 3 will be required.

Changes to osgi-wki-greetingservice

No changes needed.

Changes to osgi-wki-client-greetingservice

ApplicationContextListenerImpl

Create an `ApplicationContextListenerImpl.java` class in `eu.weknowit.tutorial.client.osgi.listeners` package:

```
package eu.weknowit.tutorial.client.osgi.listeners;
import org.springframework.context.ApplicationContext;
import
org.springframework.osgi.context.event.OsgiBundleApplicationContex
tEvent;
import
org.springframework.osgi.context.event.OsgiBundleApplicationContex
tListener;
import eu.weknowit.tutorial.client.osgi.IGreetingClient;
public class ApplicationContextListenerImpl implements
        OsgiBundleApplicationContextListener {
```

```
        private final static String CLIENT_BUNDLE_NAME =
"greeting-client";
        private String nameToTest;
        public void setNameToTest(String nameToTest) {
                this.nameToTest= nameToTest;
        }
        public void onOsgiApplicationEvent(
OsgiBundleApplicationContextEvent event) {
                if
(CLIENT_BUNDLE_NAME.equals(event.getBundle().getSymbolicName())){
                        ApplicationContext appCtx =
event.getApplicationContext();
                        IGreetingClient ff = (IGreetingClient)
appCtx.getBean("greetingClient");
                        System.out.println("SAY: " +
ff.askForGreeting(nameToTest));
                }
        }
}
```

This listener will be execute during the deployment of client service. It retrieves `IGreetingClient` from application context (`IGreetingClient` was registered by Spring DM configuration) and execute it's `askForGreeting` method. Then it prints out the value returned by this method.

This way, the cooperation between the service provider and client service will be tested.

Spring DM configuration

Add this snippet to the `greeting-client-service-context.xml` file:

```
  <bean name="greetingClientApplicationContextListener"
class="eu.weknowit.tutorial.client.osgi.listeners.ApplicationConte
xtListenerImpl">
      <property name="nameToTest" value="Traveller" />
  </bean>
```

The whole file should now look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd">
   <bean name="greetingClient"
class="eu.weknowit.tutorial.client.osgi.impl.GreetingClientImpl" >
       <property name="greetingService"
ref="greetingServiceReferenceOsgi" />
   </bean>
  <bean name="greetingClientApplicationContextListener"
class="eu.weknowit.tutorial.client.osgi.listeners.ApplicationConte
xtListenerImpl">
```

```
      <property name="nameToTest" value="Traveller" />
  </bean>
</beans>
```

File `greeting-client-service-osgi-context.xml` should also be updated with this snippet of xml configuration:

```
<osgi:service id="greetingClientOsgiApplicationContextListener"
                ref="greetingClientApplicationContextListener"
interface="org.springframework.osgi.context.event.OsgiBundleApplic
ationContextListener" />
The whole file should now look like:
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd">
   <osgi:service id="greetingClientOsgiApplicationContextListener"
                ref="greetingClientApplicationContextListener"
interface="org.springframework.osgi.context.event.OsgiBundleApplic
ationContextListener" />
   <osgi:reference id="greetingServiceReferenceOsgi"

interface="eu.weknowit.tutorial.osgi.IGreetingService" />
</beans>
```

pom.xml configuration

Some changes are needed in the configuration of the maven-bundle-plugin. We need to add `eu.weknowit.tutorial.client.osgi.listeners` to `Import-Package` and `Export-Package`:

```
<configuration>
 <instructions>
   <Bundle-SymbolicName>greeting-client</Bundle-SymbolicName>
     <Import-Package>
         eu.weknowit.tutorial.osgi,
         eu.weknowit.tutorial.client.osgi.listeners,
         org.apache.log4j,
         org.apache.servicemix.common,
         org.apache.servicemix.common.osgi,
         org.osgi.framework.*,
         org.springframework.osgi.config,
         org.springframework.osgi.context.event,
         org.springframework.osgi.service.exporter,
         org.springframework.osgi.service.exporter.support,
         org.springframework.osgi.service.importer,
         org.springframework.osgi.service.importer.support,
         org.springframework.beans.factory.config,
         org.springframework.context,
         org.springframework.util,
         org.springframework.core,
         org.springframework.aop,
         !*
     </Import-Package>
```

```
     <Private-
Package>eu.weknowit.tutorial.client.osgi.impl</Private-Package>
     <Export-
Package>eu.weknowit.tutorial.client.osgi,eu.weknowit.tutorial.clie
nt.osgi.listeners</Export-Package>
     <Spring-Context>*;create-asynchronously:=true;wait-for-
dependencies:=true;timeout:=120;publish-context:=true</Spring-
Context>
 </instructions>
</configuration>
```

Result

After you rebuild the client bundle and deploy it once again to the WKI System you should see following log on the console:

