Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

Specification of the "Early Integrated Infrastructure"

Deliverable number	*D6.5.1*
Dissemination level	*Public*
Delivery date	*28 February 2013*
Status	*Final*
Author(s)	*Ivan Martinez (AtoS), Miguel Angel Tinte (AtoS)*

This project is supported by the European Commission under the Information and Communication Technologies (ICT) Theme of the 7th Framework Programme for Research and Technological Development.

Executive summary

This document reports on the outcome of the T6.4 task related to the definition of the main integration aspects for the KHRESMOI project. Firstly, this concerns the definition of the integration approach or methodological guidelines to integrate the existing components in the project following the KHRESMOI Architecture defined in task 6.2. Secondly, it concerns the definition of the interfaces exposed to all components and the interaction between KHRESMOI components.

Based on the integration strategy defined and described at the beginning of the document a first iteration of the integration has been performed (where necessary simulating the expected behaviour of unavailable components) to check the viability of the defined integration.

This iteration included:

- *Identification* of the new functionalities that should be implemented as well as the ones that should be extended. To do that, we considered on the one hand the main functionalities derived from main business processes, and on the other hand we identified the relevant functionalities already available in the existing components.
- *Specification* of the existing services that should be adapted to the KHRESMOI system and for the new ones their interface and references have to be defined.
- *Realization* of the services in terms of development and deployment.

In our case, the interface and the reference declaration have been specified using the Service Component Architecture (SCA) standard. Essentially, SCA provides an XML mechanism to declare a service definition. The current document reports details of both the atomic and composed services using the SCA standard, and describes the mechanisms used to integrate different services, from different partners, into complex workflows.

Finally, we include a discussion of how the SCA services can be used with the cloud based prototype.

Table of Contents

1 List of abbreviations

SOA	Service Oriented Architecture
SCA	Service Component Architecture
SOMA	Service-Oriented Modelling Architecture
VM	Virtual Machine
REST	Representational State Transfer

Table 1: Abbreviations and acronyms

2 List of figures

3 List of tables

4 Introduction

4.1 Introductory Explanation of the Deliverable

System integration takes, as its starting point, the architecture design and components provided by the partners. The integration involves extensive testing of the overall system as well as of the integration points expected workflows. The document reports the work done in Task 6.4. Firstly, methodological guidelines have been defined to integrate the components of the project into the architecture defined in task T6.2 (D6.3.1, D6.3.2 and D6.3.3).

4.2 Purpose and Audience

4.2.1 Purpose

The purpose of this deliverable is to provide the strategy or methodological guidelines to integrate the existing project components into the KHRESMOI Architecture as defined in task 6.2. This concern the definition of the interfaces exposed to all components and the interaction between KHRESMOI components as well as the integration of the software architecture and the cloud infrastructure.

4.2.2 Audience

This deliverable is relevant to all technical work packages in KHRESMOI (WP1-WP9). The target audience includes: component providers, users, and any person inside or outside of the KHRESMOI project interested in learning about the internal operation of the KHRESMOI integrated platform. As such this deliverable presents the guidelines and technological details for the implementation of the KHRESMOI Integrated Prototype, the various components, the atomic and composites services defined using SCA and the integration of the KHRESMOI project infrastructure.

4.3 Structure of the Document

This deliverable is organized as follows: Section 5 describes the Integration approach proposed to define, implement and deploy the early integrated prototype. Section 6 lists the different atomic and composite services which are defined by means of SCA specification, providing support to the integration layer included in the KHRESMOI Architecture. Section 7 describes how SCA Composites have been integrated with the Early Cloud Infrastructure, and finally in Section 8, the conclusions of the deliverable are presented.

5 Integration Approach

As we reported in [1], a SOA approach for the KHRESMOI system integration has been adopted. To complement the SOA approach chosen, we provided a method (a) to identify the main services and (b) to integrate such services as a full system. SOA defines how to organise a complex system, which includes many different software components, in an abstract fashion and with a high level perspective and, as such, is ideal for bringing together the components developed by multiple project partners to form an integrated system.

Specifically, we have adopted the method proposed in [4], which is called Service-Oriented Modelling and Architecture (SOMA), due to the lack of guidelines to identify and specify the different services of the system in SOA. SOMA provides some methodological principles to organise SOA systems, proposing nine layers for the construction of a SOA based system, as shown in Figure 1.

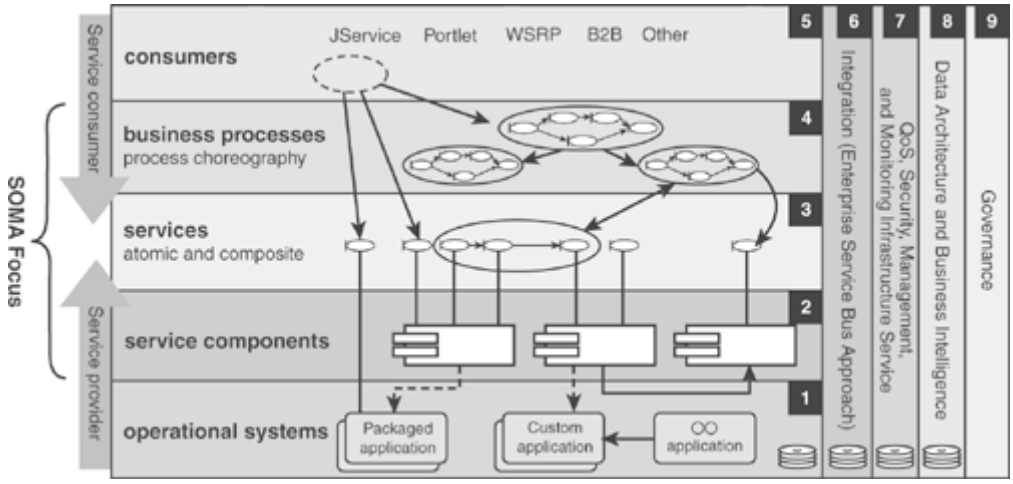

Figure 1. SOA reference architecture

- Layer 1: is the Operational System layer that contains all the different existing systems or software components that should be integrated in the new system.
- Layer 2: is the Enterprise Components layer, which is the intermediate layer, used by the Enterprise to deploy the existing systems previously described.
- Layer 3: is the Services layer that is composed of all the main services or atomic software components.
- Layer 4: is the Business Process Composition or Choreography layer. This layer corresponds to the service compositions that are required by the business processes.
- Layer 5: is the Presentation layer. This is the User Interface layer that will permit the end-users to realize the different business processes.
- Layer 6: is the Integration layer that provides a homogeneous framework to enable the different layer communications.

Finally, the three other layers are complementary layers that are important to take into account, but not as important as the previous ones. For example, the quality or the security will be assigned to a specific layer and should contain some services that will address it in the system.

All the different layers are used to design the KHRESMOI architecture. Of course, the specific requirements identified imply many adaptations of this very abstract structure.

The specification of the SOA systems depends on the user requirements (defined in [1], section 6.4.2) and the existing components (listed in [1], section 6.4.1). The idea is to find an intermediate solution to satisfy those two important constraints.

For this, the method SOMA proposes is to combine a top-down approach (focused on business processes) with a bottom-up approach (focused on the analysis of the existing components) and which consists of three steps:

- Identification of the services,
- Specification, and
- Realization.

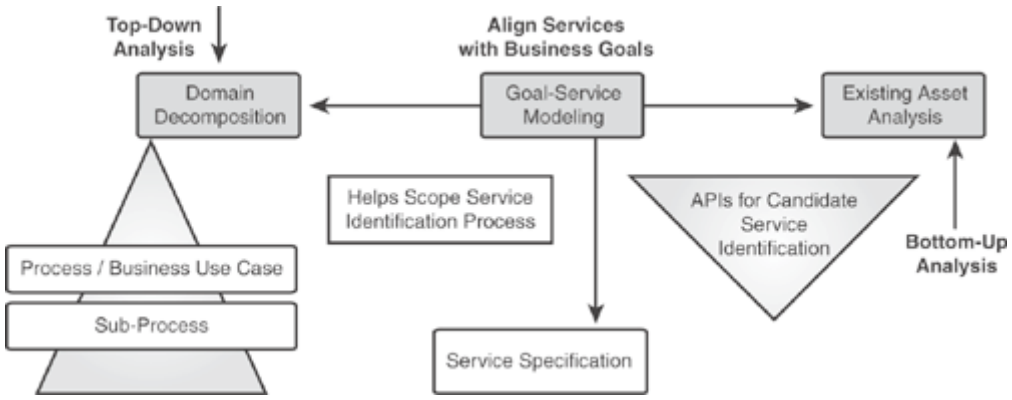

Figure 2. Process for the service specification

A. ***Identification***

On the one hand, the first approach should allow the identification of the main functionalities that are derived from the business processes. On the other hand, the second approach should identify the relevant functionalities already available in the existing components. Therefore, both the new functionalities to be implemented and the existing ones that should be extended can be identified.

As result of this step and in combination with the work done in WP3 related to UI functionality identification we now have taxonomy of functionalities grouped in 7 main blocks which are: Search, Hotlist Management, Collaboration, Working with Documents, Search History, User Interface and Account Management and finally, User Profiles. Figure 2 shows the UI functionalities identified for the KHRESMOI System.

