Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

Evaluation of the 'Early software architecture' and further specification

Deliverable number	*D6.3.2*
Dissemination level	*Public*
Delivery date	*17 May 2012*
Status	*Final*
Author(s)	*Iván Martínez Rodríguez, Miguel Angel Tinte García*

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 D6.3.2 of the KHRESMOI project describes the approach that has been followed for the evaluation of the “Early Software Architecture”.

SOA applications and Web Services testing require a new approach and new techniques in order to change from code-level testing to user interface testing. This represents a huge challenge because working with SOA supposes not to know the logic model behind and sometimes without any existing user interface. Therefore, testing SOA architecture needs to be performed by dividing the system into different levels as governance, component-level, etc., and testing them independently.

Besides this, our system presents other specific features that must be taken into account during metrics definition. Concretely, we are not evaluating just a SOA system but we are evaluating also Tuscany SCA integration approach and Web Services performance through a Restful implementation. Therefore, the test type’s definition will have to consider all these premises to achieve an optimal evaluation.

The report presents the results of the evaluation applied on the first implementation of the “Early Software Architecture”. The objective consists of enriching the first architecture specification and providing more specific recommendations for the following developments, specifically with regard to requirements for large scale performance.

Table of Contents

1 List of abbreviations

SOA	Service Oriented Architecture
SCA	Service Component Architecture
EAI	Enterprise Application Integration
ESB	Enterprise Service Bus
UP	Unified Process
REST	Representational State Transfer
OCCI	Open Cloud Computing Interface
OOP	Object-oriented programming
CBSE	Component-based software engineering

Table 1: Abbreviations and acronyms

2 List of figures

3 List of tables

4 Introduction

4.1 Introductory Explanation of the Deliverable

The goal of task 6.2 "Architecture design", in which the Evaluation task is contextualized, is to specify and implement the KHRESMOI functional and detailed software architecture.

First, the specifications of the use cases help to determine the required functionalities for the KHRESMOI system. Then, based on the KHRESMOI functionalities, we will list the existing technologies (available components, services, API), and specify missing ones.

Finally, following the SOA principles, we will specify a distributed and flexible architecture. The architecture will be defined as 3-tier architecture:

- the persistence layer implements APIs to access knowledge, objects, etc.,
- the core services layer implements all the main services required for the KHRESMOI system,
- the application-services layer implements all the composition/orchestration services to develop specific applications.

Scaling the system depends on the components, processes involved, but primarily on the architecture design principles and their application in practice through the system integration. The success of the project depends heavily on the scale the resulting system is capable of covering. The task of scaling up the system will focus on iterative scale-up cycles involving evaluation and improvement of the key characteristics of the system with formal progress criteria.

4.2 Purpose and Audience

4.2.1 Purpose

The purpose of this deliverable is to enrich the first architecture specification (D6.3.1) and provide more specific recommendations for the next/subsequent developments, specifically for large scaling requirements.

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 platform. As such this deliverable presents the results of the evaluation applied on the first implementation of the "Early Software Architecture".

4.3 Structure of the Document

This deliverable is organized as follows: Section 5 describes the Evaluation Approach followed by the evaluation of the "Early Software Architecture". After evaluation approach in terms of phases in Section 5, Section 6 describes a set of metrics for each one of these phases. Section 7 provides a description of the main tests scenarios that have been defined according to initial WP8 and WP9 requirements. The metrics have been classified into three main categories, which are Complexity metrics, Critically and Reliability metrics and Performance metrics. Section 8

presents the results report and finally in Section 9, the conclusions of the deliverable are presented.

5 Evaluation Approach

The Architecture Validation and Testing in the scope of WP6 are technical; therefore, usability testing and field tests are not in the scope of WP6. Usability testing is done in WP3.

On the other hand, as we exposed in deliverable D6.3.1, we have defined SOA architecture as the solution for the KHRESMOI platform. Now, we have to (a) divide the architecture into domains, such as services, security, and governance and (b) test each domain separately using recommended approaches and tools. We will follow a SOA Testing Methodology approach [1].

Figure 1 represents the KHRESMOI model of SOA components and how they are interrelated. The Validation team designing the Project Validation approach and plans must have a macro understanding of how all of the components work independently and collectively.

