Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

Prototype and Evaluation of the 'Full Software Architecture'

Deliverable number	*D6.3.3*
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

Deliverable D6.3.3 of the KHRESMOI project describes the full software architecture prototype deployed that has been achieved after several months of coordinated work within the project. This prototype should accomplish every functional requirement from the scenarios provided as well as every other technical aspect regarding the system architecture. In this document we focus on describing the architecture proposed in previous prototypes as well as the refinement performed to achieve the full architecture system.

The main points that have taken into account to elaborate this report are related with the two technological approaches achieved to deploy this architecture prototype: the SOA services deployment and the layered-architecture design. These two concepts have allowed to develop the KHRESMOI full architecture system prototype by applying some SOA implementation as SCA Tuscany, that allows us to build a loose-coupling system of SCA Components based on some basic web services created by the partners, and make all of them work together in an easy to update system. Also, the layered architecture allows separating logically the functionalities of the system and in this manner to create different levels focused on specific tasks of the KHRESMOI system.

Therefore, this report presents the final stage in terms of Architecture deployment describing the main goals achieved as well as preparing the architecture to be able to apply any future changes over this full architecture that could be necessary to adopt.

Table of Contents

1 List of abbreviations

SOA	Service Oriented Architecture
SCA	Service Component Architecture
SOMA	Service-Oriented Modelling Architecture
REST	Representational State Transfer
API	Application Programming Interface
NC	Network Cohesion
NSIC	Number of Services Involved in a Compound Service
AIS	Absolute Importance of a Service
ADS	Absolute Dependence of a Service
ACS	Absolute Critically of a Service

Table 1: Abbreviations and acronyms

2 List of figures

3 List of tables

4 Introduction

4.1 Introductory Explanation of the Deliverable

This document aims to describe the final stage of the architecture refinement and deployment of KHRESMOI system. This phase represents the final step in order to collect all the work done previously within Task 6.2, Architecture Design. During the whole process of the Architecture Design these were the sub-tasks performed:

- T6.2.1: State-of-the-art and concepts of SOA
- T6.2.2: Specification (Early version)
- T6.2.3: Implementation of early prototype of the software architecture
- T6.2.4: Evaluation
- T6.2.5: Specification refinement (Full version)
- T6.2.6: Implementation of full prototype of the software architecture (project milestone)
- T6.2.7: Evaluation

In the first stage of Task 6.2 were included T6.2.1 and T6.2.2 where a designing and theoretical work was achieved in order to create some guidelines for the future tasks to be accomplished. After that, the Implementation and Evaluation of the Early Prototype architecture were implemented following the guidelines created previously and described deeply in deliverable D6.3.2 "Evaluation of the 'Early software architecture' and further specification" [1].

Therefore, this document related to "Prototype and evaluation of the Full Software Architecture" is focusing on T6.2.5 where it was performed a refinement process as well as on T6.2.6 and T6.2.7 where a final architecture prototype for the KHRESMOI System and its evaluation have been achieved.

4.2 Purpose and Audience

4.2.1 Purpose

The purpose of this deliverable is to describe final architecture deployed as the result of work performed in Task 6.2 during the first thirty months of the project. The "Full Software Architecture" and its implementation results as well as the results of the final evaluation of software architecture are presented.

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 processing of the KHRESMOI software architecture. As such this deliverable presents a description of the KHRESMOI Full Prototype software architecture so its importance increases and its implications could affect to any KHRESMOI member.

4.3 Structure of the Document

This deliverable is organized as follows: Section 5 describes the specification of the full prototype presented as final KHRESMOI architecture software system and Section 6 focuses on describing its evaluation approach. Section 7 includes a brief explanation about the tests scenarios exposed for KHRESMOI system. These scenarios will help us to know how successful is this prototype by achieving expected workflows. Related to these scenarios, Section 8 provides a results report by which are shown all the outcomes achieved by the proposed prototype. Section 9 presents the conclusions as well as main ideas and insights obtained to refine the system as much as possible. Finally, in Appendix Section future work related to full software architecture and refinement process is described.

5 Specification of the Full System Architecture

This chapter aims to describe the current specification for the KHRESMOI software system after different iterative developments and refinement processes. This specification is aligned with the first software specification described in [3] and pursues to complete it in order to accomplish KHRESMOI system requirements and functionalities. Although this specification can be considered as a final model, permanent evaluation process that is going to be performed over the KHRESMOI system platform until the end of the project may us to take into consideration any possible open issues that could be improved during the rest of the work to be done. These topics are described in more detail in Appendix chapter.

5.1 Specification of the Software System

The specification of the full architecture software proposed is mainly based in two principles:

- Layers-separated architecture
- Service Oriented Architecture (SOA) approach.

These main principles have been used during all refinement and development process so we will describe them in order to understand how the full software architecture has been designed.

On the one hand, the layered architecture represents the system as decomposed into three different layers, which correspond to the following functional blocks: the application, the services and the persistence of the system.

On the other hand, the SOA approach has been performed thorough Service Component Architecture (SCA) model that permits the assembly of services into business solutions. SCA provides many benefits that really fit with SOA requirements:

- Loose coupling system: components integrate with other without needing to know how other components are implemented.
- Easily invoked: services can be easily invoked either synchronously or asynchronously.
- Can easily be replaced: components can be replaced without many changes and that provides a high flexibility to the system.
- SCA simplifies development experience for all developers, integrators and application deployers.

In the Figure 1 we can see a diagram that represents these two principles in the full architecture software design. In general, the Full Prototype KHRESMOI System represents an innovative and advanced search engine for medical information.

From the architecture point of view KHRESMOI System is composed by two complementary sub-systems: the search system which provides support to the end users for information retrieval and the batch processing system which is in charge of the information analysis, retrieved by the different crawlers existing in the KHRESMOI System, and to index the information processed.

The search engine in the Full Architecture Prototype provides different modes for querying medical information. Queries can be defined with keywords, sentences, or conceptual descriptions (the keywords with the concept selection). But queries can be also defined with images. If an image is selected as a query, the system retrieves similar images according to graphical criteria (e.g. colour histogram, texture similarities, region or form detection, etc.). Finally, the system has taken into account the multilingual aspects. Due to that, three main functional blocks of services are required for

image, text and multilingual computation. These three main blocks are represented in Figure 1 by means of the centred green functional boxes in the Core Services Layer.

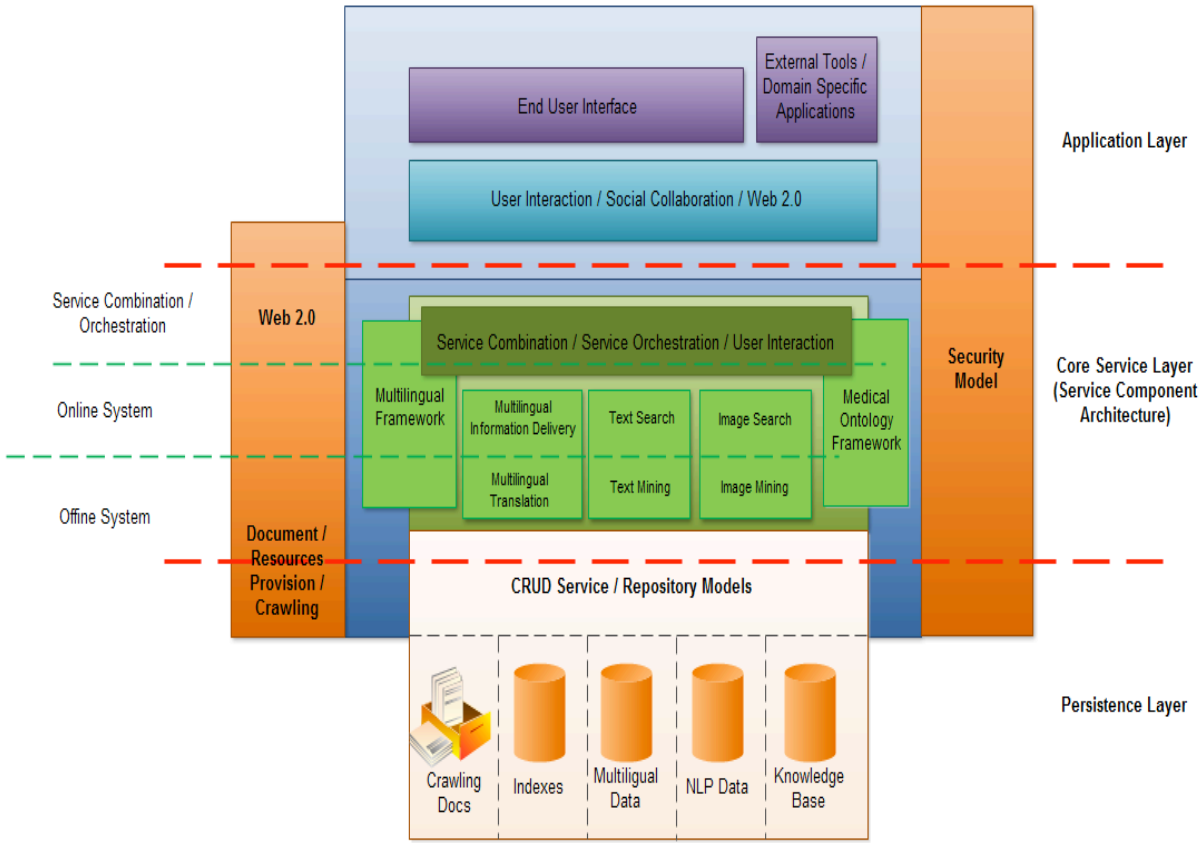

Figure 1. KHRESMOI FULL Logical Architecture.