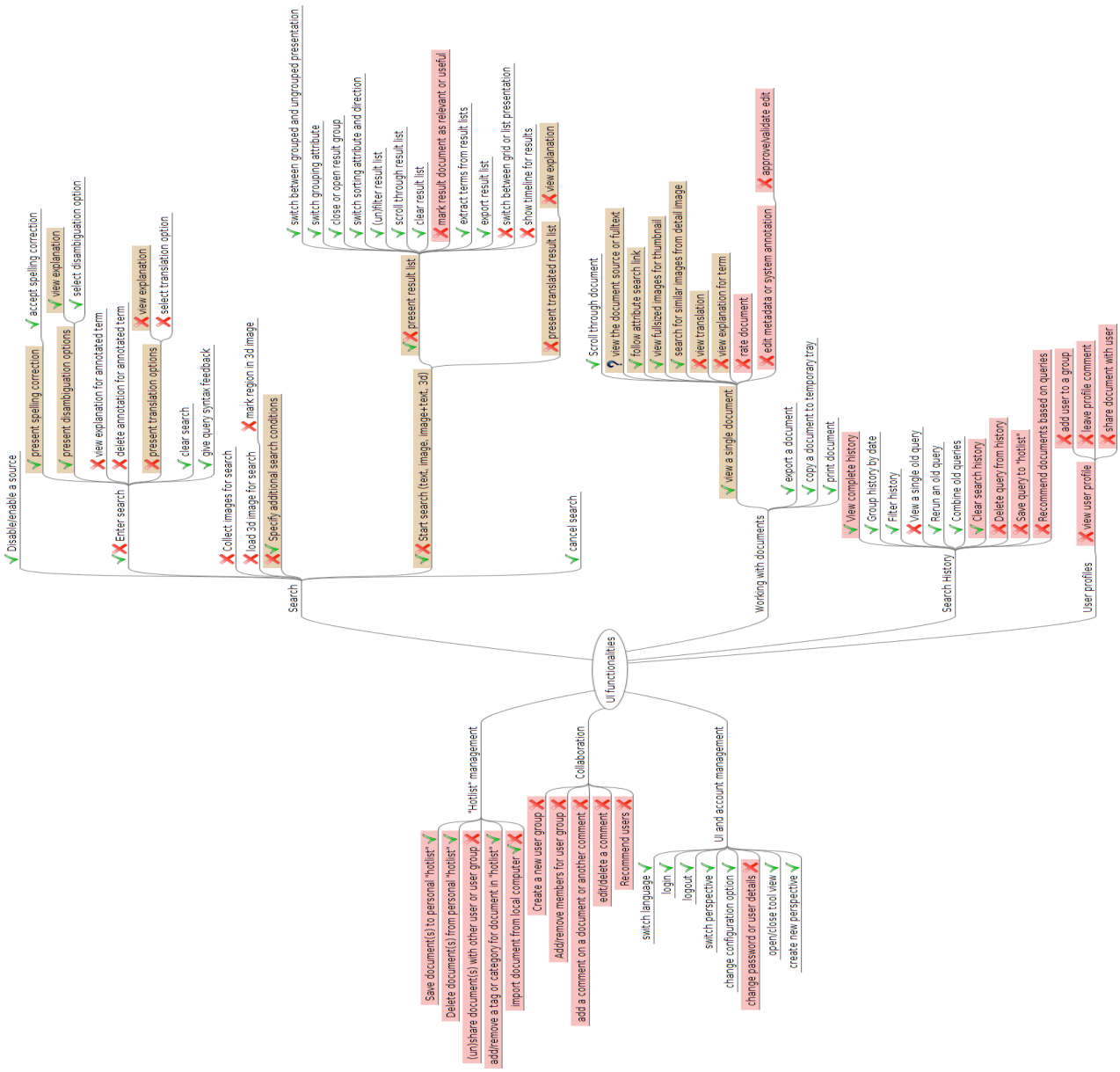

Figure 3. KHRESMOI UI functionalities.

B. ***Specification***

When all the services are identified, those that already exist but should be adapted to the KHRESMOI system, and the new ones that should be implemented, their interface and references should be specified.

In Table 2, all the components provided by the partners are presented. For each one, the relevant functionalities for KHRESMOI are listed. The functionalities presented in this table could be deployed directly in the KHRESMOI system, but for the majority of them, they have been extended in the corresponding WP to fit with the user requirements.

For each component, all the information has been collected (installation, implementation language, interface, references, etc.).

Partner	Tool	Functionalities
USFD	MIMIR, QMS	- Mimir – index annotations and text - Mimir – query indices - Query Mapping
ONTO	Owlim (exposed as Sesame repository)	- Persist RDF data - Evaluate SPARQL query - Disambiguation (autocomplete)
HES-SO	ParaDISE	- Image database indexation - Query by example image retrieval - Support user feedback for query refinement
UDE	ezDL Daffodil	- Docking, tool-based UI framework for search with perspectives: o meta search over several sources with result integration, sorting, filtering, term extraction and different methods for query specification o viewing, collect, organize and export results, terms and past queries - Agent-based backend for ezDL clients: o wrapper and query framework (includes transformation of query syntax), integration of results from different sources, caching of result metadata o logging of all user and system actions o user administration for authentication and personalization - Strategic support using CBR
CUNI	MOSES: Czsem Mining Suite	- TectoMT Analysis for GATE: provides TectoMT linguistic analysis of text for GATE documents. - Mímir Index Feeder: Custom component used to fill Mímir index with analysed Gate documents, under development. - TectoMT (http://ufal.mff.cuni.cz/tectomt/): Set of NLP tools used for linguistic analysis of text.
MUW	MUW-3D: Image Learning and Retrieval	- Derive Internal Models given a set of data - Retrieval System Anatomy - Retrieval System Pathology - Semi-supervised feedback

		- Batch system:
		  o Detect Anatomic Regions - Learning
		  o Abnormality Similarity - Learning
		  o Knowledge Persistence Helper (batch)
		- Search system:
		  o Detect Anatomic Regions - Retrieval
		  o Abnormality Similarity – Retrieval

HON	CheckSpeller	- Spelling correction - MeSH Terms Extractor

Table 2. List of the existing components provided by the consortium

Whatever the implementation language used by the service, it should declare the standard interface such as SOAP, REST, etc. This interface will enable the interoperability with the other services that are deployed in the system.

Furthermore, to work properly, the service can require data from other components. This dependency is called *reference* and should be also defined. This part contains the protocol and the bindings to use the required service.

In our case, the interface and the reference declaration have been specified with the SCA formalism. Essentially, SCA provides an XML mechanism to declare the service definition. Section 6 provides all details about the atomic and composed services specification using SCA formalism

C. ***Realization***

Based on the service specification, the realisation of the services consists of development and deployment. In fact, thanks to the service specification, the service is implemented for deployment through the SCA runtime allowing the composition of services to be tested within a homogeneous framework.

Finally, Figure 4 shows the KHRESMOI SOA system with the different services defined in each layer, and their respective interactions.

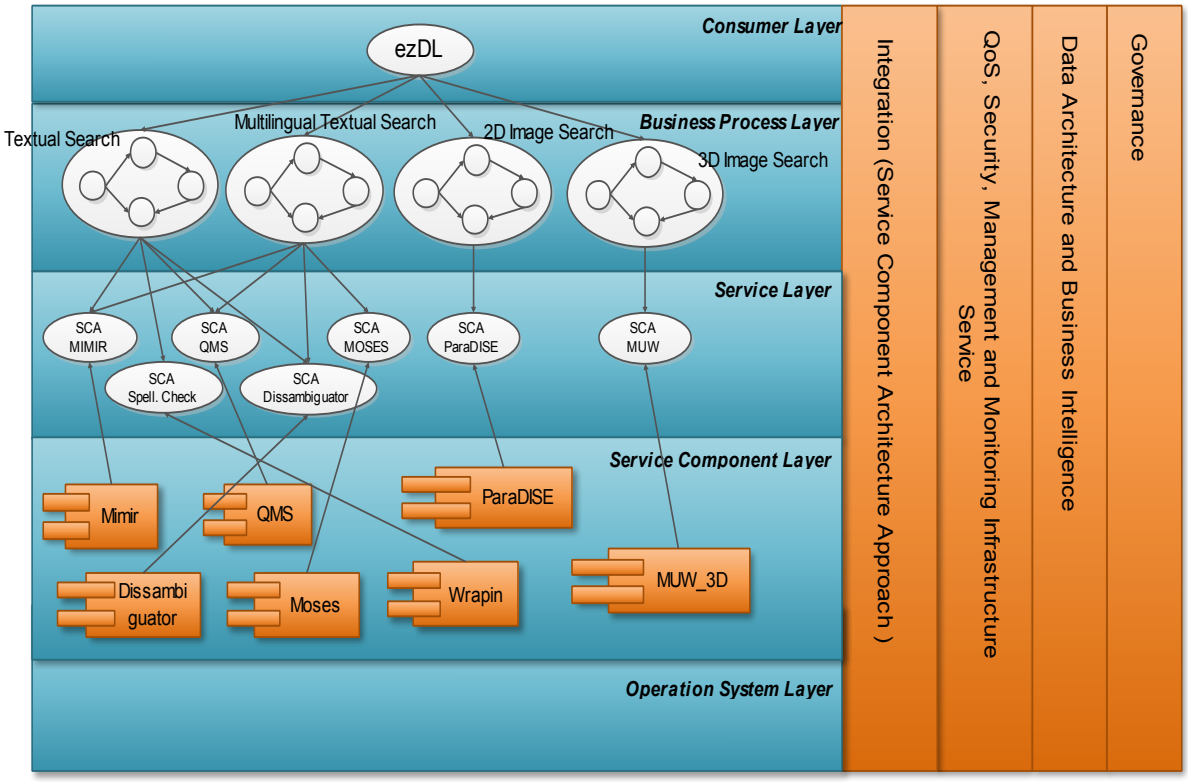

Figure 4. KHRESMOI SOA System

As the user requirements activities are still open and the evaluation of the early prototype is taking place, some design iterations are missing which prevents us from presenting the full KHRESMOI architecture.

6 Atomic and Composite Services Definition using SCA

Service Component Architecture (SCA) is a model that permits for the assembly of service components into business solutions. The SCA specification simplifies the component programming model for implementing easily services. Moreover, SCA provides a number of benefits that help implement SOA requirements:

- Component integration does not rely on knowing how each component is implemented. This is very important when aiming for a loosely coupled system.
- The components can easily be replaced by other components, which allows for a highly flexible system.
- The services can be easily invoked either synchronously or asynchronously.
- The composition mechanism is very clearly described.

This kind of model is very useful when prototyping a complex system which brings together many different components/technologies.

Based on the architecture definition, the initial process for describing the KHRESMOI system is to define the individual components (both atomic and composed).

Before defining specific components we first briefly outline how an SCA atomic component is defined. Each component has a number of properties which allow different components to work together. Figure 5 shows how an SCA Component is described by means of the following elements:

- The services: a service is the declaration of a specific functionality (or method) provided by the component.
- The references: a reference is the definition of the components required to work properly.
- The binding: the binding is the connection created between components in order to make them work together.
- And eventually some properties, to express specific data that are used to configure a specific context to use the component, i.e. user connection credentials, etc.

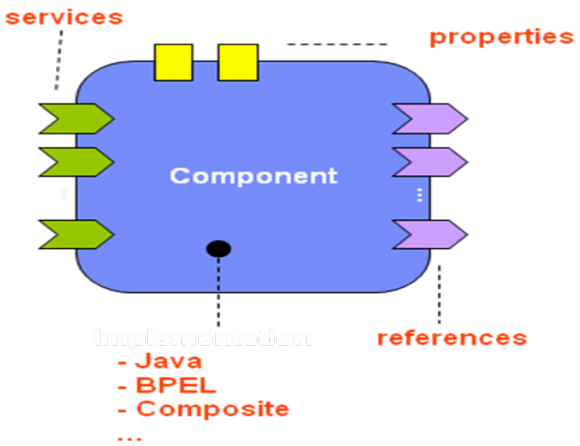

Figure 5. Graphic representation of the SCA component

The remainder of this chapter aims to describe the currently available KHRESMOI SCA Components that have been developed in order to integrate different partner's services into complex workflows. The implementation is written in Java and utilises the Apache Tuscany SCA framework technologies.