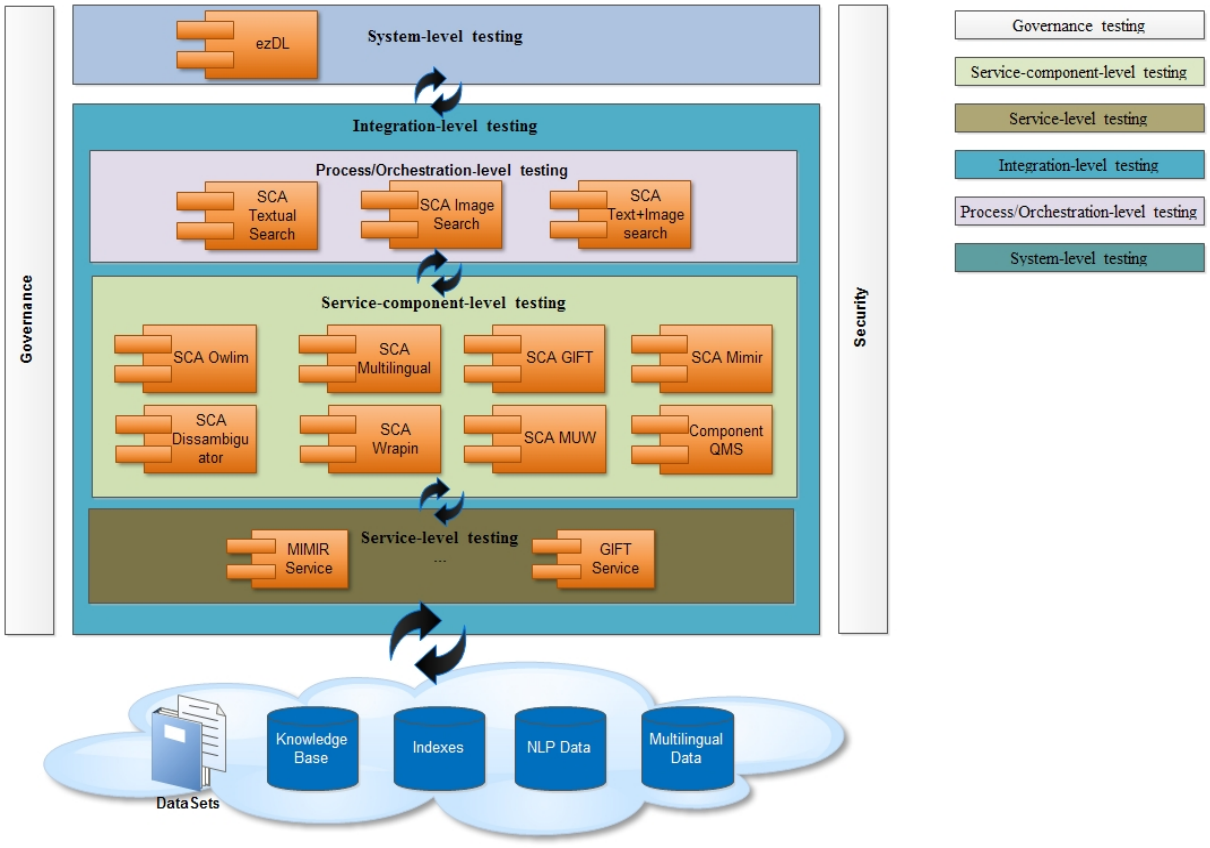


Figure 1. Khresmoi SOA Domains.

We propose to categorize SOA validation into the following phases:

- Governance testing
- Service-component-level testing
- Service-level testing
- Integration-level testing

- Process/Orchestration-level testing
- System-level testing

Next we detail each one of these phases with the test type proposed.

5.1 Governance Testing

SOA Governance is a key factor in the success of any SOA Implementation. It is also the most 'loosely' used term, as it covers the entire lifecycle of SOA Implementation – from design to run time to ongoing maintenance. SOA Governance refers to the Standards and Policies that govern the design, build and implementation of a SOA solution and the Policies that must be enforced during runtime.

Test cases will be constructed and executed in all of the project test phases to determine if SOA Policies are being enforced. SOA policies can be enforced at runtime, by using technologies and/or monitoring tools.

SOA Governance testing will not be a separate test phase. Testing that SOA Governance is enforced and will take place throughout the project life cycle, through formal peer reviews and different test scenarios that will be executed during the separate test phases.

5.2 Service-component-level Testing

Service-component-level testing or Unit testing, is normally performed by the developers to test that the code not only successfully compiles, but the basic functionality of the components and functions within a service are working as specified.

The primary goal of Component testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as we expect. Each Component is tested separately before integrating it into a service or services.

The following quality and test activities are recommended in this phase/level of testing:

- Formal peer reviews of the code to ensure it complies with organization standards and to identify any potential performance and security defects or weaknesses.
- Quality entry and exit criteria are not only defined for this level of testing, but are achieved before moving to the next level of testing.

5.3 Service-level Testing

Service testing will be the most important test level/phase within our SOA Test approach. It is best practices when we build a Web service to perform limited unit testing and accelerate its delivery to the integration test phase, to allow the test team to evaluate its quality. Service reuse will demand each service to be delivered following this level/phase of testing with a comprehensive statement of quality and even a Guarantee.

The following quality and test activities are recommended in this phase/level of testing:

- Formal peer reviews of the code to ensure it complies with organization standards and to identify any potential interoperability, performance and security defects or weaknesses.
- Functional, performance and security regression suites to be executed against the service. This will require the help of automated test tools and the development of sophisticated harnesses and stubs.

- Quality entry and exit criteria are not only defined for this level of testing, but are achieved before delivering the service to the next level of testing.

Service Level testing must ensure that the service is meeting the business and operational requirements of the Khresmoi project extracted from WP8 and WP9.

5.4 Integration-level Testing

The integration test phase will focus on service interfaces. This test phase aims to determine if interface behavior and information sharing between the services, are working as specified. The test team will ensure that all the services delivered to this test phase comply with the defined interface definition, in terms of standards, format and data validation. Integration testing test scenarios should also 'work' the layers of communications, the network protocols.

5.5 Process/Orchestration-level Testing

Process/Orchestration testing ensures services are operating collectively as specified. This phase of testing would cover business logic, sequencing, exception handling and process decomposition (including service and process reuse).

5.6 System-level Testing

System Level testing will form the majority, if not all of the User Acceptance Test phase. This test phase will test that the SOA technical solution has delivered the defined business requirements and has met the defined business acceptance criteria. To ensure that this phase/level of testing is targeting only the key business scenarios of the solution, the business stakeholders and testers must fully understand the quality and test coverage that has been achieved in previous test phases.