Full Architecture Prototype provides a multi-lingual multi-modal search and access system for biomedical documents and knowledge, allowing querying in many languages, potentially combined with image queries. Next, we can highlight the main features of the Full Architecture Prototype:

- *Data access from many sources*: sources from which information have been extracted including peer reviewed journals (e.g. full-text of the Cochrane Systematic Reviews, and Open Access Biomedical journals) as well as other sources such as HON code certified websites, and biomedical repositories such as MEDLINE. Additional information extraction from sources as Wikipedia is currently an on-going task and therefore is not supported in last implementation of the System Architecture. Image sources include the images from biomedical publications, but also the images from Picture Archiving and Communication System (PACS) systems in two hospitals.
- *Multi-dimensional (2D, 3D, 4D) medical images Analysis:* images form is an important part of the information used in the medical field but are not taken into account by the majority of biomedical search engines. In particular, routines to analyse radiological data, including 3D (MRI, CT, PET), and 4D (with time, such as fMRI and MRI images taken at regular intervals) are currently supported.

- *Improved search capabilities*: Based on information extraction techniques, documents and images will be linked to facts in the biomedical knowledge base. The knowledge base will be used to improve search results (in particular recall), and to return targeted information items such as phrases (similar to question-answering) or relevant short passages.
- *Multi-lingual environment:* queries written in the user's own language will be used to search multi-lingually, but return summaries in the user's preferred language. In the project, priority is given to English, French, German, Spanish and Czech; however the framework (MOSES) is generic enough to integrate additional languages.
- *Providing understandable text and trustable results for the user:* the results are presented at a level suitable for the user, ranging from layperson to medical professional. Content search and selection will be personalized depending on the class and expertise of the user.

5.2 Application Layer

The Application Layer represents the application provided to the end users. According to the different use cases defined in the project (WP8 and WP9), different applications have been built to provide adapted user interfaces according to the specific requirements such as those from the radiologists (WP9). The Application Layer should deal with the configuration of the user interface, and the management of the user interaction to dispatch the events towards the Core Service Layer.

Mainly, this layer is covered by the ezDL tool. This component deals with all the user interactions, it means that it can intercept the different events and dispatch them towards the appropriate services. Thus, the Application Layer is composed of the user interfaces that are based on a Web Application framework such as AJAX and the controller of the user interactions. The controller is ezDL itself that has been extended to fit with the KHRESMOI requirements (for example, the logging of the user activities with the semantic annotation). Also, the controller contains a set of configurations to adapt the search engine to different situations (such as adaptation to a specific use case, adaptation to a user profile, selection of particular libraries, etc.). In some cases, it is needed to be connected with external tools to exchange interesting information about the users (like the tags, the social networks, etc.), or about the medical resources (like the new resources, the conceptual updates). Due to that, specific adapters have been implemented to enable those kinds of connections.

5.3 Core Service Layer

As we described previously the Service Layer is the core of the system. It contains all the main services provided by the system. These services have been specified with SCA and deployed through the runtime Apache Tuscany. Table 2 provides a summary of the main services defined in KHRESMOI Service Layer next to the functionalities exposed by each one of them.

Partner	Tool/Service	Functionalities
USFD	MIMIR, QMS	- Mimir – index annotations and text - Mimir – query indices - Query Mapping
ONTO	Owlim (exposed as Sesame repository)	- Persistent RDF data - Evaluate SPARQL query

		- Disambiguation (autocomplete)
HES-SO	ParaDISE	- Image database indexation - Query by example image retrieval - Support of user feedback for query refinement

UDE	ezDL	- 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. KHRESMOI Main Services

Finally, in the Core Service Layer are included the different main workflows or business processes defined in the KHRESMOI System associated to: Textual search, 2D Image Search, 3D Image Search and Multilingual Textual Search. In [6] are shown in detail each one of these business processes specified using SCA by means of a set of SCA Components and Composites.

5.4 Persistence Layer

The last layer is dedicated to the system persistency, which is why it is called the Persistence Layer. It is in charge of the mechanisms and models to store the information. For each kind of information, a repository is required to store the data. Each repository provides a basic API to describe its own CRUD (Create, Read, Update and Delete) functionalities to permit easy access to the data. After the first results of integration, a generic approach would be studied to define a standard API for services connexion to the KHRESMOI platform. In KHRESMOI, the most important repository is related to the storage of the knowledge, which is supported by means of OWLIM component. As the formalism to represent the knowledge in KHRESMOI is based on RDF, RDF repositories have an important place in the architecture.

6 Evaluation Approach

For the evaluation of the Full Architecture Prototype we have followed the same approach described in [1] dividing the architecture into domains, such as services, security, and governance and testing each domain separately using for each of them recommended approaches and tools as JUnit or Application Manager.

To evaluate these domains, we have taken as reference the metrics proposed by Rud, Schmietendorf, and Dumke [5] to evaluate Service-Oriented Infrastructures, which were defined in detail in [1]. The metrics selected were classified in three main categories, which are Complexity metrics, Critically and Reliability metrics, and Performance metrics.

Finally, Figure 2 shows the overall vision of the Full Prototype, which has been taken as reference to execute and calculate the metrics mentioned before.

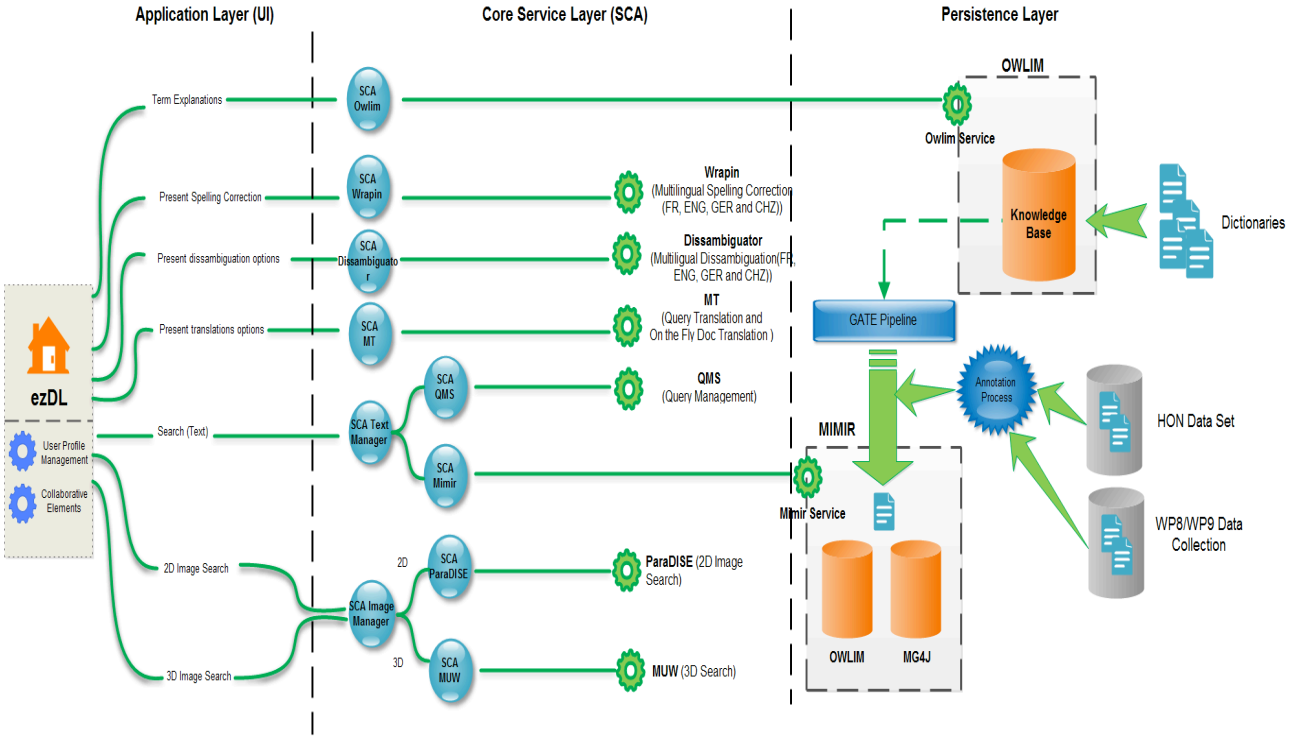

Figure 2. Full Prototype Overview

7 Tests Scenario Definition

This chapter describes the main test scenarios identified so far and remain the same as in the previous deliverables [1] [2] related to Architecture and Cloud Evaluation. Despite this, some minor changes have been done to these scenarios, in order to update the workflows to changes done during the last year and to update some components that have been replaced by some partners during this time. In these cases, the test scenarios workflows have been updated to new component functionalities. In addition to this, future changes may be adopted in next iterations and they will be evaluated along with cloud prototypes evaluation deliverables.