The code which implements the components described within the rest of this section can be found in Appendix A: Java code implementation for SCA Components.

6.1.1 SCA MIMIR (USFD)

The SCA Mimir Component provides a common interface to the Mimir Service provided by The University of Sheffield. This component is used within KHRESMOI to provide semantic text search. The main elements of this SCA implementation are:

- Services: two main services are defined; mimirHitService and mimirDocService
- References: the references to the services used are defined in the web.composite file:

```
<reference name="mimirHitService">…</reference>
<reference name="mimirDocService">…</reference>
```

- Binding: SCA Mimir has been bound or linked with USFD through a REST invocation defined in web.composite file:

```
<tuscany:binding.rest uri="http://services.gate.ac.uk/"/>
<tuscany:binding.rest uri="http://demos.gate.ac.uk/"/>
```

Figure 6 below shows the different elements described:

Figure 6. Mimir SCA Implementation

6.1.2 SCA QMS (USFD)

The SCA QMS Components provides an interface to the Query Mapping Service from The University of Sheffield. The QMS is responsible for converting text queries into Mimir queries. The main elements of this SCA Implementation are:

- Services: QMS service has defined qmsService that implements the functionalities provided.
- References: the references to the services used are defined in the web.composite file:

```
<reference name="qmsService">
```

- Binding: SCA QMS has been bound or linked with USFD through a REST invocation defined in web.composite file:

```
<tuscany:binding.rest
uri="http://services.gate.ac.uk/khresmoi/rest/service">
```

Figure 7 below shows the different elements described:

Figure 7. QMS SCA Implementation

6.1.3 SCA Check Speller (HON)

The SCA Speller implements HON Speller REST Service API in order to integrate its functionality into the Textual Search workflow. The typical use of this application is spelling check of the query that the user has typed in the UI.

The main elements of this SCA Implementation are:

- Services: Speller service has defined wrapinService that implements the functionalities provided.
- References: the references to the services used are defined in the web.composite file:

  ```
  <reference name="wrapinService">
  ```

- Binding: SCA Speller has been bound or linked with HON service through a REST invocation defined in web.composite file:

  ```
  <tuscany:binding.rest uri="http://khresmoi.honservices.org"/>
  ```

Figure 8 below shows the different elements described:

Figure 8. SCA Speller Implementation

6.1.4 SCA Dissambiguator (ONTO)

The SCA Dissambiguator component implements Ontotext Dissambiguator REST Service as a query terms dissambiguator in order to integrate its functionality into the Textual Search workflow. The supported languages are: English, German (de), Czech (cz), Spanish (sp), French (fr).

The main elements of this SCA Implementation are:

- Services: Dissambiguator service has defined dissambiguatorService that implements the functionalities provided.
- References: the references to the services used are defined in the web.composite file:

  ```
  <reference name="dissambiguationService">
  ```

- Binding: SCA Dissambiguator has been bound or linked with Ontotext service through a REST invocation defined in web.composite file:

  ```
  <tuscany:binding.rest uri="http://khresmoi.ontotext.com/autocomplete.json">
  ```

Figure 9 below shows the different elements described:

Figure 9. SCA Dissambiguator Implementation

6.1.5 SCA MT (CUNI)

The SCA Multilingual Components implements MOSES Service from CUNI and provides complete Machine Translation for KHRESMOI. The translation is supported for several languages as English, Czech, French and German. Also it works with both document summaries and full documents.

The main elements of this SCA Implementation are:

- Services: MOSES service has defined multilingualService that implements the functionalities provided.
- References: the references to the services used are defined in the web.composite file:

  ```
  <reference name="multilingualService">
  ```

- Binding: SCA MT has been bound or linked with CUNI Moses service through a REST invocation defined in web.composite file:

  ```
  <tuscany:binding.rest uri="http://quest.ms.mff.cuni.cz:8888/">
  ```

Figure 10 below shows the different elements described:

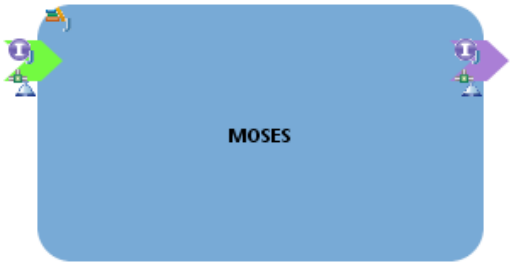

Figure 10. SCA MT Implementation

6.1.6 SCA ParaDISE (HES)

The ParaDISE service is an image retrieval component provided by HES to manage 2D Image search workflow within KHRESMOI project. The implementation of this SCA Component requires two different references: one related to content-based queries and another one related to retrieve images.

The SCA ParaDISE component is defined in the code implementation:

```
<component name="ParaDISEWebComponent">
```

The main elements of this SCA Implementation are:

- Services: ParaDISE service has defined paradiseService that implements the functionalities provided and imageRepositoryService that allows access to images server.
- References: the references to the services used are defined in the web.composite file:

  ```
  <reference name="paradiseService">
  <reference name="imageRepositoryService">
  ```

- Binding: SCA ParaDISE has been bound or linked with HES ParaDISE service through a REST invocation defined in web.composite file:

  ```
  <tuscany:binding.rest uri="http://fast.hevs.ch"/>
  <tuscany:binding.rest uri="http://fast.hevs.ch:2080"/>
  ```

Figure 11 below shows the different elements described:

Figure 11. SCA ParaDISE Implementation

6.1.7 SCA MUW 3D (MUW)

MUW is a 3D image search engine that allows image searching through different queries. SCA MUW implements the MUW service REST API in order to integrated 3D Image search into KHRESMOI workflows.

The main elements of this SCA Implementation are:

- Services: MUW service has defined paradiseService that implements the functionalities provided and imageRepositoryService that allows access to images server.
- References: the references to the services used are defined in the web.composite file:
  ```
  <reference name="muwService">
  ```
- Binding: SCA ParaDISE has been bound or linked with MUW service through a REST invocation defined in web.composite file. This binding needs an http-header for Authorization credentials in order to log into the service:
  ```
  <tuscany:binding.rest uri="http://api.cir.meduniwien.ac.at/khresmoi/rest/">
     <tuscany:http-headers>
        <tuscany:header name="Authorization" value="******"/>
     </tuscany:http-headers>
  </tuscany:binding.rest>
  ```

Figure 12 below shows different main elements described:

Figure 12. SCA MUW Implementation

6.2 Composite Services description with SCA

The composite is a basic unit of composition that allows atomic components to be put together to provide higher level services

After the services deployment, they are considered as public and can be consumed by other components. To define the specific connection between two components inside the composites, a wire is declared. A wire permits to link the service (functionality to be consumed) and the reference of the component (functionality required to achieve the process).

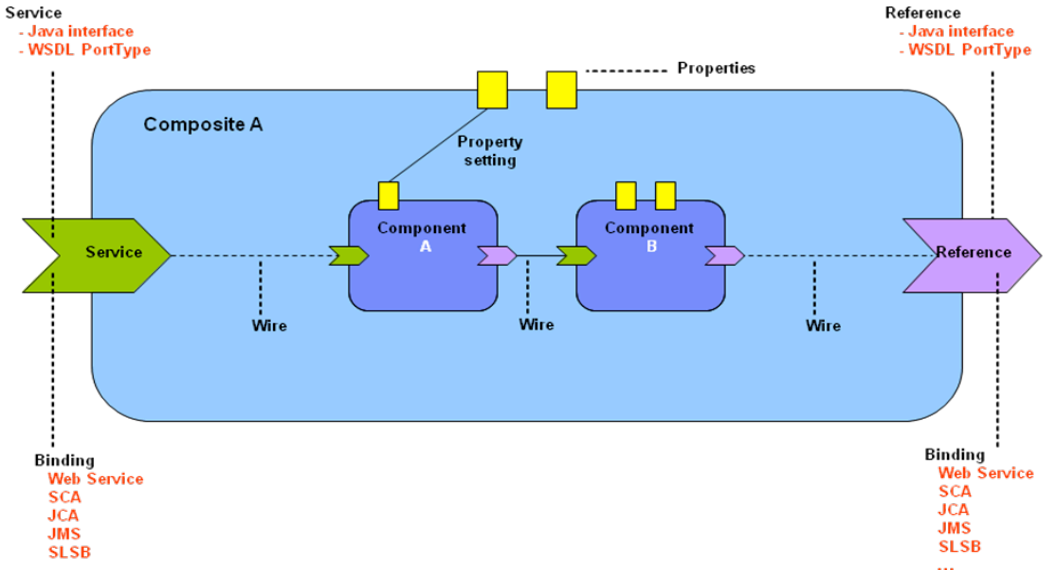

Figure 13. Graphic representation of the SCA composite

The composite represents the composition of different components, but it can also represent the composition of different composites. This composition mechanism may be used as an implementation of components at next higher layer.

6.2.1 SCA Textual Search Composite

SCA Textual Search Composite represents the KHRESMOI Textual Search workflow defined for the project and integrates the required SCA Components already created for the proper working. Within this composite we have the next components and services:

- *eZDL Query Interaction* Component: this component represents the User Interface, that is eZDL in our case. This component is integrated in the workflow because query typed by the end user starting the textual search launch several calls to rest of components in order to fulfil the search performed.
- *CheckSpeller*: this component can be represented as SCA Speller and is in charge of correcting any possible mistake from the grammar point of view when query terms are introduced.
- *Dissambiguator*: this service is the SCA Dissambiguator component that processes query terms to accurate scope of the query.
- *Textual Search Manager*: this new component is in charge of managing textual search queries and searches over QMS and Mimir. Here is where are taken as input a set of query terms after spelling and disambiguate them creating an expanded query through SCA QMS Component. Once this query is created, it is launched over Mimir index in order to return either hits or documents
- *QMS, Mimir*: SCA QMS and SCA Mimir are directly invoked through Textual Search manager and return query results as the composite output.