6 Evaluation Measures

We have specified the evaluation approach in terms of phases in Section 5, and in this section we need to define a set of metrics for each one of these phases. For that purpose, we have taken as reference the metrics proposed by Rud, Schmietendorf, and Dumke [2] to evaluate Service-Oriented Infrastructures. The metrics are classified in three main categories, which are Complexity metrics, Criticality and Reliability metrics, and Performance metrics. In next subsections we explain in details each one of them.

We will take as reference the definition of the following operations needed to understand the formulas associated to each one of the metrics:

- Selection $\sigma_{condition,\ ...}(R)$: selects tuples (rows) from the relation *R*, which conform to the given set of conditions.
- Projection $\pi_{attribute,\ id,\ ...}(R)$: selects attributes (columns) from the relation *R*, eliminating duplicates.
- Renaming $\beta_{new\_atrr\_id \leftarrow old\_atrr\_id,...}(R)$: renames attributes in the relation R.

The following quantities and symbols will be used in the definition of the metrics:

- N – Set of service providers' nodes.
- S[n] – Set of services provided by the node n ϵ N.
- S[*] – Set of services from all nodes.
- C[*] – Set of compound services from all nodes.
- S^R [c] – Set of services that take part in the compound service $c \epsilon C[*]$, both directly and indirectly.
- B[m, n] – (Expected) invocation rate of the operation $m \epsilon M[*]$ produced by clients located on the node $n \epsilon N$.
- M[*] – Set of operations of all services of all nodes.
- A – Set of tuples (relation) <*InvokerNode, Invoker, ServiceNode, Service, Operation*>. The meaning of each tuple is "Software component *Invoker* that is located on the node *InvokerNode* invokes the operation *Operation* of the service *Service* that is located on the node *ServiceNode*".

6.1 Complexity Metrics

6.1.1 Network Cohesion

This metric is determined as the count of direct (unidirectional) ties between the nodes in the system:

$$NC = |\pi_{InvokerNode,ServiceNode}(\sigma_{InvokerNode=ServiceNode}(A))|$$

A unidirectional tie between nodes *n*1 and *n*2 exists when the composition logic supposes an invocation of a service provided by the node *n*2 from a client located on *n*1.

6.1.2 Number of Services Involved in a Compound Service

NSIC[*c*] – Number of Services Involved in the Compound service $c \in C^R[*]$ (both directly and indirectly):

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

This metric is the simplest complexity indicator of a compound service. Other properties of service compositions will be investigated in the next subsection, where we will discuss criticality and reliability aspects.

6.1.3 Services Interdependence in the System

SIY – Services Interdependence in the System is the count of pairs of services which depend on each other. This value can be determined as:

$$SIY = |\beta_{Service1 \leftarrow Invoker,Service2 \leftarrow Service}(\pi_{Invoker,Service}(A)) \cap \beta_{Service2 \leftarrow Invoker,Service1 \leftarrow Service}(\pi_{Invoker,Service}(A))|$$

Service interdependence attests poor services' design and therefore should be avoided – for example, by combining interdependent services to a coarse-grained one, if they belongs to the same provider.

6.2 Criticality and Reliability Metrics

6.2.1 Absolute Importance of the Service

AIS[*s*] – Absolute Importance of the Service $s \in S[n]$, $n \in N$ is defined as count of clients which depend on *s*, i.e. which invoke its operations. Note that we do not count here clients that are located on the node *n*, because:

- Communication between pieces of software that are located on the same node does not produce additional network traffic.
- It does not seem to be very likely that one service will become unavailable while other services residing on the same node will not.

The formula for *AIS*[*s*] is:

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

6.2.2 Absolute Dependence of the Service

ADS[*s*] – Absolute Dependence of the Service $s \in S[n]$, $n \in N$ is defined as count of other services this service depends on:

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

This metrics is inversely proportional to the self-sufficiency of the given service, and, consequently, to the level of its potential autonomy and reusability in another environment.

6.2.3 Absolute Criticality of the Service

ACS[*s*] – Absolute Criticality of the Service $s \in S[n]$, $n \in N$ is the product of its absolute importance and absolute dependence:

$$ACS[s] = AIS[s] \times ADS[s]$$

The rationale behind this metric is the following: neither an important service that does not depend on a plenty of other services nor an unimportant one that depends on many others are as critical for the system as a service that is both very important and highly dependent. To some extent, the *ACS*[*s*] metric represents the degree of attention that the designer of the service-oriented system should pay to the service *s*.

6.2.4 Overall Reliability of the Compound Service

RC[*c*] – Overall Reliability of the Compound service $c \in C[*]$ is inversely proportional to "the weakest link of a chain", i.e. to the reliability of the most critical service that takes part in *c*:

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

6.3 Performance Metrics

6.3.1 Sizes of Input and Output Messages

The number of arguments of a method (of a function, of an operation, etc.) is a classic design metric in the OOP and CBSE domains. But it is not always applicable to service operations because under the document interaction style all values (e.g. name, address and phone number of a customer) will be "packed" into one coarse-grained data structure, which can also be weak, i.e.some of its parts/elements may be optional. Therefore our metric must relate not to the number of parameters, but to their absolute size in bytes.

In doing so, size overhead conditioned by the messages format (e.g. XML tags and auxiliary headers in SOAP) must be considered as well. The meaning of such metric in the case of SOA lies not only in design and interface complexity (as in the case of OOP and CBSE), but – due to the network-based nature of communication between services – rather in performance.