In this document, this chapter is focusing on architecture implementation and consequences related to these workflow mentioned before. These consequences will be taken into account for possible refinement of the software architecture.

7.1 Textual Search Scenario

This scenario represents one of the main workflows in the KHRESMOI project and uses most of the components created by all partners. From the previous Textual Search workflow, some updates and new functionalities are shown in Table 3:

Full Architecture Prototype Achievements	New and updated
Wrapin service updated to Speller service (HON)	Components, Table 4
Text Manager from Mimir: updated with method renderDocument()	Main Functionalities to be integrated, Table 4

Table 3. Textual Search scenario updates

The Textual Search workflow described in Table 4 contains the following components, steps and functionalities:

Prototype 1		**Textual search**
Components	ezDL (UDE)	ezDL is a multi-agent search system for heterogeneous data sources and a tool-set for building search user interfaces to support complex tasks
	Speller (HON)	HON's medical multilingual spell checking

		service
	Multilingual Translator (CUNI)	CUNI Multilingual translator service
	Disambiguator (ONTO)	ONTO Disambiguation service
	QMS (USFD)	Given a string of words and concept URIs, the service allows you to generate a Mimir query
	Mimir (USFD)	Mímir search Web Service
	TextManager (AtoS)	This component manages the complete Textual Search Workflow that integrates the other components and its iterations
Scenario	Step1 : search(keywords)	The user introduces a list of keywords through the UI in order to obtain a proper answer. In this version, the system allows also to add some constraints in order to adjust the search.
	Step2 : improve query	The user obtains some improvements to the query keywords performed by the system: spelling correction, possible language translation and disambiguation.
	Step3 : perform search of final query	The query is divided for different topics and the search is performed
	Step4 : return list of Documents	The user receives a list of documents with hits found as the answer of the search
	Step5 : view document	The user can render the document of the list returned from the search
Main functionalities to be integrated	Functionalities used by User: from ezDL	search(List<Keywords>)
	Functionalities used by User: from ezDL	viewDoc(docURI)
	Functionalities used by ezDL: from Speller	List<Suggestion> :getSpelling(keywords,lang)
	Functionalities used by ezDL: from MT	translateQuery(text,docType,sourceLang,targetLang,profile)
	Functionalities used by ezDL: from MT	translateDoc(docURI)

	Functionalities used by ezDL: from Disambiguation	List<Label> :getDissambiguation(keywords)
	Functionalities used by ezDL: from TextManager	searchByText(userProfile,keywords)
	Functionalities used by TextManager: from QMS	List<Query> :queryMapping(id,keyword)
	Functionalities used by TextManager: from Mimir	postQuery(index_id,queryString) hitCount(index_id,query_id) hits(index_id,query_id,start_index,count) docMetadata(index_id,query_id,doc_id) docText(index_id,query_id,rank,term_pos,length) renderDocument(doc_id)

Table 4. Textual Search scenario

7.2 2D Image Search Scenario

The Image Search workflow in KHRESMOI can be divided into two different sub-workflows: 2D Image Search and 3D Image Search.

In respect to 2D Image Search, the main change with respect to the last version of the scenario is that the previous image retrieval service (GIFT) has been replaced by ParaDISE:

Full Architecture Prototype Achievements	**New and updated**
GIFT service updated to ParaDISE service (HEVS)	Components, Table 5

Table 5. 2D Image Search scenario updates

We can observe in Table 6 the list of components, steps and functionalities involved in 2D Image Search workflow:

Prototype 1		**2D Image search**
Components	ezDL (UDE)	ezDL is a multi-agent search system for heterogeneous data sources and a tool-set for building search user interfaces to support complex tasks
	ParaDISE (HEVS)	HEVS content-based image retrieval service

	SCA-ParaDISE (AtoS)	SCA Component for ParaDISE
	Repository	Image repository
	ImageManager (AtoS)	This component manages the complete Image Search Workflow that integrates the other components and its iterations
Scenario	Step1 : search(images)	The user introduces a list of images through the UI in order to obtain a proper answer
	Step2 : the user obtains a list of images	The user obtains a list of images after the search processing ranked by a predefined score
Main functionalities to be integrated	Functionalities used by User: from ezDL	search(List<Image>)
	Functionalities used by ezDL: from ImageManager	get2DSimilarImages(List<ImageScore>, List<ImageCollection>, userProfile)
	Functionalities used by ImageManager: from SCA ParaDISE	searchBySimilarity(captionQuery, relevantImages, irrelevantImages)

Table 6. 2D Image Search scenario

7.3 3D Image search Scenario

The 3D Image Search workflow has suffered some changes related to its API since the last version described that:

Full Architecture Prototype Achievements	**New and updated**
The new methods exposed in the SCA REST Service are described and available on this url: http://khresmoicloud3.es.atos.net:8080/khresmoi-MUW/ • rest/muw/image/series/{id} • rest/muw/report/{id} • rest/muw/query • rest/muw/query/anatomy • rest/muw/query/pathology • rest/muw/image/{id}	Main functionalities to be integrated, Table 8

Table 7. 3D Image Search scenario updates

In Table 8**Error! Reference source not found.** we can see the list of components, steps and functionalities involved in 3D Image Search workflow:

Prototype 1		**3D Image search**
Components	ezDL (UDE)	ezDL is a multi-agent search system for heterogeneous data sources and a tool-set for building search user interfaces to support complex tasks
	MUW (MUW)	MUW 3D image retrieval web service
	SCA-MUW (AtoS)	SCA Component for MUW
	ImageManager (AtoS)	This component manages the complete Image Search Workflow that integrates the other components and its iterations
Scenario	Step1 : search(images)	The user introduces a list of images through the UI in order to obtain a proper answer
	Step2 : the user obtains a list of thumbnails from images	The user obtains a list of thumbnails from images similar to search performed
	Step3 : the user selects an image	The user selects an image thumbnail in order to obtain the full image
Main functionalities to be integrated	Functionalities used by User: from ezDL	search(Image,Text)
	Functionalities used by ezDL: from ImageManager	3DsimilarImages(image,text)
	Functionalities used by ImageManager: from SCA MUW	getImage(String id)

Table 8. 3D Image Search scenario

7.4 Multilingual Search Scenario

This scenario supports the dynamic translation of query text when it is introduced into the ezDL interface. This allows the user to find out terms in different languages that can improve the original query. The Multilingual Textual Search is considered as a special type of Textual Search workflow where the query text performed can be translated during the search. This workflow, shown in Table 9, is composed by the next components, steps and functionalities:

Prototype 1	**Multilingual query translation**	
Components	ezDL (UDE)	ezDL is a multi-agent search system for heterogeneous data sources and a tool-set for building search user interfaces to support complex tasks
	Multilingual Translator (CUNI)	CUNI Multilingual translator service
	SCA-MT (AtoS)	SCA Component for Multilingual Translator
Scenario	Step1 : translate(keyword)	The user introduces keyword through the UI in order to obtain a proper answer
	Step2 : the user obtains dynamically a list of possible translations	The user obtains a list of translations in a different language for the word introduced
	Step3 : the user selects a word	The user selects a word as a translation of the keyword entry
	Step4: Textual Search workflow	The rest of the scenario is the same as Textual Search workflow
Main functionalities to be integrated	Functionalities used by User: from ezDL	search(Image, Text)
	Functionalities used by ezDL: from SCA-MT	Translate(keyword)
	Functionalities used by SCA-MT: from Multilingual Translator	getTranslation(action, sourceLang, targetLang, text)

Table 9. Multilingual Search scenario

8 Results Report

This chapter aims to show the full architecture system prototype and their interactions that will allow us to know some useful statistical metrics related to them. These metrics have been already defined before for previous iterations of the architecture [1][2] so we will be able to compare the results along these different architecture implementations but we do not need to redefine them again. Once the specification of the full architecture system and the evaluation approach has been specified, the scenarios have been used to perform KHRESMOI System Testing.

The Figure 3 shows the full architecture interdependences and the number of direct calls among services. This graphic will be used in order to obtain valid metrics that may be useful to measure the main features of the system. Current version of diagram shows some changes with respect to previous version in terms of new components and others that were updated:

- SCA_Speller (SCA_SPE), SCA_Paradise (SCA _PAR), Speller_Service (SPES) and ParaDISE_Service (PARS) are updates of previous components.
- SCA_Multilingual translator (SCA_MT) has been added because is appears in the Multilingual Search workflow.