The main elements of this SCA Composite that allow to different components and services work together are:

- The SCA Textual Search component is defined in the code implementation:
  `<component name="TextualSearchWebComponent">`
- Services: the Textual Search composite service has defined four main services: wrapinService, owlimService, mimirService and qmsService that implement the main components described above.
- References: the references to the services used are defined in the web.composite file:

```
<reference name="wrapinService"></reference>
<reference name="owlimService">…</reference>
<reference name="mimirService">…</reference>
<reference name="qmsService">…</reference>
```

- Binding: SCA Textual Search has been bound or linked with SCA Components through different REST invocation defined in web.composite file:

```
<tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-wrapin"/>
<tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-owlim”/>
<tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-QMS"/>
<tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-mimir”/>
```

  These bindings refer to components already deployed in several architecture nodes, so therefore workflow is centralized into the KHRESMOI platform.

The figure 14 below shows the different elements described above:

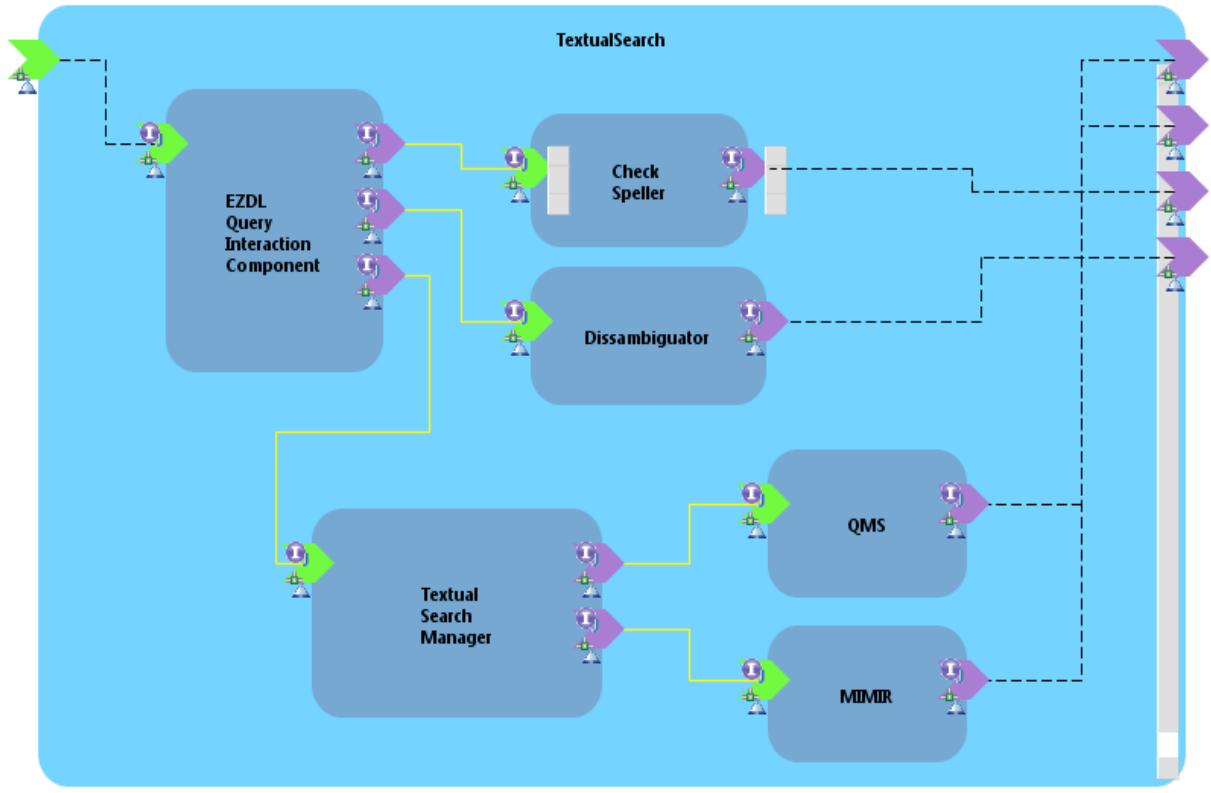

Figure 14. SCA Textual Search Implementation

Figure 14 shows how the SCA Textual Search Composite encompasses different SCA Components involved in the Textual Search workflow through different services and references previously described.

6.2.2 SCA 2D Image Search Composite

This composite aims to encompass different SCA Components and services in order to fulfil with 2D Image Search requirements. To do so, another SCA Component has been created, Image Search Manager to deal with both 2D and 3D Image search workflows. Therefore, this new component connects directly to SCA ParaDISE

In order to communicate Image Search from UI with component in charge of returning results for it.

The main components and services used here are:

- eZDL: this component is UI where the end-users are able to search for an image ID to retrieve it as an image or to search relevant Images related to an ID as a JSON format.
- Image Search Manager: this SCA Component is the one to deal with Image search workflows. For 2D Image Search are exposed several methods as REST under "/2D/…" path implementing the functionalities offered by ParaDISE. His goal is also to manage properly image search queries launched by the users.
- SCA ParaDISE: this is the SCA Component Implementation of ParaDISE service.

The main elements of this SCA Implementation are:

- The SCA Image Search component is defined in the code implementation:
  ```
  <component name="ImageSearchWebComponent">
  ```
- Services: the Image Search service has defined three main services: giftService, paradiseService and muwService that implement functionalities required for 2D and 3D Image Search. Both giftService and paradiseService are used for 2D Image Search although currently paradiseService is the last version it is being used.
- References: the references to the services used are defined in the web.composite file:
  ```
  <reference name="giftService">…</reference>
  <reference name="paradiseService">…</reference>
  <reference name="muwService">…</reference>
  ```
- Binding: SCA Image Search has been bound or linked with SCA Components through a REST invocation defined in web.composite file:
  ```
  <tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-GIFT/"/>
  <tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-ParaDISE/"/>
  <tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-MUW/"/>
  ```

  These bindings refer to components already deployed in several architecture nodes, so therefore workflow is centralized into the KHRESMOI platform.

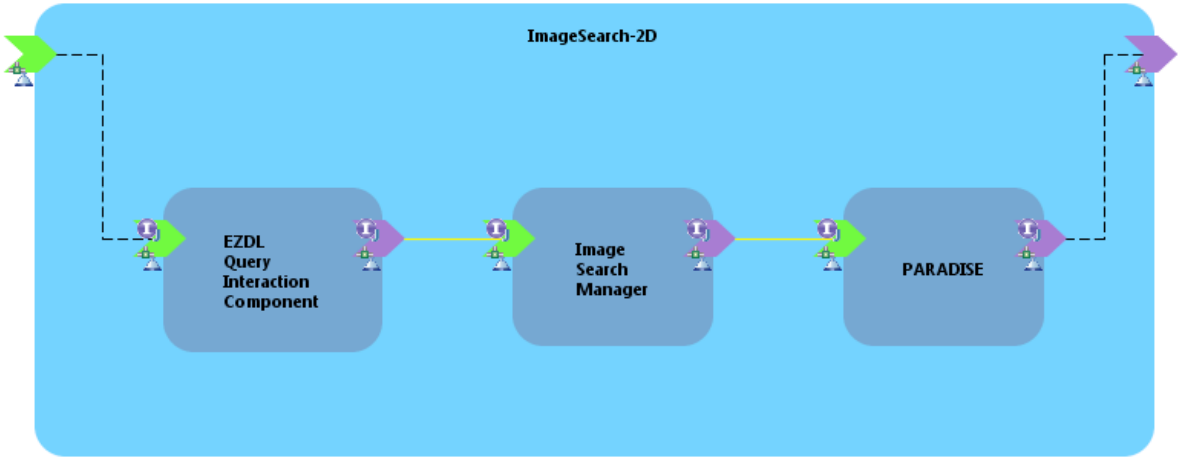

Figure 15. SCA 2D Image Search Implementation

Figure 15 shows the composite that implements the 2D Image Search workflow and all the components involved in that. The Image Search Manager is presented as the key element to deal with the search and to communicate UI with components.

6.2.3 SCA 3D Image Search Composite

Similarly to the previous composite, this SCA Composite focus on describing components involved in 3D Image Search workflow.

The main components and services used here are:

- eZDL: this component is UI where the end-users are able to search for an image ID to retrieve images, series or report related to an ID as a XML format.
- Image Search Manager: this SCA Component is the one to deal with Image search workflows. For 3D Image Search are exposed several methods as REST under "/3D/…" path implementing the methods offering by MUW.
- SCA MUW: this is the SCA Component Implementation of MUW service.

The main elements of this SCA Implementation are:

- The SCA Image Search component is defined in the code implementation:
  `<component name="ImageSearchWebComponent">`
- Services: the Image Search service has defined three main services: giftService, paradiseService and muwService that implement functionalities required for 2D and 3D Image Search. Last one, muwService is the one that is being used for 3D Image Search.
- References: the references to the services used are defined in the web.composite file:
  `<reference name="muwService">…</reference>`
- Binding: SCA Image Search has been bound or linked with SCA Components through a REST invocation defined in web.composite file:
  `<tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-MUW/"/>`

  These bindings refer to components already deployed in several architecture nodes, so therefore workflow is centralized into the KHRESMOI platform.

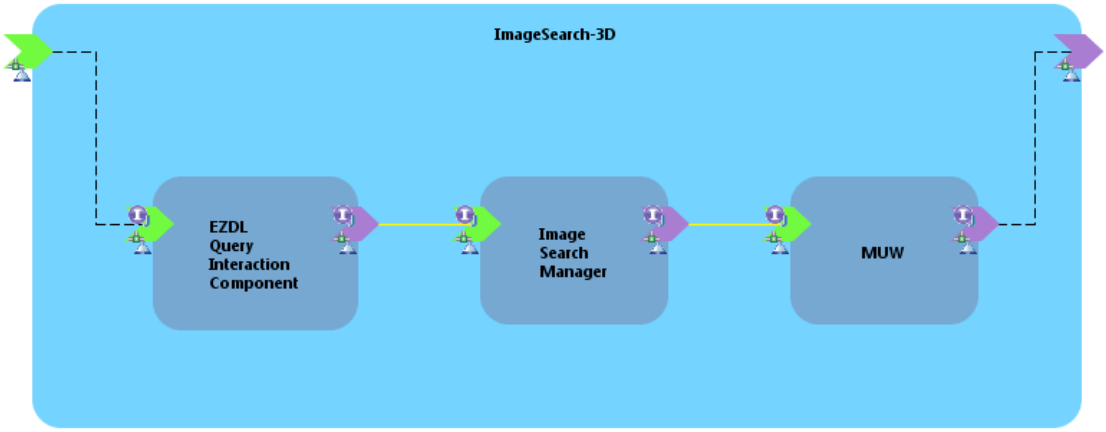

Figure 16. SCA 3D Image Search Implementation

In Figure 16, the final outcome of this integration is exposing 3D Image Search workflow within SCA Composites and Components integration.