```
SAY: Welcome Traveller
```

Don't put such testing-listeners into production code !

This method should be used during development phase, but such listeners as the one we created here, which purpose is solely to give feedback during development, should not be a part of production services !

11.5.5. Spring, OSGi, beans and services

Figure 12 explains once again the interplay between the elements

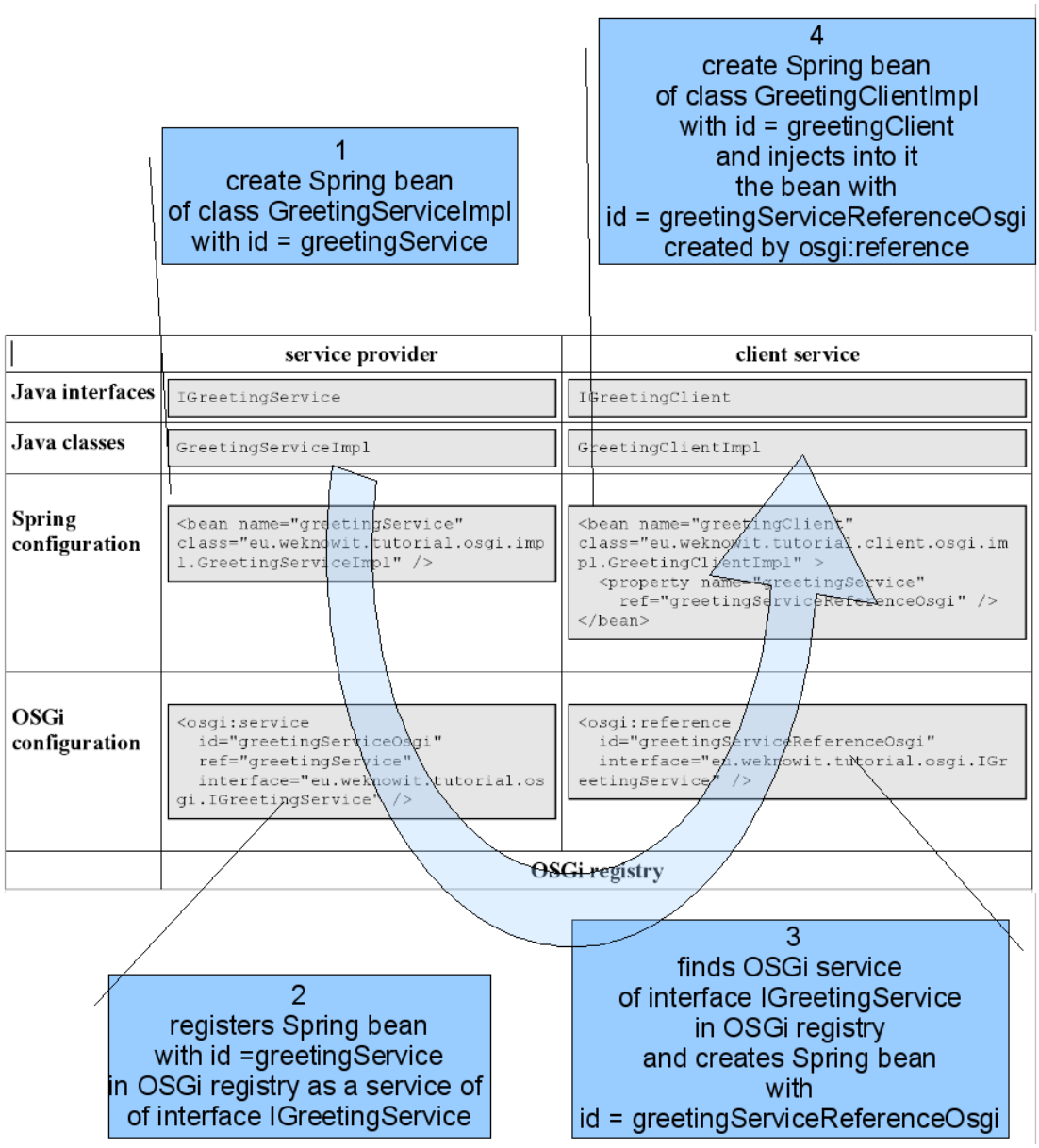

Figure 11 Spring, OSGi, beans and services

that we have created in this tutorial.

11.5.6. References

Further information may be found here:

- http://fusesource.com/docs/esb/4.0/getting_started/index.html
- http://www.osgi.org/ - OSGi site
- Some pointers to the he introductory material on IOC/Dependency Injection and Spring:

  - http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework
  - http://martinfowler.com/articles/injection.html
  - http://static.springframework.org/spring/docs/2.5.x/reference/introduction.html
- To learn more about Spring and Spring DM please refer to http://www.springsource.org, http://www.springsource.org/osgi
- To find out more about osgi declarations used in configuration files please refer to: http://static.springframework.org/osgi/docs/1.1.2/reference/html/service-registry.html.

11.6. How to deploy OSGi bundles to the WKI System

Date: 21.01.2009
Author(s): Łukasz Baran, Tomasz Kaczanowski
Organisation(s): SMIND
WP/Task/Subtask: WP 6/T6.2
Version: 0.1

History

Ver	Author	Description
0.1	Łukasz Baran, Tomasz Kaczanowski	First version

11.6.1. 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,
- The WKI System installed and running - see 'How to install the WKI System' guideline.

Outcome

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

11.6.2. 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 `osgi list` command from OSGi shell.

```
servicemix> osgi list
START LEVEL 100
   ID   State         Level  Name
[   0] [Active] [  0] System Bundle (1.2.1)
[   1] [Active] [ 10] geronimo-servlet_2.5_spec (1.1.2)
...
[ 129] [Active] [ 50] osgi-wki-greetingservice(1.0)
```

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:

```
servicemix> osgi help
```

11.6.3. 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

```
servicemix> log de
```

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/servicemix.log
```

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