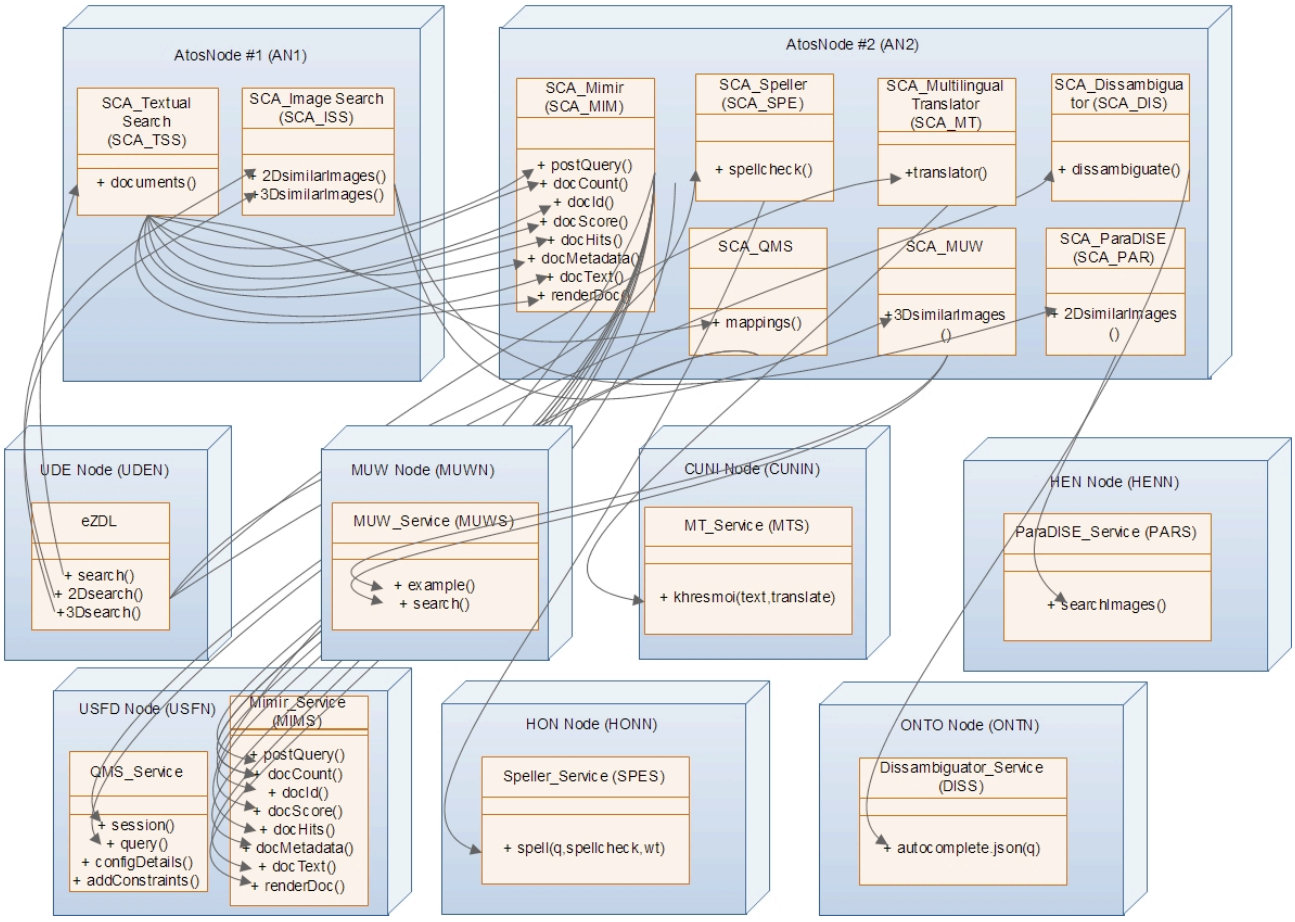

Figure 3. Khresmoi Component Nodes.

The next box shows the updated set A of calls between components (this set was already defined in deliverable [1]):

A = {<AN1, SCA_TSS, AN2, SCA_QMS, SCA_QMS.mappings()>,
<AN1, SCA_TSS, AN2, SCA_MIM, SCA_MIM.postQuery()>,
<AN1, SCA_TSS, AN2, SCA_MIM, SCA_MIM.docCount()>,
<AN1, SCA_TSS, AN2, SCA_MIM, SCA_MIM.docId()>,
<AN1, SCA_TSS, AN2, SCA_MIM, SCA_MIM.docScore()>,
<AN1, SCA_TSS, AN2, SCA_MIM, SCA_MIM.docHits()>,
<AN1, SCA_TSS, AN2, SCA_MIM, SCA_MIM.docMetadata()>,
<AN1, SCA_TSS, AN2, SCA_MIM, SCA_MIM.docText()>,
<AN1, SCA_TSS, AN2, SCA_MIM, SCA_MIM.renderDoc()>,
<AN1, SCA_ISS, AN2, SCA_PAR, SCA_ PAR.2DsimilarImages()>,
<AN1, SCA_TSS, AN2, SCA_MUW, SCA_MUW. 3DsimilarImages()>,
< AN2, SCA_MUW, MUWN, MUWS, MUWS.getSimilarImages()>,
< AN2, SCA_PAR, HENN, PARS, PARS.getSimilarImages()>,
< AN2, SCA_SPE, HONN, SPES, SPES.select()>,
< AN2, SCA_MT, CUNIN, MTS, MTS.khresmoi()>,
< AN2, SCA_MIM, USFN, MIMS, MIMS.postQuery()>,
< AN2, SCA_MIM, USFN, MIMS, MIMS.docCount()>,
< AN2, SCA_MIM, USFN, MIMS, MIMS.docId()>,
< AN2, SCA_MIM, USFN, MIMS, MIMS.docScore()>,
< AN2, SCA_MIM, USFN, MIMS, MIMS.docHits()>,
< AN2, SCA_MIM, USFN, MIMS, MIMS.docMetadata()>,
< AN2, SCA_MIM, USFN, MIMS, MIMS.docText()>,
< AN2, SCA_MIM, USFN, MIMS, MIMS.renderDoc()>,
< AN2, SCA_QMS, USFN, QMSS, QMSS.mappings()>,
< AN2, SCA_DIS, ONTN, DISS, DISS.autocomplete()>,
< UDEN, EZDL, AN1, SCA_TSS, SCA_TSS.documents()>,
< UDEN, EZDL, AN1, SCA_ISS, SCA_ISS.2DsimilarImages()>,
< UDEN, EZDL, AN1, SCA_ISS, SCA_ISS. 3DsimilarImages()>,
< UDEN, EZDL, AN2, SCA_SPE, SCA_SPE.spell()>,
< UDEN, EZDL, AN2, SCA_MT, SCA_MT.translator()>,
< UDEN, EZDL, AN2, SCA_DIS, SCA_DIS.dissambiguate ()>}

8.1 Scenario Independent Results

This section is focusing on describing metrics results independent from scenario as in previous prototypes evaluations. Most of the following metrics have been already described in previous deliverable [1], so we just present the new results obtained for full architecture software prototype.

8.1.1 Network Cohesion

This metric was already defined in [1]. This value can be calculated as follows:

$NC = |\pi_{InvokerNode,ServiceNode}(\sigma_{InvokerNode \neq ServiceNode}(A))| = \{<AN1, AN2>, <AN2, MUWN>, <AN2,GIFN>, <AN2, HONN>, <AN2, USFN>, <AN2, ONTN>, <AN2,CUNIN>, <UDEN, AN1>, <UDEN, AN2>\} = \mathbf{9}$

In our case, the KHRESMOI system shows a NC of 9. Related to the 9 nodes we have, it relies as a good value that ensures a good couple degree of the full system.

8.1.2 Number of Services Involved in a Compound Service

This metric was already defined in Deliverable [1]. In the full architecture software for the KHRESMOI system, the composed services are SCA_TSS, SCA_ISS and ezDL and the metric values for these cases is calculated as a simple sum of the number of associated services:

$NSIC[c] = |S^R[c]|$

$NSIC[SCA\_TSS] = |S^R[SCA\_TSS]| = \{SCA\_QMS, SCA\_MIM, QMSS, MIMS\} = \mathbf{4}$

$NSIC[SCA\_ISS] = |S^R[SCA\_ISS]| = \{SCA\_MUW, SCA\_ParaDISE, MUWS, PARS\} = \mathbf{4}$

$NSIC[ezDL] = |S^R[EZDL]| = \{SCA\_TSS, SCA\_ISS, SCA\_QMS, SCA\_MIM, SCA\_MUW, SCA\_ParaDISE, MUWS, PARS, SCA\_SPE, SPES, SCA\_DIS, DISS, SCA\_MT, MTS\} = \mathbf{13}$

The metric results over shows an equal value for SCA_TSS and SCA_ISS that are in charge of managing the business logic of the two main system workflows whereas that eZDL shows a higher value for this metric. This must be taken into account in terms of performance, so a high difference between these values for different services can affect to the complete system. For instance, the NSIC[ezDL] shows a value quite high comparing with Composites NSIC values, so therefore maybe new SCA Composites should be defined to get these values closer.