6.2.4 SCA Multilingual Textual Search Composite

SCA Multilingual Composite is a special case of Textual Search workflow where query is translated during execution. In this case, many of the components and services remain the same for multilingual translator based on MOSES system service that is added to this composite:

- MOSES: this service is implemented through SCA MT Component to provide dynamic multilingual translation during workflow.

The main elements of this SCA Composite that allow to different components and services work together are:

- The SCA MT component is defined in the code implementation:

```
<component name="MTWebComponent">
```

- Services: the Multilingual Search composite service has defined another service besides the four already defined for Textual Search: multilingualService.
- References: the references to the services used are the same that for Textual Search but including multilingual service defined in the web.composite file:

```
<reference name="multilingualService"></reference>
```

- Binding: SCA Multilingual Search has been bound or linked with SCA Components described before and also to MT service through different REST invocation defined in web.composite file:

```
<tuscany:binding.rest uri="http://46.165.196.42:8080/khresmoi-MT"/>
```

  These bindings refer to components already deployed in several architecture nodes, so therefore workflow is centralized into the KHRESMOI platform.

Figure 17 below shows the different components working together in this workflow:

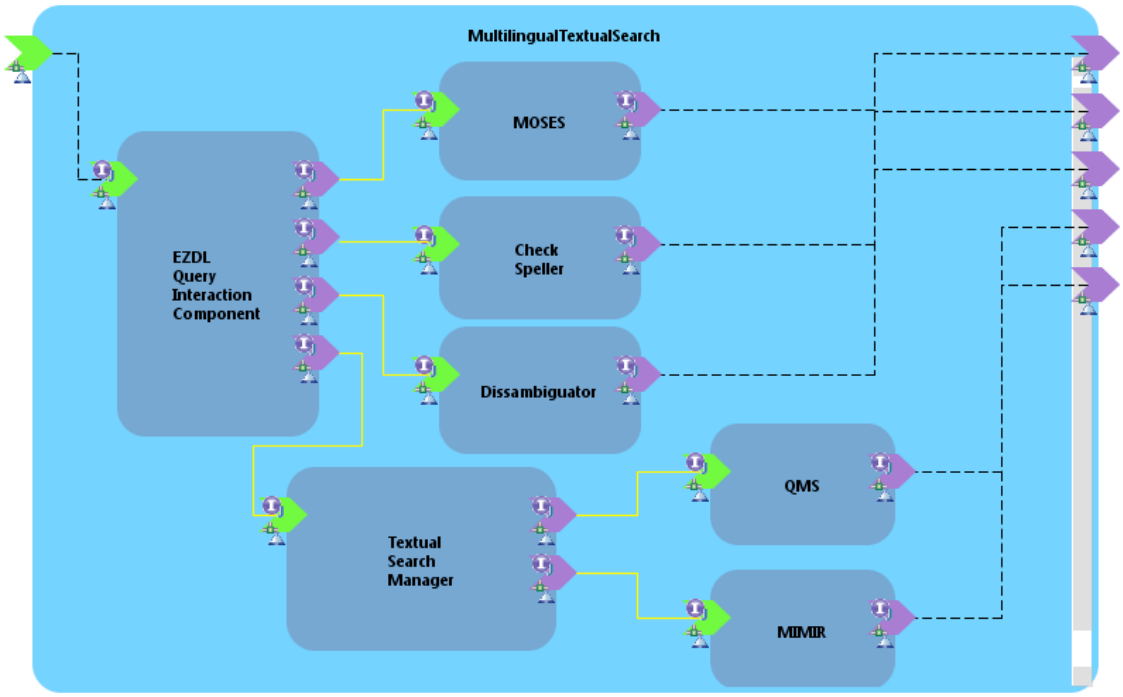

Figure 17. SCA Multilingual Search Implementation

7 Integrating SCA Composites on the Early Cloud

The SCA approach allows the addition of the Service description around the existing components without imposing a component re-factoring. The benefits of this approach are (a) an easy deployment of services and (b) a testing of the possible composition between services. For example, the four integrated prototypes presented in [1] have been deployed and tested with this approach. In addition, the new functionalities which appeared during the first 27 months of the project have been quickly specified to be in concordance with the KHRESMOI architecture. Finally, the result of the combination of the different integrated prototypes will be the full integrated prototype.

As was presented in deliverable D6.3.1 [1], Apache Tuscany is the most relevant SCA runtime that permits us to easily deploy the SCA implementation. In particular, Apache Tuscany supplies the Tuscany Domain manager that allows the configuration of server nodes where several composites are deployed.

Different configurations of composites can be deployed and tested to evaluate their efficacy. The system can be (a) locally deployed, that is, all composites are deployed on one local server, as well as (b) deployed in a distributable fashion, which means that several composites are deployed on several servers. The second option has been the one selected to deploy the KHRESMOI Integrated Prototype.

SCA simplifies the composite deployment and is completely compliant with the requirements of any cloud infrastructure.

The technologies and tools we use include:

- Apache Tuscany provides a platform for SOA Development and Management based on the guidelines specified by Open Service Component Architecture (OASIS OpenSCA). With SCA standard definition, Tuscany offers integration among applications developed in different languages and placed over different environments that can be integrated using this platform. In this document, it is described Apache Tuscany SCA as the infrastructure solution for creating and hosting the domains, components etc. that could be deployed across the cloud and partners infrastructure, thus creating a composite application.
- OpenNebula Cloud is an open source cloud infrastructure solution. It is currently API compatible with Amazon EC2, S3 and EBS. We use the OpenNebula Cloud as the cloud platform on which a part of the composite application resides.

Next, we describe in detail the approach followed to build and integrate composite applications (SCA Composites listed in Section 5) using Apache Tuscany and OpenNebula to create a hybrid composite application. To show that distributed applications comprising of composite modules (distributed across the cloud and different partners local infrastructures) can be integrated and function as a single unit using SCA without compromising on security, we create a composite application that components spread over different domains distributed across the cloud and the enterprise infrastructure. We then use SCA to host and integrate this composite application so that it fulfils the necessary functional requirements. To ensure information and data security, we set up a virtual private network (VPN) between the different domains, creating a point-to-point encrypted network which provides secure information exchange between the two environments.

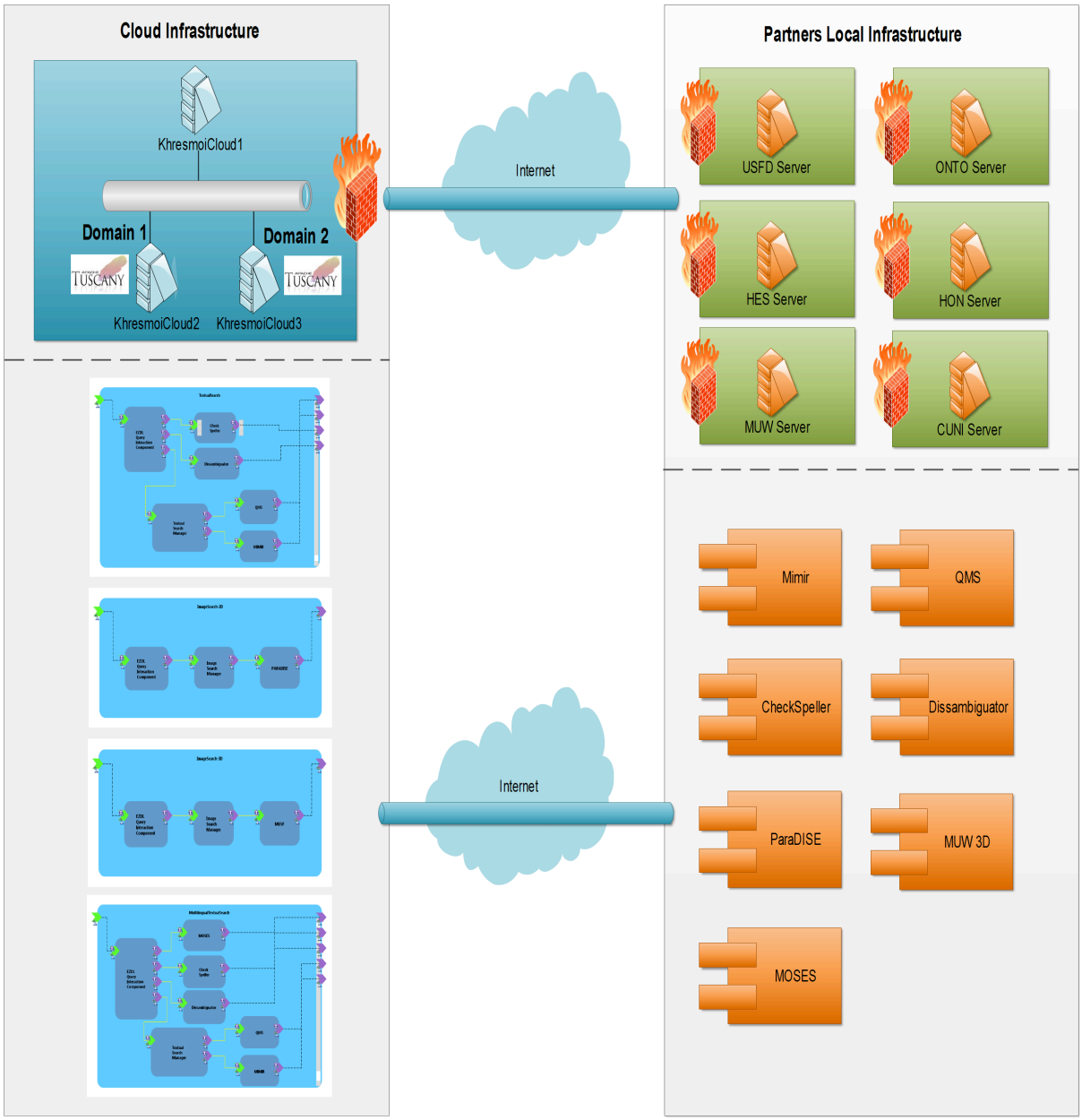

Figure 18. Integrated Prototype Deployment Architecture

As Figure 18 illustrates, we developed a KHRESMOI prototype composed of four complex distributed composites which uses the web service binding provided by Apache Tuscany SCA to communicate between the client (Domains 1 and 2, hosted on ATOS private cloud) and the partners servers (hosted on partners local infrastructure). The part of the application which is hosted on the partners local infrastructure is referenced by the different components exposed as a service on the cloud; resulting in the creation of a seamlessly integrated composite application spread over the cloud and partners local infrastructure.