If an operation sends some data back to the caller (return value or output parameter), the size of this data must be taken into account as well. So we can define the following two metrics:

• *AIMSO*[*m*] – Average Input Message Size of the Operation $m \in M[*]$.

• *AOMSO*[*m*] – Average Output Message Size of the Operation $m \in M[*]$. This value equals to zero for one-way operations (which are used primarily under the document interaction style).

To determine their values, knowledges of both business process' domain and technical infrastructure are necessary.

Average Message Size in the System can be calculated as

$$AMSY = (\sum_{m \in M[*]} AIMSO[m] + \sum_{m \in M^{RR}[*]} AOMSO[m]) / |M[*]| + |M^{RR}[*]|$$

where $M^{RR}[*]$ is a subset of M[*] containing request-response operations (i.e. operations that require/imply both input and output messages).

6.3.2 Message Rates

An important metric of a node that is providing services is the total number of input and output messages this node has to handle during one unit of time – i.e. to parse messages when they arrive (unmarshalling) and to generate messages when some data must be sent (marshalling), respectively.

Message processing consumes CPU time, memory and other resources of the node; therefore it could be expedient to assess it.

We introduce the following metrics here:

• *IMRN*[*k*] – Incoming Message Rate of the Node *k* 2 N:

$$IMRN[k] = \sum_{n \in N} \sum_{m \in M[k]} B[m, n]$$

• *OMRN*[*k*] – Outgoing Message Rate of the Node *k* 2 N:

$$OMRN[k] = \sum_{m \in M[*]} B[m, k]$$

• *MRN*[*k*] – Overall Message Rate of the Node $k \in N$. It can be supposed that the amounts of resources which are being consumed to handle an incoming message and an outgoing message are roughly equal. Therefore we can add their rates together and use this sum later on to estimate the total consumption of computing resources:

$$MRN[k] = IMRN[k] + OMRN[k]$$

6.3.3 Network Load

Based on the *AIMSO*[*m*] and *AOMSO*[*m*], the overall amount of data sent and received by a node during one unit of time can be estimated. The metrics for incoming and outgoing traffic base on expected invocation rates *B[m, n]* introduced in the previous subsection and are listed below:

• *ITN*[*k*] – Incoming Traffic of the Node *k* 2 N consists of:

- input parameters of operations of services located on this node, called by services located on other nodes, and
- return values of operations of services of other nodes, called by services located on this node:

$$ITN[k] = \sum_{n \in N \setminus \{k\}} \sum_{m \in M[k]} (B[m, n] \times AIMSO[m]) + \sum_{n \in N \setminus \{k\}} \sum_{m \in M[n]} (B[m, k] \times AOMSO[m])$$

• *OTN*[*k*] – Outgoing Traffic of the Node *k* 2 N consists of:

- input parameters of operations of services located on other nodes, called by services located on this node, and
- return values of operations of services of this node, called by services located on other nodes:

$$OTN[k] = \sum_{n \in N \setminus \{k\}} \sum_{m \in M[n]} (B[m, k] \times AIMSO[m]) + \sum_{n \in N \setminus \{k\}} \sum_{m \in M[k]} (B[m, n] \times AOMSO[m])$$

Amounts of sent and received data characterise not only the network load, but also the consumption of other resources of the nodes (e.g. memory, CPU time, etc.) necessary to process this traffic.

6.4 Test Phases and Test Metrics

SOA testing will span a number of test phases and test metrics. Taking into account the phases defined in previous section and the metrics defined in the current section we show in the Table 2 a summary of the test metrics that may be required in the different test phases:

	Complexity Metrics			**Criticality and Reliability Metrics**			
Test Phase	Network cohesion	Services involved	Services Interdep.	Absolut Importance	Absolut Dependency	Abs. Criticalicity	Overall Reliability
Service Testing	-	-	-	👍	👍	👍	-
Integration & Orchestration testing	-	👍	-	-	-	-	👍
System level testing	-	-	-	-	-	-	-

	Performance Metrics			
Test Phase	Input & Output Size	Messages Rates	Network Load	Response Time
Service Testing	👍	👍	👍	👍
Integration & Orchestration testing	👍	👍	👍	👍
System level testing	👍	👍	👍	👍

Table 2. Test Phases and Test Metrics correlation.

7 Tests Scenarios Definition

The discussions about the possible scenarios were initiated very early in the project, just after the component description because user requirements are indispensable to identify the functional requirements and to initiate the architecture design.

Thus, at the beginning a fictive scenario was elaborated to describe a possible session of information retrieval. Such a scenario permits to identify the main functionalities that the system should provide and the possible workflows. Based on this very general scenario, WP8 and WP9 started to specify more concretely the scenarios for the specific Use Cases targeted in the project. As the user requirements are still in progress (in parallel to the architecture for WP9, and later for WP8), the architecture is fully aligned with the current status of this task but will need some refinement after the first year based on requirements analysis and specification activities within WP8 and WP9.

The main tests scenarios have been defined according to these initial WP8 and WP9 requirements. Therefore, each scenario is associated with a main workflow and a set of components involved on it. The evaluation of the software architecture will be performed over these workflows and components. In the next tables, a description of components, scenarios and main functionalities appears for each workflow to evaluate.

Relations between all these elements can also be seen more clearly in the Sequence Diagrams attached in the

Appendix section.