8.1.3 Services Interdependence in the System

This metric was already defined in [1]. There is no service interdependence in the KHRESMOI full architecture software system.

8.1.4 Absolute Importance of a Service

This metric was already defined in [1]. Calculating this value for KHRESMOI services we get the following results:

$AIS[s] = |\pi_{Invoker}(\sigma_{InvokerNode=s,\ ServiceNode \neq n}(A))|$

$AIS[ezDL] = \mathbf{0}$

$AIS[SCA\_TSS] = \mathbf{1}$

$AIS[SCA\_ISS] = \mathbf{1}$

$AIS[SCA\_SPE] = \mathbf{1}$

*AIS[SCA_DIS]=**1***

*AIS[SCA_QMS]=**1***

*AIS[SCA_MIM]=**1***

*AIS[SCA_MUW]=**1***

*AIS[SCA_PAR]=**1***

*AIS[SCA_MT]=**1***

*AIS[DISS]=**1***

*AIS[SPES]=**1***

*AIS[QMSS]=**1***

*AIS[MIMS]=**1***

*AIS[MUWS]=**1***

*AIS[PARS]=**1***

*AIS[MTS]=**1***

Analysing the results, all the services present a similar value so we can infer that none of the service can be consider as more important than other. This functionality can affect to availability of the system so an uncoupled architecture ensures availability of services.

8.1.5 Absolute Dependence of a Service

This metric was already defined in [1]. The results obtained above show a higher ADS value for eZDL so we can suppose it as the less autonomous service despite of DISS, SPES, etc. that are completely autonomous in our system.

$ADS[s] = |\pi_{Service}(\sigma_{Invoker=s,\ ServiceNode \neq n}(A))|$.

*ADS[ezDL]=**5***

*ADS[SCA_TSS]=**2***

*ADS[SCA_ISS]=**2***

*ADS[SCA_SPE]=**1***

*ADS[SCA_DIS]=**1***

*ADS[SCA_QMS]=**1***

*ADS[SCA_MIM]=**1***

*ADS[SCA_MUW]=**1***

*ADS[SCA_PAR]=**1***

*ADS[SCA_MT]=**1***

*ADS[DISS]=**0***

*ADS[SPES]=**0***

*ADS[QMSS]=**0***

*ADS[MIMS]=**0***

*ADS[MUWS]=**0***

*ADS[PARS]=**0***

*ADS[MTS]=**0***

8.1.6 Absolute Criticality of a Service

This metric was already defined in [1]. The results obtained for this metric are:

ACS[*s*] = *AIS*[*s*] × *ADS*[*s*].

ACS[*ezDL*] = *AIS*[ezDL] × *ADS*[ezDL]=**0**

ACS[*SCA_TSS*] = *AIS*[SCA_TSS] × *ADS*[SCA_TSS]=**2**

ACS[*SCA_ISS*] = *AIS*[SCA_ISS] × *ADS*[SCA_ISS]=**2**

ACS[*SCA_SPE*] = *AIS*[SCA_SPE] × *ADS*[SCA_SPE]=**1**

ACS[*SCA_DIS*] = *AIS*[SCA_DIS] × *ADS*[SCA_DIS]=**1**

ACS[*SCA_QMS*] = *AIS*[SCA_QMS] × *ADS*[SCA_QMS]=**1**

ACS[*SCA_MIM*] = *AIS*[SCA_MIM] × *ADS*[SCA_MIM]=**1**

ACS[*SCA_MUW*] = *AIS*[SCA_MUW] × *ADS*[SCA_MUW]=**1**

ACS[*SCA_PAR*] = *AIS*[SCA_PAR] × *ADS*[SCA_PAR]=**1**

ACS[*SCA_MT*] = *AIS*[SCA_MT] × *ADS*[SCA_MT]=**1**

ACS[*DISS*] = *AIS*[DISS] × *ADS*[DISS]=**0**

ACS[*SPES*] = *AIS*[SPES] × *ADS*[SPES]=**0**

ACS[QMSS] = *AIS*[QMSS] × *ADS*[QMSS]=**0**

ACS[*MIMS*] = *AIS*[MIMS] × *ADS*[MIMS]=**0**

ACS[MUWS] = *AIS*[MUWS] × *ADS*[MUWS]=**0**

ACS[*PARS*] = *AIS*[PARS] × *ADS*[PARS]=**0**

ACS[*MTS*] = *AIS*[MTS] × *ADS*[MTS]=**0**

Next table shows the criticity of the metrics related among them although they have the same value that in previous evaluation:

Metrics	**Criticity**		
	Very critical (>1)	Critical(=1)	Low criticity(=0)
ACS[ezDL]			×
ACS[SCA_TSS]	×		
ACS[SCA_ISS]	×		
ACS[SCA_SPE]		×	
ACS[SCA_DIS]		×	
ACS[SCA_QMS]		×	

ACS[SCA_MIM]	×	
ACS[SCA_MUW]	×	
ACS[SCA_PAR]	×	
ACS[SCA_MT]	×	
ACS[DISS]		×
ACS[SPES]		×
ACS[QMSS]		×
ACS[MIMS]		×
ACS[MUWS]		×
ACS[PARS]		×
ACS[MTS]		×

Table 10. Absolute Critically of a Service Matrix.

Aligned to previous deliverables, we can define as critical services the same that were already defined in [1].

8.1.7 Overall Reliability of a Compound Service

This metric was already defined in [1]. This metric aims to measure the reliability of a compound service by looking at its weakest component in terms of reliability. We can define overall reliability as follows:

$RC[c] = 1/\max(\{ACS[s] \mid s \in S^R[c]\})$.

$RC[\text{ezDL}] = 1/\max(\{ACS[s] \mid s \in S^R[\text{ezDL}]\}) = 1/2 = \mathbf{0.5}$

$RC[\text{SCA\_TSS}] = 1/\max(\{ACS[s] \mid s \in S^R[\text{SCA\_TSS}]\}) = 1/1 = \mathbf{1}$

$RC[\text{SCA\_ISS}] = 1/\max(\{ACS[s] \mid s \in S^R[\text{SCA\_ISS}]\}) = 1/1 = \mathbf{1}$

8.2 Scenario Dependent Results. Performance Metrics.

The prototype described so far has been focused on the specification and design of the current KHRESMOI system architecture software. After commenting some scenario independent results, the scenarios dependant results will be assessed in order to describe the full architecture software system.

Hence, this section is focused on describing the main performance metrics results related to each scenario identified in Chapter 7 which are: Textual Search, 2D Image Search, 3D Image Search and Multilingual Search. These scenarios represent the main KHRESMOI system workflows so it is very important to perform a comprehensive assessment of the main features that may affect the system: input/output messages size, network load, messages rate, etc. The three first scenarios have been already described in the previous prototype [1] so we will focus on describing the updates in the scenarios as well as the *Multingual Search Scenario* in this chapter.

The premises for the analysis of performance metrics are the same that those defined in previous evaluation deliverable [1].

8.2.1 Textual Search Scenario.

Figure 4 shows all the components involved in Textual Search scenario and all direct relations among them. This scenario was already described in deliverable [1]:

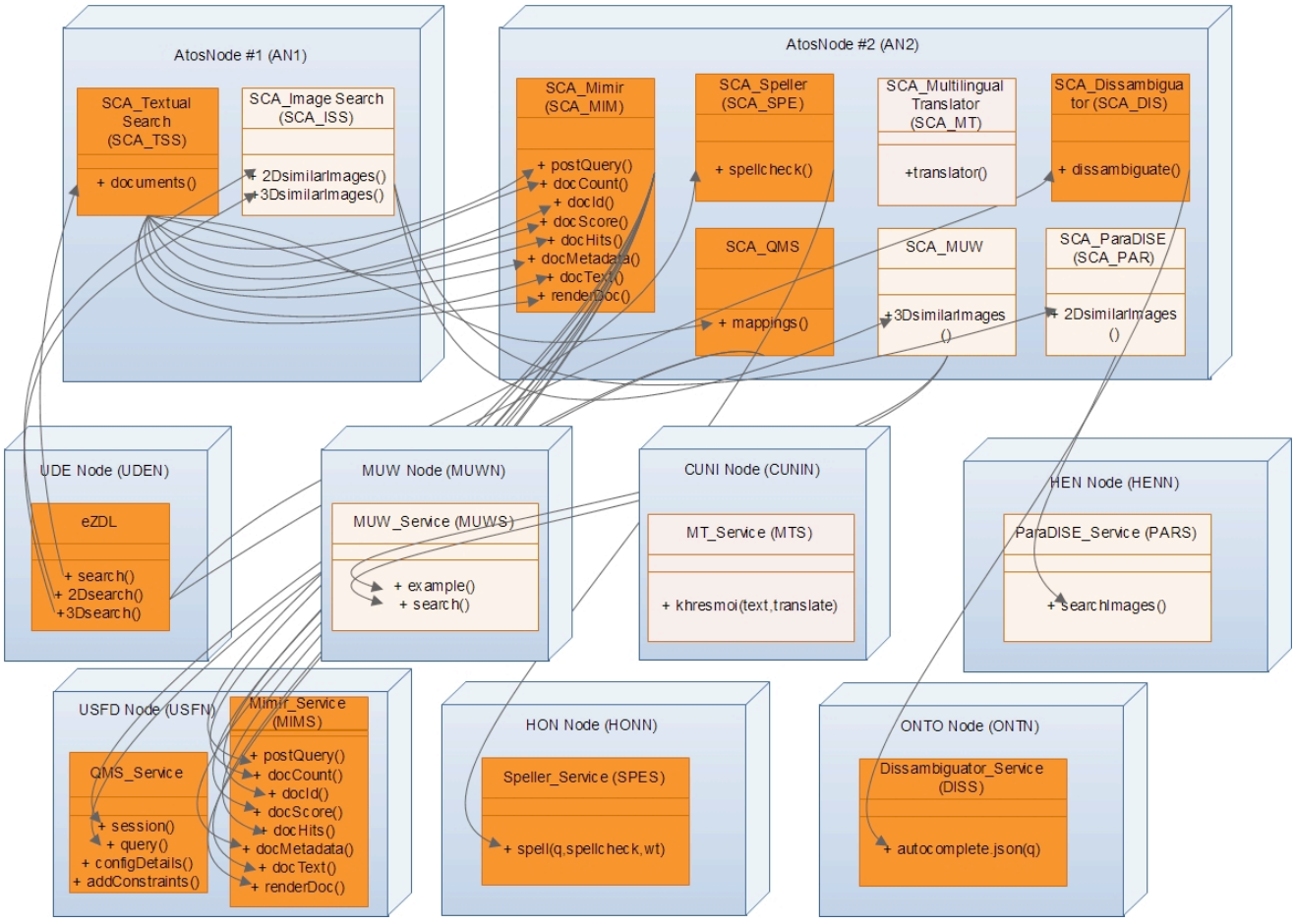

Figure 4. Textual Search Scenario components

Table 11 shows the updates and changes related to this scenario that may affect to the scenario dependant results:

Full Architecture Prototype Achievements	New and updated
SCA_SPE replaces SCA_WRP	Figure 4, AN2
renderDocument() method added to SCA_Mimir	Figure 4, AN2
Speller service new method : spell(q, spellcheck, wt)	Figure 4, HONN

Table 11. Textual Search scenario dependant results updates

The following sections focus on describing some metrics that were measured during last prototypes that we will also assess in this deliverable.

8.2.1.1 Sizes of Input and Output Messages

This metric was already defined in [1]. The metrics have been calculated performing different queries to Textual Search scenario:

- AN2.spellCheck : diabetess, lacctose, cancar

  AIMSO[SCA_SPE.spellCheck] = ***373 bytes***.
  AOMSO[SCA_SPE.spellCheck] = ***934 bytes***.

- AN2.dissambiguate : diabetes, lactose, cancer

  AIMSO[SCA_DIS.dissambiguate] = ***307 bytes.***
  AOMSO[SCA_DIS.dissambiguate] = ***2641 bytes.***

- AN1.search : diabetes, lactose, cancer

  AIMSO[SCA_TSS.search] = **384 bytes.**
  AOMSO[SCA_TSS.search] = **29504 bytes.**

We can observe in the results over that the main changes refer to AOMSO[SCA_TSS.search]. The reason is due to new updates and changes done in the backend (index, crawled pages) becomes in a different output size for textual search.

8.2.1.2 Messages Rates

This metric was already defined in [1]. Next, we can see some examples related to some Textual Search queries:

- AN2.spellCheck : diabetess, lacctose, cancar

IMRN[AN2] ≈ OMRN[AN2]= 395 mes./min

*MRN[AN2] = 395 + 395 = **790 mes./min***

- AN2.dissambiguate : diabetes, lactose, cancer

IMRN[AN2] ≈ OMRN[AN2]= **192 mes./min**

MRN[AN2] = 192 + 192 = **384 mes./min**

- AN1.search : diabetes, lactose, cancer

IMRN[AN1] ≈ OMRN[AN1]= 6 mes./min

MRN[AN1] = 6 + 6 = **12 mes./min**

The results above show very similar values compared with previous evaluation deliverables. This is due to the fact that the full architecture has been deployed in the already existing cloud system prototype, so the performance values remain quite constant as long as the system deployment is the same.

8.2.1.3 Network Load

This metric was already defined in [1]. Next, we can see some examples related to some Textual Search queries:

ITN[AN1] = N × *(384+29504)* = **6.4 calls/min. × 29888 bytes/call = 1,88 M/min**

OTN[AN1] = N × *(29504+384)* = = **6.4 calls/min. × 29888 bytes/call = 1,88 M/min**

ITN[AN2] = N × (*307+373+384+29504*) = **7.63 calls/min. × 30568 bytes/call = 1,71 M/min**

OTN[AN2] = N × *(29504+384+373+307)* = **7.63 calls/min. × 30568 bytes/call = 1,71 M/min**

The table above shows the results for network load, that are quite similar to previous prototypes, so we can infer that changes done to the system are not critical so far.

8.2.2 2D Image Search Scenario.

This scenario was already defined in [2]. The Figure 5 below shows the components involved in this scenario:

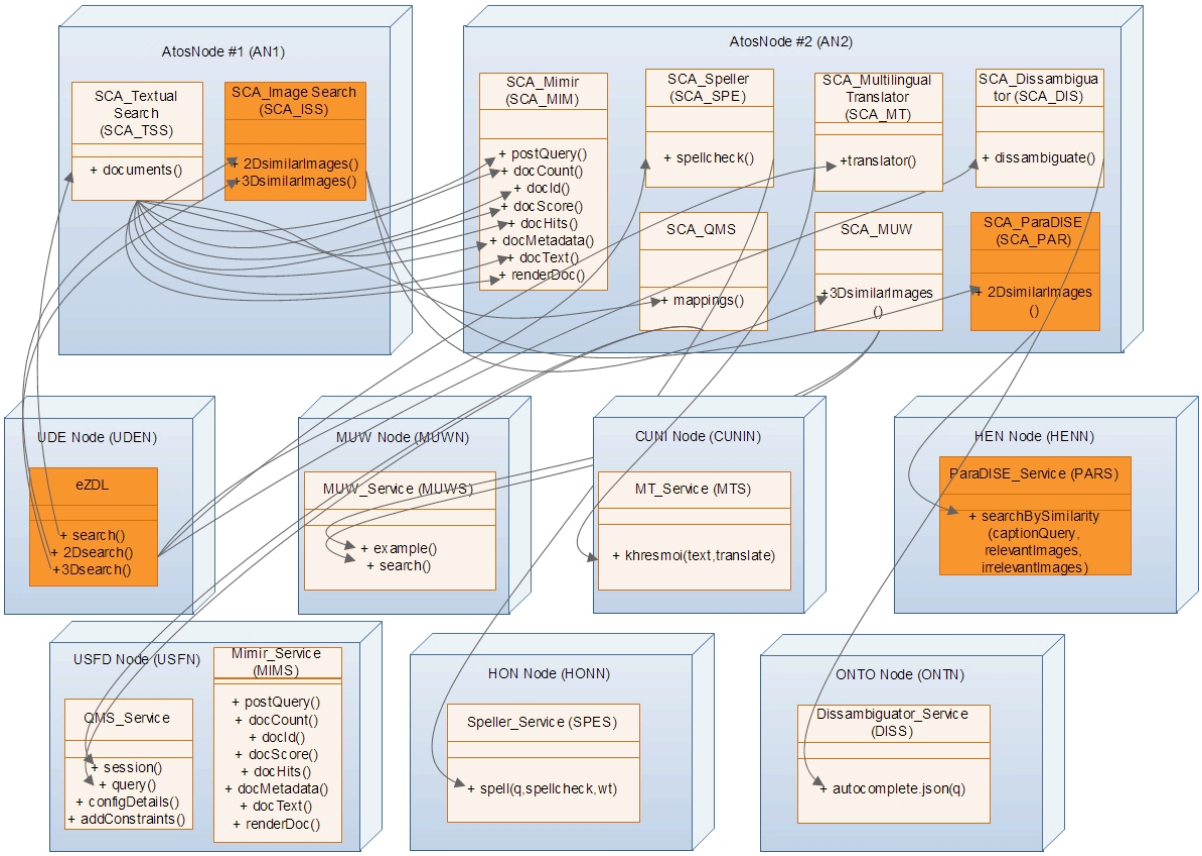

Figure 5. 2D Image Search scenario components

Table 12 shows the updates and changes related to this scenario that may affect the scenario dependant results:

Full Architecture Prototype Achievements	**New and updated**
SCA_ParaDISE replaces SCA_GIFT	Figure 5, AN2
ParaDISE service new method : searchBySimilarity(captionQuery, relevantImages, irrelevantImages)	Figure 5, HENN

Table 12. 2D Image Search scenario dependant results updates

Next, some metrics results related to this scenario will be described.