7.1.1 The SCA domain and Tuscany nodes

An SCA domain manages a set of composite applications that can be connected using SCA wires and constrained by the same policy definitions. An SCA domain consists of the following:

- A virtual domain-level composite (that is, a domain composite) whose components are deployed and running
- A set of installed contributions that contain composites, implementations, interfaces, and other artifacts

An SCA domain represents a complete configuration of the SCA assembly. The components within an SCA domain can be hosted by one or more runtime environments that provide the necessary technology infrastructure. Tuscany introduces the node concept to represent a group of components (typically declared in the same composite) that can be started, executed, and stopped together on a Tuscany runtime. Figure 19 illustrates a KHRESMOI definition of such mapping. The composites are part of the KHRESMOI scenarios based on the use cases defined in WP8 and WP9.

In this cloud infrastructure proposed for the KHRESMOI system, four Tuscany nodes are used to run the applications or SCA Composites. Tuscany is quite flexible in how it maps the domain composite to different node topologies. We are running each composite on a different node.

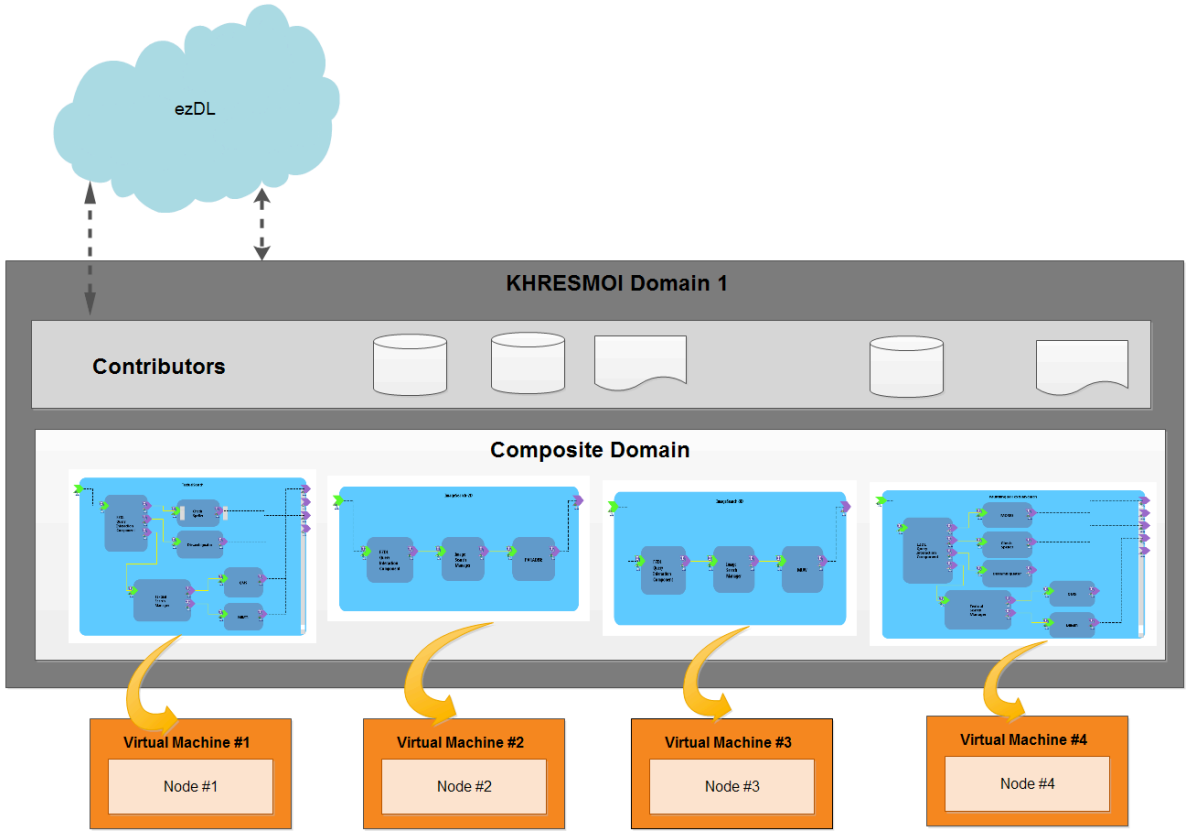

Figure 19. Tuscany Nodes Domain 1

Continuing with the cloud infrastructure proposed for the KHRESMOI system, seven Tuscany nodes are used to run the SCA components. Following the same approach that in Domain 1, each of the SCA Components is running on a different node. Figure 20 shows this mapping between Component Domain to node topology.

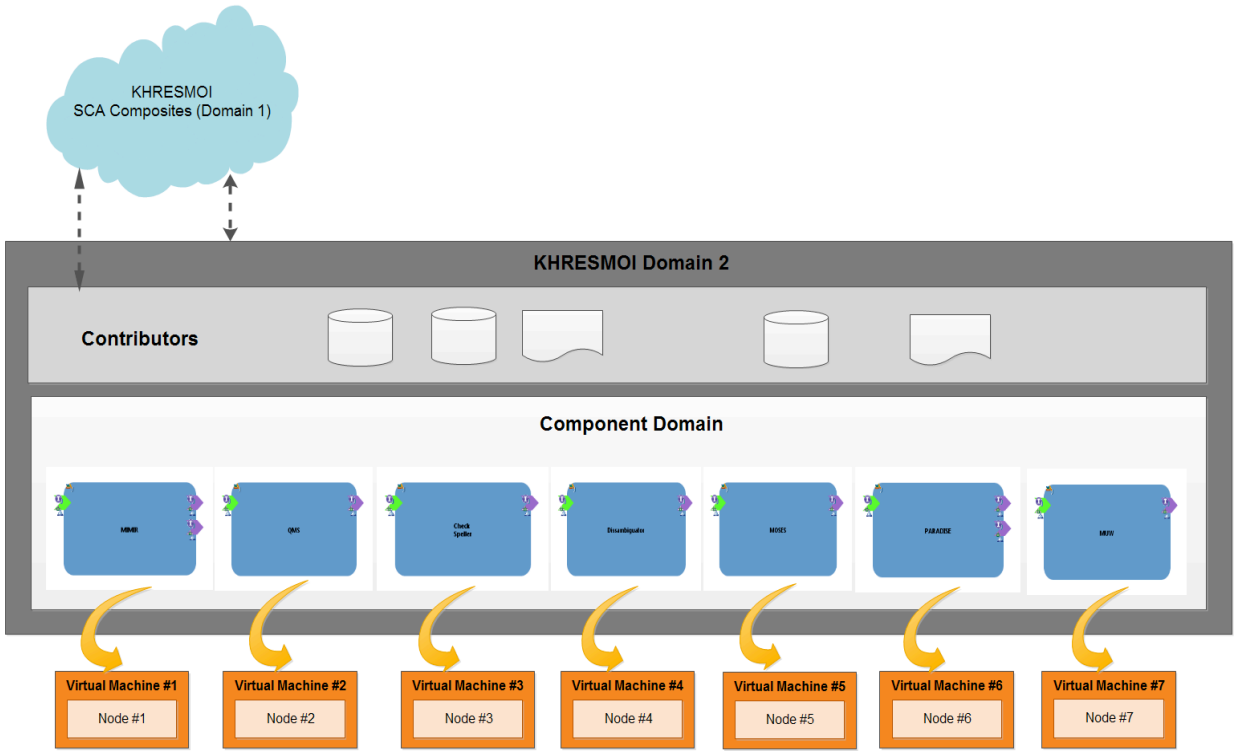

Figure 20. Tuscany Nodes Domain 2

8 Conclusions

In this deliverable, we present a summary of the work carried out in relation to T6.4 “System Integration”. The initial work involved a state of the art study into possible integration methodologies. This review led to SOMA being adopted as described in this deliverable.

The SOMA methodology provides some guidelines and principles to organise SOA system. The specification of the KHRESMOI SOA system depends on the user requirements and the existing components. SOMA method proposes to combine a top-down approach (focused on business processes) and a bottom-up approach (focused on the analysis of the existing components) by means of three different steps which are: Identification of the services, Specification and finally, Realization.

Based on this approach a first iteration of the integration has been performed. As result of this iteration all the services were identified and their interfaces and references were specified. As described in previous deliverables related to Architecture definition [1] we chose SCA formalism for the formal specification of services. SCA provides the most generic specification of software components and the mechanism to compose services. Thus, SCA provides a useful software infrastructure to integrate and deploy various software components as loosely coupled services. This deliverable includes descriptions of the main SCA component as well as KHRESMOI workflows created, which provides a general view of the integrated infrastructure proposed.

We have also shown how the integrated system can be deployed to the cloud in order to scale on demand.

During the coming months further iterations will be carried out in order to provide the specification and deployment of the Full Integrated Infrastructure for KHRESMOI System.

9 References

[1] Emmanuel Jamin, Vassil Montchev, Konstantin Pentchev, Deliverable D6.3.1 "State of the art, concepts and specification for the "Early Prototype" in KHRESMOI Project, Seventh Framework Programme.

[2] Ivan Martinez, Miguel Angel Tinte, Deliverable D6.3.2 "Evaluation of the 'Early software architecture' and further specification" in KHRESMOI Project, Seventh Framework Programme.

[3] Ivan Martinez, Miguel Angel Tinte, Deliverable D6.3.3 "Prototype and evaluation of the Full software architecture" in KHRESMOI Project, Seventh Framework Programme.

[4] Ali Arsanjani, Francisco Curbera, Nirmal Mukhi: Manners Externalize Semantics for On-demand Composition of Context-aware Services. ICWS 2004: 583-590.