7.1 Textual Search Scenario

The Textual Search workflow contains the following components, scenarios 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
	Wrapin (HON)	HON's medical multilingual speller service
	Multilingual Translator (CUNI)	CUNI Multilingual translator service
	Dissambiguator (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
	Step2 : improve query	The user obtains some improvements to the query keywords performed to the system: spelling correction, possible language translation and disambiguation.
	Step3 : perform search of final query	The query is divided for different topics and the perform the search
	Step4 : return List of Documents	The user receive a List of documents with hits found as the answer of his search
	Step5 : view document	The user can see and translate any 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:	viewDoc(docURI)

	from ezDL	
	Functionalities used by ezDL: from Wrapin	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 Dissambiguation	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,lenght)

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. For 2D Image Search we can observe the next table of components, scenarios and functionalities:

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
	GIFT (HEVS)	HEVS content-based image retrieval service
	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 queries 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	getSimilarImages(List<ImageScore>,List<ImageCollection>, userProfile)
	Functionalities used by ImageManager: from GIFT	searchBySimilarity(queryXML,List<ImageURL,score>, List<ImageCollectionLocation>)
	Functionalities used by GIFT: from GIFT	List<QueryVector, score>: createQueryVectors(List<ImageURL,score>)
	Functionalities used by GIFT: from GIFT	ModifiedQueryVector : createModifiedVector(List<QueryVector,score>)
	Functionalities used by GIFT: from Respository	getImageCollectionVectors(ImageCollectionLocation)
	Functionalities used by GIFT: from GIFT	SubList<ImageURL,score>: computeScores(ModifiedQueryVector,ImageCollectionVectors)
	Functionalities used by GIFT: from GIFT	addToList(SubList<ImageURL,score>)
	Functionalities used by GIFT: from GIFT	List<ImageURL,score>:sortList()

7.3 3D Image Search Scenario

3D Image Search workflow contains the next components, scenarios and functionalities:

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 webservice
	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	3DImageSearch(image,text)
	Functionalities used by ImageManager: from SCA-MUW	imageSearch(image,text)
	Functionalities used by SCA-MUW: from MUW	getImageId(image,text,Vector<rates>)
	Functionalities used by ezDL: from SCA-MUW	Image: getFullImage(image_thumbnail_id)
	Functionalities used by SCA-MUW: from MUW	Image: getFullImage(image_thumbnail_id)

7.4 Multilingual Textual Search Scenario

The Multilingual Textual Search will be considered as a special type of Textual Search workflow where the query text performed can be translated during the search. This workflow is composed by the next components, scenarios and functionalities:

Prototype 1	**Multilingual 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
	Wrapin (HON)	HON's medical multilingual speller service
	Multilingual Translator (CUNI)	CUNI Multilingual translator service
	Dissambiguator (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
	Step2 : improve query	The user obtains some improvements to the query keywords performed to the system : spelling correction, possible language translation and dissambiguation.
	Step3 : perform search of final query	The query is divided for different topics and the perform the search
	Step4 : return List of Documents	The user receive a List of documents with hits found as the answer of his search
	Step5 : view document	The user can see and translate any 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 Wrapin	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 Dissambiguation	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,lenght)

8 Results Report

Once the evaluation approach has been specified and the early software architecture has been implemented, the scenarios have been used to perform KHRESMOI System Testing.

The Figure 2 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.