8.2.2.1 Sizes of Input and Output Messages

This metric was already defined in [1]. In respect to previous prototype, the Average Size of Input and Output Messages has changed due to new deployed method that returns JSON format instead some Image format. For this reason, current AOMSO value returns a higher value due to some queries that returns a list of images ids within the JSON returned:

- AN1.2DsimilarImages: example query: http://khresmoicloud2.es.atos.net:8080/khresmoi-image-search/rest/images/2D/similarImages/query?captionQuery=lactose

AIMSO[AN1.2DsimilarImages] = ***835 bytes.***

AOMSO[AN1.2DsimilarImages] = ***32631bytes.***

Despite this, this value can change by creating a more accurate query with relevantImages and irrelevantImages parameters.

8.2.2.2 Messages Rates

This metric was already defined in [1]. Anyway, next results are very similar to previous ones because cloud infrastructure has not been changed for architecture system, so performance remains stable for the system:

- AN1.2DsimilarImages : example query: http://khresmoicloud2.es.atos.net:8080/khresmoi-image-search/rest/images/2D/similarImages/query?captionQuery=lactose

IMRN[AN1] ≈ *OMRN[AN1]* = *254 mes./min*

MRN[AN1] = *254 + 254* = ***508 mes./min***

8.2.2.3 Network Load

This metric was already defined in [1]. Currently, the system is able to deal with the scenario bandwidth consumed so no critical changes are required.

ITN[AN1] = *N* × *(835+32631)* = ***N calls/min.*** × ***33466 bytes/call***

OTN[AN1] = *N* × *(32631+835)* = ***N calls/min.*** × ***33466 bytes/call***

8.2.3 3D Image Search Scenario.

This scenario was already defined in [1]. We can see below in Figure 6 the components involved in 3D Image Search workflow:

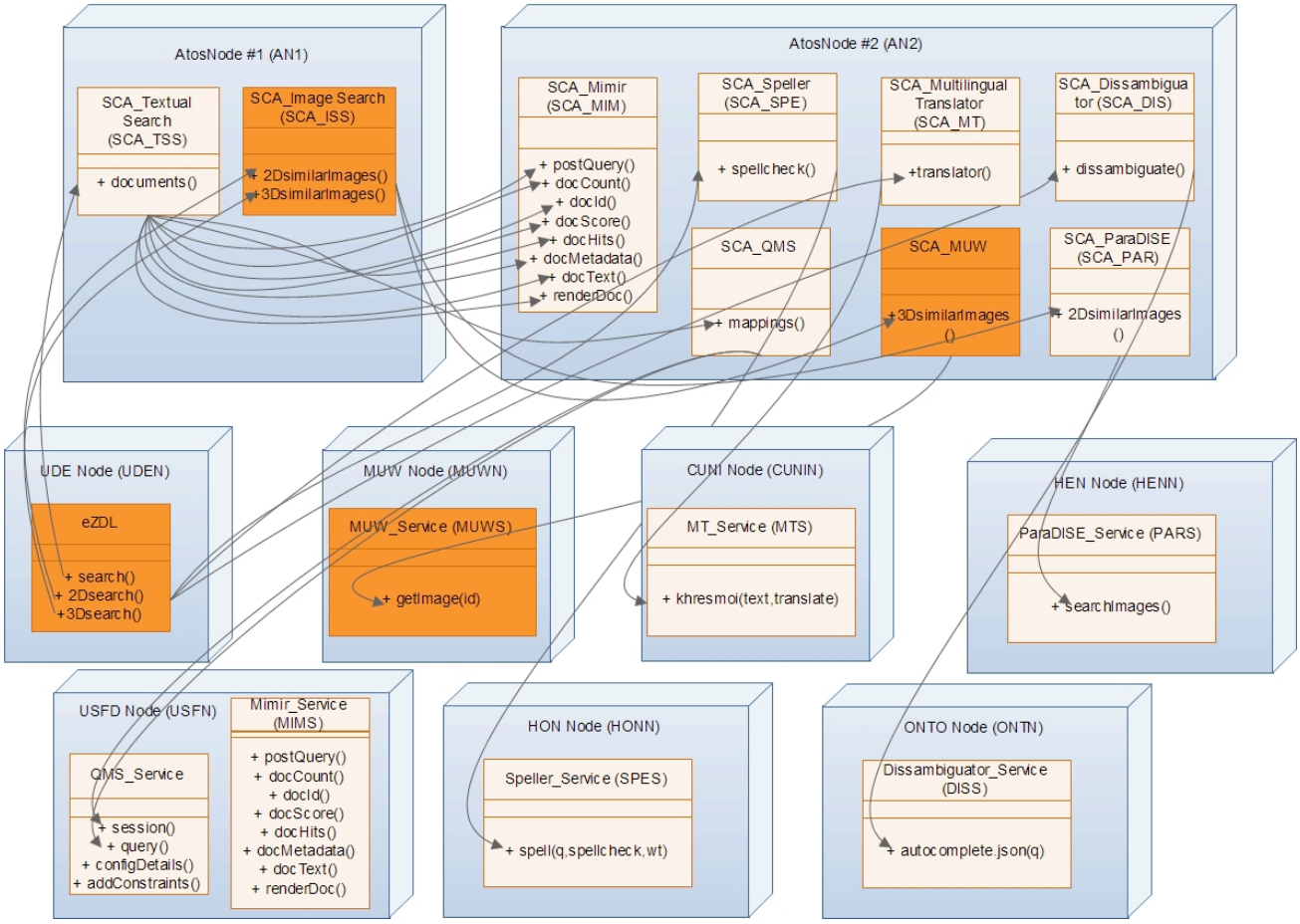

Figure 6. 3D Image Search scenario components

Table 13 shows main changes in this scenario with respect to previous iterations:

Full Architecture Prototype Achievements	**New and updated**
MUW service new method : getImage(id)	Figure 6, MUWN

Table 13. 3D Image Search scenario dependant results updates

Next, we will analyse some metrics results in order to evaluate the performance of the 3D image search scenario.

8.2.3.1 Sizes of Input and Output Messages

As mentioned before, method to obtain the image has changed so the sizes of input and output for the 3D Image Search has changed since last evaluation prototype deliverable:

- AN1.3DsimilarImages: example query

AIMSO[AN1.3DsimilarImages] = ***425 bytes.***

AOMSO[AN1.3DsimilarImages] = ***15852 bytes.***

8.2.3.2 Messages Rates

This metric was already defined in [1]. The results of this metric are similar to previous evaluations so changes in method seem not to affect the performance of the system:

- AN1.3DsimilarImages : example query

IMRN[AN1] ≈ *OMRN[AN1]*= *173 mes./min*

MRN[AN1] = *173* + *173* = ***346 mes./min***

8.2.3.3 Network Load

The importance of this metric resides in the fact of avoiding bottlenecks. It is a key in the performance of the system. In our case, we can check that full system architecture is able to deal with the Traffic Network Load described below:

ITN[AN1] = *N* × *(425+15852)* = ***N calls/min.*** × ***16277 bytes/call***

OTN[AN1] = *N* × *(425+15852)* = ***N calls/min.*** × ***16277 bytes/call***

8.2.4 Multilingual Textual Search Scenario.

This scenario is a special case of Textual Search scenario where an original query is translated "on the fly" before it starts the Textual Search workflow. For that, we have added a new multilingual translator component that will be in charge of doing the translation. The Figure 7 shows the component involved in this scenario:

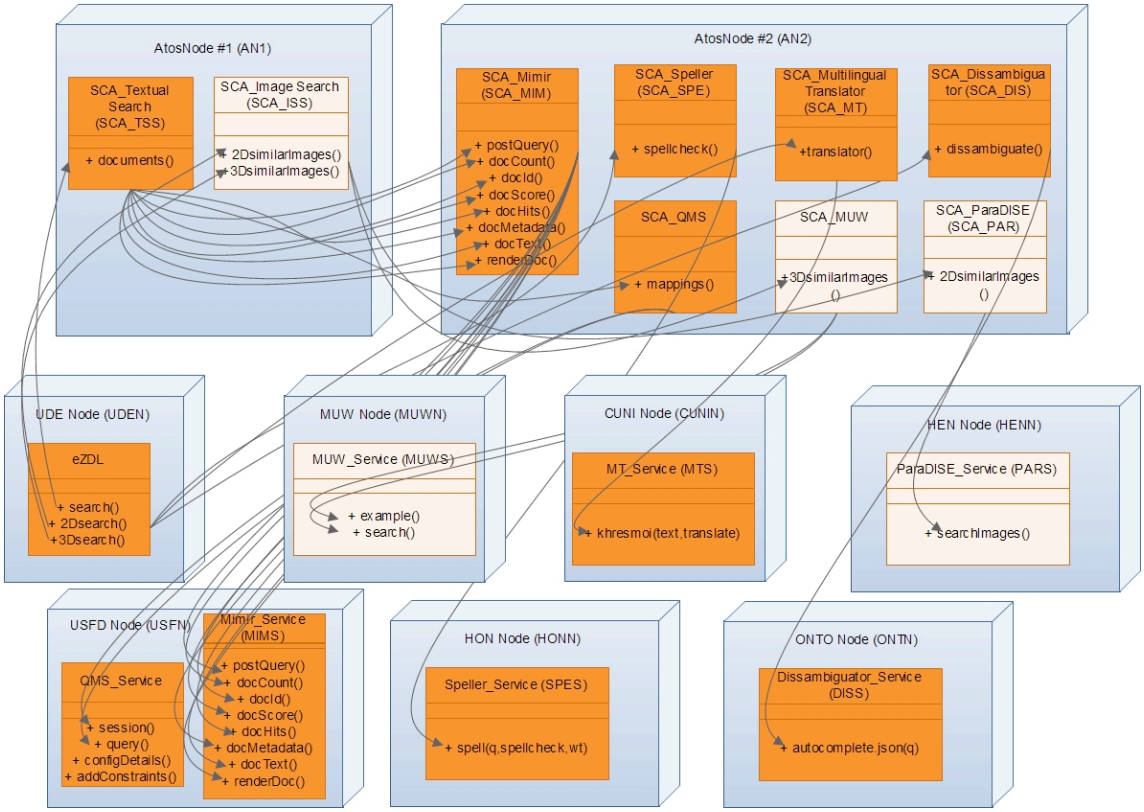

Figure 7. Multilingual Search in Khresmoi SOA.