Appendix A: Java code implementation for SCA Components

A.1 SCA Mimir

```
/**
 * MIMIR is a new semantic search engine that combines a fast full-text index and
 * a high-capacity semantic repository, allowing Boolean, SPARQL and annotation
 * pattern searching in a single query.
 */
@Path("mimir")

public class MimirApplication {

    @Reference
    protected JaxrsMimirService mimirHitService;

    @Reference
    protected JaxrsMimirDocService mimirDocService;

    /**
     * @param index_ID index ID
     * @param action Mimir actions available: postQuery, documentsCount, documentId,
       documentScore, documentHits, documentMetadata, documentText, renderDocument.
     * @param queryString the text of the query, using the Mimir query language.
     * @param queryId the ID for the query, as returned by the postQuery action.
     * @param rank the rank (position in the results list) for the requested document.
     * @param termPosition the index of the first token to be returned.
     * @param length the number of tokens to be returned. This parameter is optional, if
       omitted, all the document tokens will be returned.
     * @param fieldNames a comma-separated list of other field names to be returned.
     * @return An XML message with the ID of the new query, or an error message if there
       were any problems while parsing the query.
     */
        @GET
        @Path("{index_ID}/doc/search/{action}")
        @Produces("application/xml")
        public String getSearchDocOriented(@PathParam("index_ID") final String index,
                @PathParam("action") final String action,
                @DefaultValue("") @QueryParam("queryString") final String query,
                @DefaultValue("") @QueryParam("queryId") final String queryId,
                @DefaultValue("") @QueryParam("rank") final String rank,
                @DefaultValue("0") @QueryParam("termPosition") final String termPosition,
                @DefaultValue("") @QueryParam("length") final String length,
               @DefaultValue("") @QueryParam("fieldNames") final String fieldNames) {

                String res = "Try to start!";
                System.out.println("Query: " + query);
                try{
                 res = mimirDocService.getSearch(index, action, URLEncoder.encode(query,
                       "UTF-8"), URLEncoder.encode(queryId, "UTF-8"), rank, termPosition,
                       length, fieldNames);
                }catch(Exception e){
                    System.out.println("Exception with query: " + query + " error: " + e);
                }
                return res;
        }
```

```
/**
 * @param index_ID index ID
 * @param action Mimir action availables: postQuery, hitCount, docCount, docStats,
 * hits, getMoreHits, isActive, isComplete, renderDocument, documentMetadata,
   documentText.
 * @param queryString the text of the query, using the Mimir query language.
 * @param queryId the ID for the query, as returned by the postQuery action.
 * @param startIndex the first requested hit. A value of 0 requests the details for the
   first hit found.
 * @param count the number of hits for which the details are requested.
 * @param documentId the ID for the requested document, as returned by a call to the
   docStats action.
 * @param position the position of the first returned token. This parameter is
   optional; defaults to 0 is not provided, which means the first token of the
   document.
 * @param length the number of tokens to be returned. This parameter is optional, if
   omitted, all the document tokens will be returned.
 * @return An XML message with the ID of the new query, or an error message if there
   were any problems while parsing the query.
 */
   @GET
   @Path("{index_ID}/hit/search/{action}")
   @Produces("application/xml")
   public String getSearchHitOriented(@PathParam("index_ID") final String index,
                 @PathParam("action") final String action,
                 @DefaultValue("") @QueryParam("queryString") final String query,
                 @DefaultValue("") @QueryParam("queryId") final String queryId,
                 @DefaultValue("0") @QueryParam("startIndex") final String startIndex,
                 @DefaultValue("10") @QueryParam("count") final String count,
                 @DefaultValue("") @QueryParam("documentId") final String documentId,
                 @DefaultValue("0") @QueryParam("position") final String position,
                 @DefaultValue("10") @QueryParam("length") final String length){

         String res = "Try to start!";
         System.out.println("Query: " + query);
         try{
            res = mimirHitService.getSearch(index, action,
                  URLEncoder.encode(query, "UTF-8"), URLEncoder.encode(queryId,
                  "UTF-8"), startIndex, count, documentId, position, length);
         }catch(Exception e){
             System.out.println("Exception with query: " + query + " error: " + e);
         }
         return res;

   }
```

A.2 SCA QMS

```
   @Path("qms")
   public class QMSApplication {

       @Reference
       protected JaxrsQMSService qmsService;

      /**
       * This method allows to add some constraints in a XML format to the
       * Original query. The steps to do this are:
       * First, we get a new session id. Then, we submit the restrictions to
       * QMS. initially, we execute the mimir query to obtain a list of
       * queries.
       * @param xmlData: xml contraints.
       * Example:
          <request><query>test</query><numResults>10</numResults><constraints>
          <constraint><set>HONLabel</set><key>language</key><value>spa</value>
          <operator>EQ</operator></constraint></constraints></request>
```

```
     * @return XML format query
     */


    @POST
    @Path("mappings")
    @Consumes(MediaType.APPLICATION_XML)
    @Produces(MediaType.APPLICATION_XML)

    public MsgQueryMapping getQueryMappingwithConstraints(QueryData xmlData) {

            /*Firstly, we get a new session id */
            SessionBean session = qmsService.getSessionId();
            String sessionId = session.get_sessionId();
            xmlData.set_sessionId(sessionId);
            /* TO DO: Give a User Profile inside getSession(userProfile) method

            /*Secondly, we submit the restrictions to QMS */
            MsgConstraints message = qmsService.addConstraint(xmlData);

            /*TO DO: Finally, we execute the mimir query to obtain a list of queries */
            Query query = new Query(sessionId, xmlData.getQuery());
            MsgQueryMapping mimirQueries = qmsService.executeQuery(query);

            return mimirQueries;
            }

        /**
         * Obtaining session ID: Before a user can submit their queries to obtain mimir
         * queries, they need to obtain a session id. To do so they can send a GET to
         * the following URL.
         * http://services.gate.ac.uk/khresmoi/rest/service/session
         * This results in the following XML output
         * XML Output:
         *  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
         *  <message>
         *    <status>SUCCESS</status>
         *    <sessionId>khersmoi_1313937150704</sessionId>
         *  </message>
         *  @return sessionId: should be used for making further calls.
         */
        @GET
        @Path("mappings")
        @Produces(MediaType.APPLICATION_XML)
        public MsgQueryMapping getQueryMapping(
                    @QueryParam("userProfile") String useCaseName,
                    @QueryParam("keywords") String keywords) {

            /*Firstly, we get a new session id */
            SessionBean session = qmsService.getSessionId();
            String sessionId = session.get_sessionId();
            /* TO DO: Give a User Profile inside getSession(userProfile) method
            /*TO DO: Secondly, we submit the restrictions to QMS */
            /*TO DO: Finally, we execute the mimir query to obtain a list of queries */
            Query query = new Query(sessionId,keywords);
            MsgQueryMapping mimirQueries = qmsService.executeQuery(query);

            return mimirQueries;
            //return qmsService.getSessionId();
        }
    }
```

A.3 SCA Check Speller

```
@Path("wrapin")
public class WrapinApplication {

    @Reference
    protected JaxrsWrapinService wrapinService;

    /**
     * This method allows to suggest spelling correction of texts.
     * The typical use of this application is on the correction of the query that
     * the user has typed in the UI.
     * @param q Text to be Checked
     * @param wt Output format: json
     * @param spellcheck.count Number results
     * @param spellcheck.dictionary Dictionary where to check spelling
     * @param spellcheck.extendedResults Extended results
     * @return The service provide suggestion and all misspelled words will be
       encapsulated
     * within the <i> tag
     * @throws UnsupportedEncodingException
     */
    @GET
    @Path("speller")
    @Produces({"application/json;charset=UTF-8","text/xml;charset=UTF-8"})
    public Response getSpellingCorrection(
       @QueryParam("q") final String query,
       @DefaultValue("json")  @QueryParam("wt") final String format,
       @DefaultValue("1") @QueryParam("spellcheck.count") final int count,
       @DefaultValue("en")
       @QueryParam("spellcheck.dictionary") final String dictionary,
       @DefaultValue("false")
       @QueryParam("spellcheck.extendedResults") final String extResults)
    throws UnsupportedEncodingException{

       Gson gson = new GsonBuilder().setPrettyPrinting().registerTypeAdapter(
       Spellcheck.class,new SpellcheckDeserializer()).create();
       String response = wrapinService.getSpellingCorrection(query,format,count,
                                    dictionary,extResults);
       //return EntityUtils.toString(response, "ISO_8859_1");
       return Response.ok(response.getBytes("UTF-8")).build();
       }
}
```

A.4 SCA Dissambiguator

```
/** SCA Dissambiguator component implements Ontotext Dissambiguator REST Service in
 * order to integrate its functionality into the Textual Search workflow.
 * The supported languages are: english, german (de), czech (cz), spanish (sp),
 * french (fr)
 */
@Path("dissambiguator")
public class DissambiguatorApplication {

    @Reference
    protected JaxrsDissambiguationService dissambiguationService;



   /**
     * This method allows to dissambiguate the terms included in the query.
     * @param query Text to be processed
```

```
     * @param language The supported languages are:
     * english(default), german (de), czech (cz), spanish (sp), french (fr)
     * @return The service provide query dissambiguation with the following content:
     *
        * Binding schema: JSON
        * Object attributes:
                        count = the lucene score used for ordering
                        url = URL of the resource
                        labels = alternative names/synonyms
                        prefLabel = preferred label/display name
                        types = list of types for the resource
                        definition = short description of the resource
     * @throws UnsupportedEncodingException
     */
    @GET
    @Produces("application/xml")
    public DissambiguatedTerms getDissambiguationXml(
        @QueryParam("query") String queryText,
        @DefaultValue("en") @QueryParam("language") String lang)
    throws UnsupportedEncodingException {
        DissambiguatedTerms response = new DissambiguatedTerms();
        if (lang.equalsIgnoreCase("en"))
            response = dissambiguationService.getEnSpellingCorrection(queryText);
        else
            response = dissambiguationService.getSpellingCorrection(queryText, lang);

        return response;
    }

     /**
     * This method allows to dissambiguate the terms included in the query.
     * @param query Text to be processed
     * @param language The supported languages are:
     * english(default), german (de), czech
     * (cz), spanish (sp), french (fr)
     * @return The service provide query dissambiguation with the following content:
     *                 Binding schema: JSON
                       Object attributes:
                       count = the lucene score used for ordering
                       url = URL of the resource
                       labels = alternative names/synonyms
                       prefLabel = preferred label/display name
                       types = list of types for the resource
                       definition = short description of the resource
     * @throws UnsupportedEncodingException
    */
    @Path("json")
    @GET
    @Produces({"application/json;charset=UTF-8","text/xml;charset=UTF-8"})
    @Consumes("*/*")
    public Response getDissambiguationJson(
        @QueryParam("query") String queryText,
        @DefaultValue("en") @QueryParam("language") String lang) throws    ç
    UnsupportedEncodingException {
        String response = "";
        if (lang.equalsIgnoreCase("en"))
            response = dissambiguationService.getStrEnSpellingCorrection(queryText);
        else
        response = dissambiguationService.getStrSpellingCorrection(queryText,
        lang);
        return Response.ok(response.getBytes("UTF-8")).build();
        }
}
```