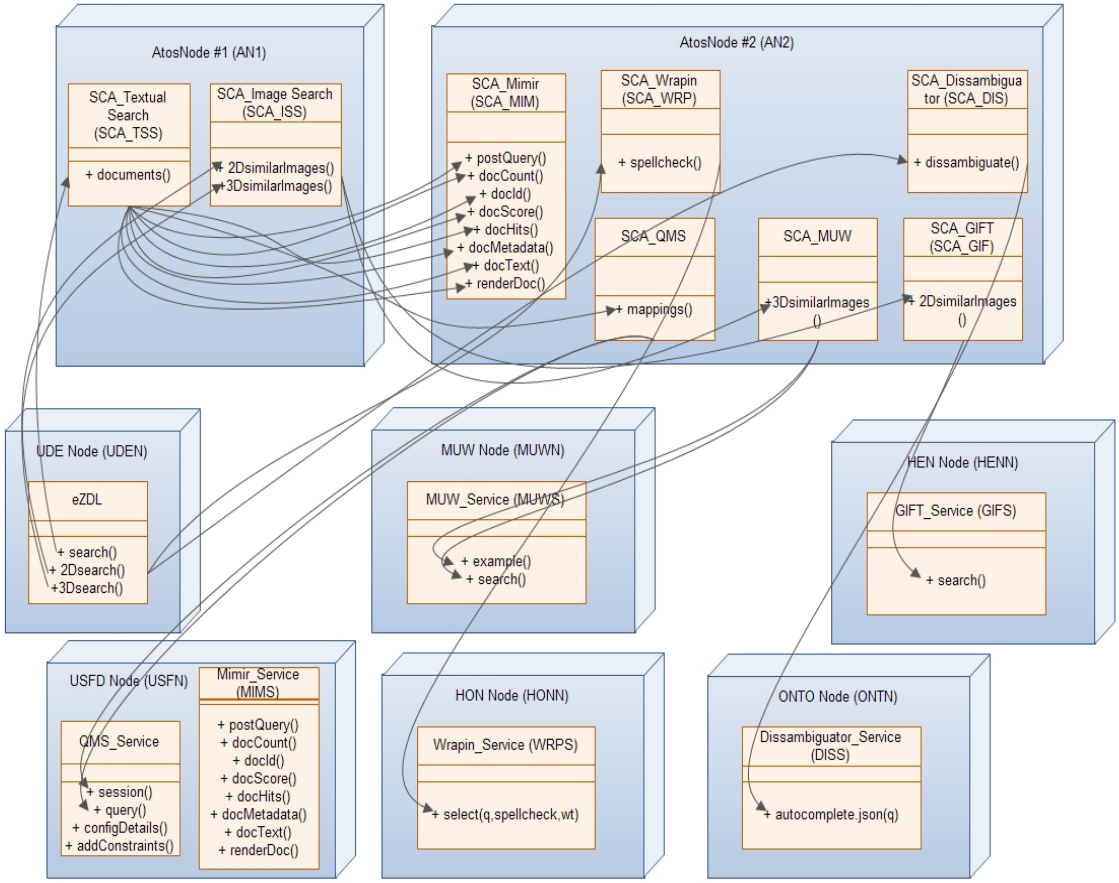


Figure 2. Khresmoi Service Oriented System.

We can define a set A as "the set of all existing calls in the system among its services". This can be calculated easily by counting the number of direct arrows that appear in the previous Figure 2. The formal description of A would be the next:

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_ISS, AN2, SCA_GIF, SCA_GIF.2DsimilarImages()>,
<AN1, SCA_TSS, AN2, SCA_MUW, SCA_MUW. 3DsimilarImages()>,
< AN2, SCA_MUW, MUWN, MUWS, MUWS.getSimilarImages()>,
< AN2, SCA_GIF, HENN, GIFS, GIFS.getSimilarImages()>,
< AN2, SCA_WRP, HONN, WRPS, WRPS.select()>,
< 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_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_WRP, SCA_WRP.spellCheck()>,
< UDEN, EZDL, AN2, SCA_DIS, SCA_DIS.dissambiguate ()>}

8.1 Scenario Independent Results

This section aims to define some scenario independent metrics that are useful to assess the system architecture design as a Service Oriented Infrastructure. The main concepts that will be taken into account during the assessment process are those related with network cohesion, services dependency, criticality, reliability, etc. Therefore, these metrics are defined to measure both quantitative and qualitative aspects.

8.1.1 Network Cohesion

Network cohesion can be defined as the degree of unity between nodes of the system. This metric is represented as the sum of all direct links between nodes. This value can be calculated as follows:

$NC = |\pi_{InvokerNode,ServiceNode}(\sigma_{InvokerNode \neq ServiceNode}(A))| = \{<AN1, AN2>, <AN2, MUWN>, <AN2,GIFN>, <AN2, HONN>, <AN2, USFN>, <AN2, ONTN>, <UDEN, AN1>, <UDEN, AN2>\} = \mathbf{8}$

On our case, the KHRESMOI system shows a good network cohesion value that ensures a good couple degree of the full system.

8.1.2 Number of Services Involved in a Compound Service

This metric is the simplest complexity indicator of a compound service. This metric is focused on showing the different services involved in the main composed services in charge of system workflows, as exposed in Figure 2. In the KHRESMOI Systems, the services are SCA_TSS,

SCA_ISS and eZDL and the metric value for these cases is calculated as a simple sum of the number of services associated:

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

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

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

$NSIC[ezDL] = |S^R[EZDL]| = \{SCA\_TSS, SCA\_ISS, SCA\_QMS, SCA\_MIM, SCA\_MUW, SCA\_GIF, MUWS, GIFS, SCA\_WRP, WRPS, SCA\_DIS, DISS\} = \mathbf{11}$

The metric results above shows an equal value for SCA_TSS and SCA_ISS that are in charge of managing the logic of the system whereas that eZDL show 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.

8.1.3 Services Interdependence in the System

This metric is formally defined as the pairs of services dependent on each other. The goal of this metric is to show any possible error in service designing in terms of incoherent interdependence between services.

$$SIY = |\beta_{Service1 \leftarrow Invoker,Service2 \leftarrow Service}(\pi_{Invoker,Service}(A)) \cap \beta_{Service2 \leftarrow Invoker,Service1 \leftarrow Service}(\pi_{Invoker,Service}(A))|.$$

There is no service interdependence in the KHRESMOI system.

8.1.4 Absolute Importance of a Service

The Absolute Importance of a Service is defined as the number of external services which call directly to its methods or operations. In this context, the clients are recognized only as those out of the node of the service so they are useful for measuring the network traffic in terms of performance. Calculating this value for KHRESMOI services we get the following results:

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

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

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

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

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

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

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

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

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

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

$AIS[DISS] = \mathbf{1}$

$AIS[WRPS] = \mathbf{1}$

$AIS[QMSS] = \mathbf{1}$

$AIS[MIMS] = \mathbf{1}$

$AIS[MUWS] = \mathbf{1}$

$AIS[GIFS] = \mathbf{1}$

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 feature 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 can be defined as the opposite to previous one, so this shows the level of dependence to other services. In this case, this value can give us an idea about the autonomy of the services and possibility of reuse as stand-alone instances, etc. Therefore, autonomy and reusability are affected for this value.

The results obtained above show a higher ADS value for eZDL so we can suppose it as the less autonomous service despite of DISS, WRPS, etc. that are completely autonomous in our system.