The main changes and updates concerning this scenario, with respect to previous iterations, can be seen in Table 14:

Full Architecture Prototype Achievements	**New and updated**
SCA_MT (Multilingual Translator) added	Figure 7, MUWN
Method : khresmoi(text,translate) added on MT_ Service	Figure 7, CUNIN

Table 14. Multilingual Search scenario dependant results updates

Next, we are going to examine the different metric results to assess the system performance for this scenario.

8.2.4.1 Sizes of Input and Output Messages

This metric aims to measure the sizes of input and output messages related to Multilingual Search. Hence, we should assess both parts of the workflow: the dynamic translation and the Textual Search workflow that has been already assessed in 8.2.4.

- AN2.translator(): Zuckerkrankheit, laktózu, Krebs

AIMSO[SCA_MT.translator] = ***124 bytes.***

AOMSO[SCA_MT.translator] = ***265 bytes.***

- AN1.search : diabetes, lactose, cancer

AIMSO[SCA_TSS.search] = **384 bytes.**

AOMSO[SCA_TSS.search] = **29504 bytes.**

8.2.4.2 Messages Rates

For MRN metrics we have to take into account that we should add one more step in this workflow so that the total number of messages is bigger and therefore the metric increases:

- AN1.3DsimilarImages : example query

IMRN[AN1] ≈ *OMRN[AN1]= 234 mes./min*

MRN[AN1] = 234 + 234 = ***468 mes./min***

8.2.4.3 Network Load

The Network Load of the system increases in our case in the sense that a composed workflow consumes a higher network bandwidth than a simple one. Anyway, the system shows to be capable to deal with this workflow with a good performance:

ITN[AN1] = N × (124+265+ITN[TSS]) = ***N calls/min. × 9574 bytes/call***

OTN[AN1] = N × (265+124+ OTN[TSS]) = ***N calls/min. × 9574 bytes/call***

9 Conclusions

This document is mainly focusing on describing the final stage of architecture development that has been called as “full architecture prototype”. This prototype has been described as a final version of iterative developments and refinement where also the evaluation has been taking into account as an important point. The starting point was the design deliverable [3] where there was a general and theoretical description on the requirements of future KHRESMOI system as well as the technologies that could fill the gap between requirements and results. Based on this description, the first prototype was developed and evaluated in [1] where we started to obtain some valuable metrics related to the performance of the system to assess how the architecture design proposed behaves in the scenarios proposed so far.

After these first reports, this deliverable D6.3.3’s objectives are to gather all these first conclusions and insight in order to design a stable version of the full architecture prototype that could deal with all future work that remains until the end of the project. This process has been based in an evaluation process based in the metrics we used in [1] and also in some descriptive process where we explain all the technologies chosen to fulfil the goals proposed previously.

In summary, this document describes the KHRESMOI architecture system designed to fulfil the functional requirements identified so far and also prepares it for any possible future changes and updates that may be required during either some evaluation process, during the development of the cloud infrastructure or any other components update process. In this sense, due to the dynamic nature of the project, the flexibility of the proposed architecture may support any possible reengineering process of main scenarios use cases and workflows during last year of KHRESMOI project. This refinement process and future work related to full software architecture are described in Appendix chapter.

10 References

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

[2] Martinez Rodriguez, Ivan and Tinte Garcia, Miguel Angel, Deliverable D6.4.2 "Evaluation of the 'Early cloud infrastructure' and specification refinement for the 'Full cloud infrastructure' in KHRESMOI Project, Seventh Framework Programme.

[3] Jamin Enmanuel, Montchev Vassil, Pentchev Konstantin. Deliverable D6.3.1 "State of the art, concepts and specification for the 'Early software architecture'' in KHRESMOI Project, Seventh Framework Programme.

[4] (2007) Tony Harris Business Solutions: SOA Testing Methodology. www.thbs.com/soa.html

[5] D. Rud, A. Schmietendorf, R. Dumke, "Product metrics for service-oriented infrastructures," in Proceedings of "16th International Workshop on Software Measurement/DASMA Metrik Kongress 2006" (IWSM/MetriKon 2006), pp. 161-174, November 2-3, 2006, Potsdam, Germany

[6] Martinez Rodriguez, Ivan and Tinte Garcia, Miguel Angel, Deliverable D6.5.1 "Specification of the Early Integrated Infrastructure" in KHRESMOI Project, Seventh Framework Programme

11 Appendix

11.1 Architecture refinement and future work in KHRESMOI full architecture prototype

Despite this document aims to describe the full architecture software prototype as a system that plenty fulfils the KHRESMOI project requirements and expectations, this document also takes into account any possible changes that may be adopted in order to improve this system. In this sense, the architecture proposed should provide the mechanisms to adapt technologically and functionally the system in any critical point that may be detected. Therefore, next we are going to mention some important issues and topics that could be the reason of future refinement over the current presented architecture. Some basic divisions can be done between these topics:

- Integration Issues: these issues are those mainly focusing on any possible changes in components or workflows that may be necessary to adopt to improve the system.
- Multiple Search Issues: this topic refers to the fact that basic search performed from eZDL may be changed in order to join different search engines such as Mimir or Solr.
- Scalability Issues: the scalability issues refer to the problem of how to solve the concurrent access to KHRESMOI cloud system in order to deal with multiple requests as in a real platform scenario.

Figure 8 shows a graph with the main open issues described for the KHRESMOI project year 3:

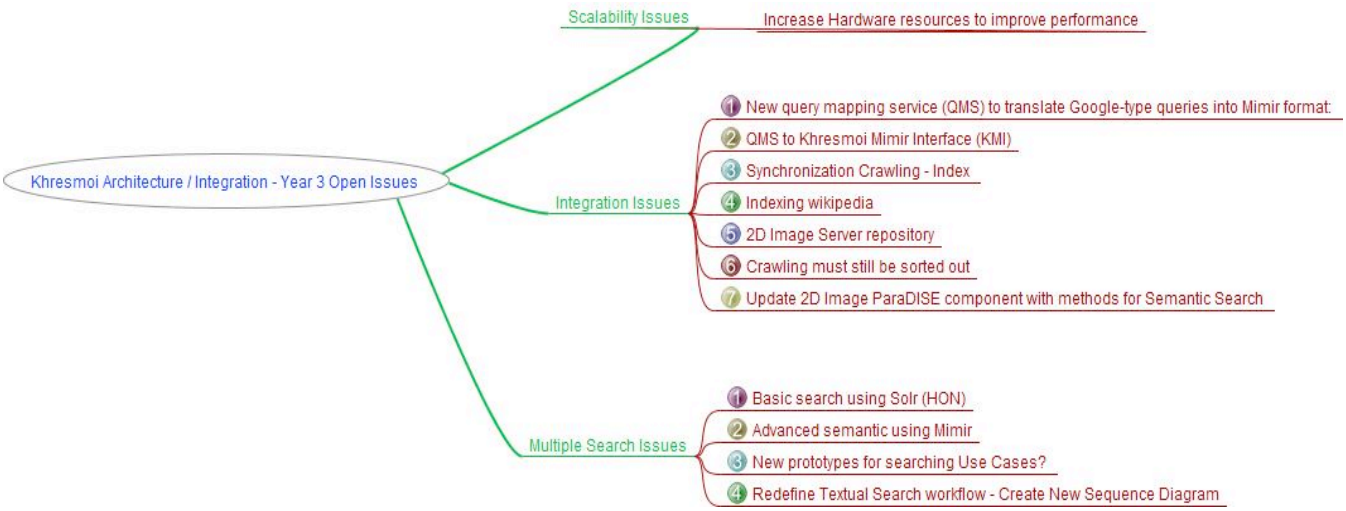

Figure 8. Khresmoi architecture open issues