A.5 SCA MT

```
/**
* The MT component provides complete Machine Translation services for Khresmoi.
* The functionality includes translation of user queries from Czech, French, and
* German to English, translation of document summaries from English to Czech,French,
* and German and * * translation of full documents from English to Czech, French,
* and German.
* Optionally, the service provides multiple translation options for a given input.
* The translations can also be supplied with alignment information
* (which links parts of the input sentence with corresponding parts of the output
* sentence).
* The requests and responses conform the JSON format.
* A sample request is given below:
* {
*  "action": "translate", "sourceLang": "en", "targetLang": "de",
*  "text": "We all do hope, that this difference was not statistically significant."
*  }
*/
@Path("MT/translator")
public class MTApplication {

@Reference
protected JaxrsMTService multilingualService;

/**
  * GET Method:
  * The possible request parameters are:
  * @param action: string, function name, for testing purposes the only option is
  * translate
  * Stable version provides additional service functions (required)
  * @param sourceLang: string, ISO 639-1 code of the source language
  * (cs, en, de, fr)
  * @param targetLang: string, ISO 639-1 code of the target language
  * (cs, en, de, fr)
  * @param alignmentInfo: boolean value (0 or 1), should the word alignment be
  * included or
  * not (optional, default = 0, i.e. alignment not included)
  * @param nBestSize: integer, maximum number of candidates for translation
  * (optional,
  * default = 1, i.e. one best translation is provided, the maximum value is set to
  * 10).
  * @param text: string, text to be translated in UTF-8 character encoding
  * (required,
  * maximum length is limited to 100 words)
  * @return An string with JSON format with the Image examples
  */
 @GET
 @Path("get/simple")
 @Produces({"application/json","text/xml"})
 public String getTranslation(@QueryParam("action") final String action,
                @QueryParam("sourceLang") final String sourceLang,
                @QueryParam("targetLang") final String targetLang,
                @DefaultValue("0") @QueryParam("alignmentInfo") final String
                alignmentInfo,
                @DefaultValue("1") @QueryParam("nBestSize") final String nBestSize,
                @QueryParam("text") final String text){

       String response = multilingualService.getTranslation(
                                                action, sourceLang,
                                                targetLang, alignmentInfo,
                                                nBestSize, text);
       return response;
}
```

```
/**
  * POST Method:
  * The possible request parameters are:
  * @param action: string, function name, for testing purposes the only option is
  * translate
  * Stable version provides additional service functions (required)
  * @param sourceLang: string, ISO 639-1 code of the source language (cs, en, de,
  * fr)
  * @param targetLang: string, ISO 639-1 code of the target language (cs, en, de,
  * fr)
  * @param docType: string (reserved)
  * @param profileType: string (reserved)
  * @param alignmentInfo: boolean value (0 or 1), should the word alignment be
  * included or
  * not (optional, default = 0, i.e. alignment not included)
  * @param nBestSize: integer, maximum number of candidates for translation
  * (optional,
  * default = 1, i.e. one best translation is provided, the maximum value is set to
  * 10).
  * @param userId: string, globally unique user ID (optional in the dev version,
  * required
  * in the stable version, IDs will be issued by CUNI upon request)
  * @param text: string, text to be translated in UTF-8 character encoding
  * (required,
  * maximum length is limited to 100 words)
  * @return An string with JSON format with the Image examples
*/
@GET
@Path("get/advanced")
@Produces({"application/json","text/xml"})
public String postTranslation(@QueryParam("action") final String action,
                @QueryParam("sourceLang") final String sourceLang,
                @QueryParam("targetLang") final String targetLang,
                @QueryParam("docType") final String docType,
                @QueryParam("profileType") final String profileType,
                @DefaultValue("0") @QueryParam("alignmentInfo") final String
                alignmentInfo,
                @DefaultValue("1") @QueryParam("nBestSize") final String nBestSize,
                @QueryParam("userId") final String userId,
                @QueryParam("text") final String text){

   String response = multilingualService.postTranslation(action, sourceLang,
                                                         targetLang, docType,
                                                         profileType, alignmentInfo,
                                                         nBestSize, userId, text);
   return response;
}
```

A.6 SCA ParaDISE

```
/**
 * ParaDISE (the GNU Image-Finding Tool) is a Content Based Image Retrieval System.
 * It enables you to do Query By Example on images, giving you the opportunity to
 * improve query results by relevance feedback.
 */
@Path("paradise")
public class ParaDISEApplication {

    @Reference
    protected JaxrsParaDISEService paradiseService;
    @Reference
    protected JaxrsImageRepositoryService imageRepositoryService;

    /**
```

```
 * This method allows to search for similar images using full text based search.
 * @param maximumResults: text to search for in the image captions
 * @param query: text to search for in the article's full text
 * @return jJSON array with the Image search results
 */
@GET
@Path("fulltextSearch")
@Produces(MediaType.APPLICATION_JSON)
public String getImagesByQuery(
    @DefaultValue("50") @QueryParam("maximumResults") String maximumResults,
    @DefaultValue("") @QueryParam("query") String query){
    return paradiseService.searchByQuery(maximumResults,query);
}


/**
 * This method allows to search for similar images.
 * @param maximumResults: text to search for in the image captions
 * @param caption-query: text to search for in the image captions
 * @param relevant-images: JSON array containing URLs with relevant images OR a
 * URL to a JSON file containing such an array
 * @param irrelevant-images: same principle as the parameter above, only for
 * irrelevant images
 * @return JSON array with the Image search results
 * @throws UnsupportedEncodingException
 */
@GET
@Path("imageSearch")
@Produces({"application/json;charset=UTF-8","text/xml;charset=UTF-8"})
@Consumes("*/*")
public Response getImagesBySimilarity(
    @DefaultValue("50") @QueryParam("maximumResults") String maximumResults,
    @DefaultValue("") @QueryParam("captionQuery") String captionQuery,
    @DefaultValue("") @QueryParam("relevantImages") String relevantImages,
    @DefaultValue("") @QueryParam("irrelevantImages") String irrelevantImages)
throws UnsupportedEncodingException {
    String response =
    paradiseService.searchBysimilarity(maximumResults,captionQuery,
                                       relevantImages,irrelevantImages);
    return Response.ok(response.getBytes("UTF-8")).build();
}

/**
 * This method shows the URL prefixes to return the full images
 * @return XML with URL prefixes
 */
@GET
@Path("imagePrefix")
@Produces(MediaType.APPLICATION_XML)
public ImagePrefixBean getPrefixImage(){
    final String URL = "http://fast.hevs.ch/images/other/imageclef2012/";
    final String THUMB_URL =
    "http://fast.hevs.ch/images/other/imageclef2012_thumbs/";
    ImagePrefixBean imagePrefix = new ImagePrefixBean();
    imagePrefix.setUrl(URL);
    imagePrefix.setThumb_url(THUMB_URL);
    return imagePrefix;
}

/**
 * This method returns the full image of the Id given
 * @param id: Image identificator
 * @return
 */
 @GET
 @Path("images/{id}")
```

```
    @Produces({"image/jpeg","image/jpg","image/png"})
    public byte[] getImage(@PathParam("id") final String id){
              byte[] image = imageRepositoryService.getFullImage(id);
       return image;
    }

   /**
     * This method returns the image thumbnail of the Id given
     * @param id: Image identificator
     * @return
     */
    @GET
    @Path("images/thumbnails/{id}")
    @Produces({"image/jpeg","image/jpg","image/png"})
    public byte[] getThumbnail(@PathParam("id") final String id){
              byte[] image = imageRepositoryService.getImageThumbnail(id);
       return image;
    }
}
```

A.7 SCA MUW

```
/**
 * MUW is 3D image search engine that allows image searching through
 * different queries. SCA MUW implements the MUW REST API with the
 * following methods
 */
@Path("muw")
public class MuwApplication {

@Reference
protected JaxrsMuwService muwService;

        /**
         * This method allows to return an image given its Id
         * @param id: for example ID_RA10001172134000_3_1
         * @param size: NORMAL by default
         * @param direction: SAGITTAL by default
         * @param slice: 0 by default
         * @return XML with Image
         * @throws UnsupportedEncodingException
         */
        @GET
        @Path("image/{id}")
        @Produces("application/xml")
        public byte[] getImage(@PathParam("id") final String id,
       @DefaultValue("NORMAL") @QueryParam("size") final String size,
       @DefaultValue("SAGITTAL") @QueryParam("direction") final String
       direction,
       @DefaultValue("0") @QueryParam("slice") final String slice)
       throws UnsupportedEncodingException{
              byte[] image = muwService.getImage(id, size, direction,
                                                 slice);
              return image;
}


/**
* This method allows to return some series given an image Id
* @param id: for example ID_RA10001172134000_3_1
```

```
* @param size: NORMAL by default
* @param direction: SAGITTAL by default
* @return XML series
*/
@GET
@Path("image/series/{id}")
@Produces("application/xml")
public byte[] getSeries(@PathParam("id") final String id,
@DefaultValue("NORMAL") @QueryParam("size") final String size,
@DefaultValue("SAGITTAL") @QueryParam("direction") final String direction){
	byte[] image = muwService.getSeries(id, size, direction);
	return image;
}

/**
* This method allows to return a report given an image Id
* @param id: for example ID_RA10001172134000_3_1
* @return XML report
*/
@GET
@Path("report/{id}")
@Produces("application/xml")
public byte[] getReport(@PathParam("id") final String id){
	byte[] image = muwService.getReport(id);
	return image;
}

/**
* This method POST a Query object
* @param query : JSON format, for example: {
"queryImageID":"ID_RA10001172134000_3_1" }
* @return returns Result object
*/
@POST
@Path("query")
@Consumes({"application/json", "application/xml", "text/xml"})
@Produces({"application/json", "application/xml", "text/xml"})
public byte[] postQuery(String query){
	byte[] image = muwService.postQuery(query);
	return image;
}

/**
* This method POST a Query object
* @param query : JSON format, for example: {
"queryImageID":"ID_RA10001172134000_3_1"}
* @return returns Result object
*/
@POST
@Path("query/anatomy")
@Consumes({"application/json", "application/xml", "text/xml"})
@Produces({"application/json", "application/xml", "text/xml"})
public byte[] postAnatomy(String query){
	byte[] image = muwService.postAnatomy(query);
	return image;
}
/**
* This method POST a Query object
* @param query : JSON format, for example: {
```

```
"queryImageID":"ID_RA10001172134000_3_1" }
* @return returns Result object
*/
@POST
@Path("query/pathology")
@Consumes({"application/json", "application/xml", "text/xml"})
@Produces({"application/json", "application/xml", "text/xml"})
public byte[] postPathology(String query){
	byte[] image = muwService.postPathology(query);
	return image;
}
```