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

$ADS[ezDL]=\mathbf{4}$

$ADS[SCA\_TSS]=\mathbf{2}$

$ADS[SCA\_ISS]=\mathbf{2}$

$ADS[SCA\_WRP]=\mathbf{1}$

$ADS[SCA\_DIS]=\mathbf{1}$

$ADS[SCA\_QMS]=\mathbf{1}$

$ADS[SCA\_MIM]=\mathbf{1}$

$ADS[SCA\_MUW]=\mathbf{1}$

$ADS[SCA\_GIF]=\mathbf{1}$

$ADS[DISS]=\mathbf{0}$

$ADS[WRPS]=\mathbf{0}$

$ADS[QMSS]=\mathbf{0}$

$ADS[MIMS]=\mathbf{0}$

$ADS[MUWS]=\mathbf{0}$

$ADS[GIFS]=\mathbf{0}$

8.1.6 Absolute Criticality of a Service

This metric is calculated as the product of two previous metrics: AIS and ADS. Therefore, its value represents the level of importance that should be paid over this service during service-oriented system design because a positive value shows the service as critical. This is based on the idea of that a service with both metrics positive are more important that one with high ADS but poor AIS.

$ACS[s] = AIS[s] \times ADS[s].$

$ACS[ezDL] = AIS[ezDL] \times ADS[ezDL]=\mathbf{0}$

$ACS[SCA\_TSS] = AIS[SCA\_TSS] \times ADS[SCA\_TSS]=\mathbf{2}$

$ACS[SCA\_ISS] = AIS[SCA\_ISS] \times ADS[SCA\_ISS]=\mathbf{2}$

$ACS[SCA\_WRP] = AIS[SCA\_WRP] \times ADS[SCA\_WRP]=\mathbf{1}$

$ACS[SCA\_DIS] = AIS[SCA\_DIS] \times ADS[SCA\_DIS]=\mathbf{1}$

$ACS[SCA\_QMS] = AIS[SCA\_QMS] \times ADS[SCA\_QMS]=\mathbf{1}$

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

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

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

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

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

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

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

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

*ACS[WRPS] = AIS[GIFS] × ADS[GIFS]=**0***

In other words, we can represent this relation as a table where columns represent criticality related to the previous metrics:

Metrics	**Criticity**		
	Very critical (>1)	Critical(=1)	Low criticity(=0)
ACS[ezDL]			×
ACS[SCA_TSS]	×		
ACS[SCA_ISS]	×		
ACS[SCA_WRP]		×	
ACS[SCA_DIS]		×	
ACS[SCA_QMS]		×	
ACS[SCA_MIM]		×	
ACS[SCA_MUW]		×	
ACS[SCA_GIF]		×	
ACS[DISS]			×
ACS[WRPS]			×
ACS[QMSS]			×
ACS[MIMS]			×
ACS[MUWS]			×
ACS[WRPS]			×

Table 3. Absolute Critically of a Service Matrix.

To sum up, according to the table, we can define as critical services:

- SCA_TSS, SCA_ISS with 2 as a value
- Also with lower value of 1: SCA_WRP, SCA_DIS, SCA_QMS, SCA_MIM, SCA_MUW, SCA_GIF.

8.1.7 Overall Reliability of a Compound Service

Overall reliability aims to measure the reliability of a compound service by looking at its weakest component in terms of reliability. The result value is calculated in base to the previous one. Therefore we can define overall reliability as follows:

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

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

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

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

8.2 Scenario Dependent Results. Performance Metrics.

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 and 3D Image 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 to the system: input/output messages size, network load, messages rate, etc.

Therefore, the next sections will describe the previous scenarios as well as the metrics involved in all of them. The metrics results are represented scalar values so we can assess the quality of them by comparison among the different scenarios.

As premise for the analysis of performance metrics we will consider that representative groups of end users are available for sizable evaluations, accessed currently through a medical search engine with 11000 queries per day, a professional association of 2700 medical doctors, and two radiology departments with 175 radiologists. Taking into account this requirement potentially we will have 7,63 queries per minute in the System.

8.2.1 Textual Search Scenario.

Figure 3 shows all the components involved in Textual Search scenario and all direct relations among them. These relations represent direct calls to component methods during workflow execution, so it allows us to measure the metrics defined within them. During Textual Search workflow it is very important to ensure a good performance of these relations as there are a lot of components involved and it is necessary to check every single component as stand-alone before texting it as a whole system.

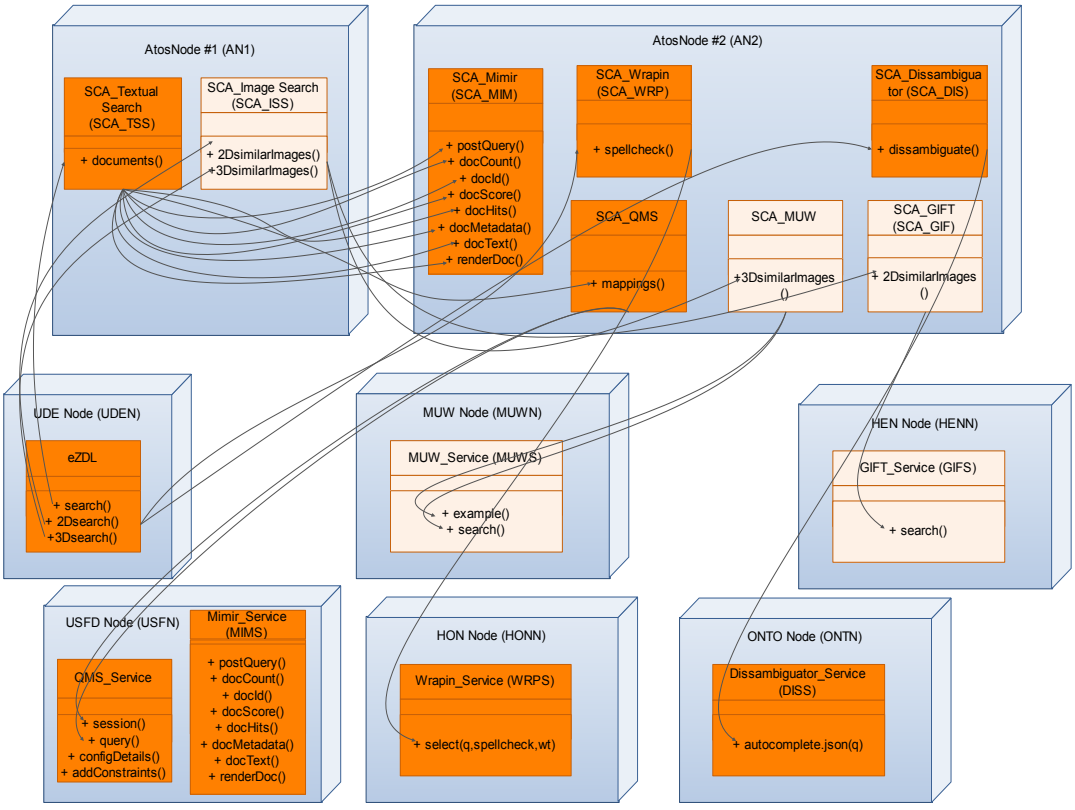


Figure 3. Textual Search Scenario in Khresmoi SOA.

Next, it is shown some metric results for different query searches performed by the Textual Search workflow.

8.2.1.1 Sizes of Input and Output Messages

This metric can be defined as critical because sending large data through the network can affect system performance in terms of response time and latency.

- AN2.spellCheck : diabetess, lacctose, cancar

AIMSO[SCA_WRP.spellCheck] = ***373 bytes***.

AOMSO[SCA_WRP.spellCheck] = ***558 bytes***.

- AN2.dissambiguate : diabetes, lactose, cancer

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

AOMSO[SCA_DIS.dissambiguate] = ***21863 bytes.***

- AN1.search : diabetes, lactose, cancer

AIMSO[SCA_TSS.search] = ***378 bytes.***

AOMSO[SCA_TSS.search] = ***19467 bytes.***

We can see above the results for some query searches: diabetes, lactose, cancer. One important thing to see is how AOMSO appears as bigger than AIMSO and although it is only twice bigger for spellcheck, it is much bigger for disambiguation. This issue must be taken into account to control output responses latency during Textual Search workflow.

8.2.1.2 Messages Rates

The importance of this metric is that a larger number of messages may affect the system performance. Next, we can see some examples related to some Textual Search queries:

- AN2.spellCheck : diabetess, lacctose, cancar

IMRN[AN2] ≈ *OMRN[AN2]*= *391 mes./min*

MRN[AN2] = *391* + *391* = ***782 mes./min***

- AN2.dissambiguate : diabetes, lactose, cancer

IMRN[AN2] ≈ *OMRN[AN2]*= ***188 mes./min***

MRN[AN2] = *391* + *391* = ***376 mes./min***

- AN1.search : diabetes, lactose, cancer

IMRN[AN1] ≈ *OMRN[AN1]*= *6 mes./min*

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

We can observe that AN1.search has a lower MRN value than other services because it represents the absolute input/output of the system. However, AN2.spellcheck and AN2.dissambiguate have a higher MRN because these components are involved in the middle of the logic workflow. Therefore, this metric reveal us the importance of controlling IMRN and OMRN of components involved in workflow as well as IMRN and OMRN of User Interface Component.

8.2.1.3 Network Load

This metric attempts to demonstrate the viability of the system operation through the network and to identify the bandwidth required for the correct performance of the system. For this scenario, we are interested on testing the network bandwidth by launching text queries over the system. Below, we can observe some metric results:

ITN[AN1] = N × (378+19467) = ***7.63 calls/min. × 19845 bytes/call = 0.0189 M/min***
OTN[AN1] = N × (19467+378) = ***7.63 calls/min. × 19845 bytes/call = 0.0189 M/min***
ITN[AN2] = N × (307+373+558+21863) = ***7.63 calls/min. × 23101 bytes/call = 0.1680 M/min***
OTN[AN2] = N × (558+21863+307+373) = ***7.63 calls/min. × 23101 bytes/call = 0.1680 M/min***

In this scenario, we can compare ITN and OTN values for AN1 and AN2, because they are the main nodes of the system for Textual Search workflow as they contain all the components and composites of the system. We can observe how AN2 has a higher traffic network consume because it contains all the SCA Components within. However, it is important to ensure also the total Traffic network value because the N calls/min can be very variable.

We are not considering here the input and output sizes associated to operation “view document” covered by the step 5 in the definition of the current scenario. Due to the current implementation status of the different services involved in this workflow it was not possible to include the bytes volume in the final value of the AN1 and AN2.

8.2.2 2D Image Search Scenario.

2D Image Search scenario represents the workflow in charge of searching images through the KHRESMOI system. As we can b see in Figure 4 bellow, this scenario contains a smaller number of components involved in the workflow. This will affectthe metric results in a quantitative way. Moreover, this scenario is supposed to be working with images in the future, although in the current system architecture we just return an id for each Image.

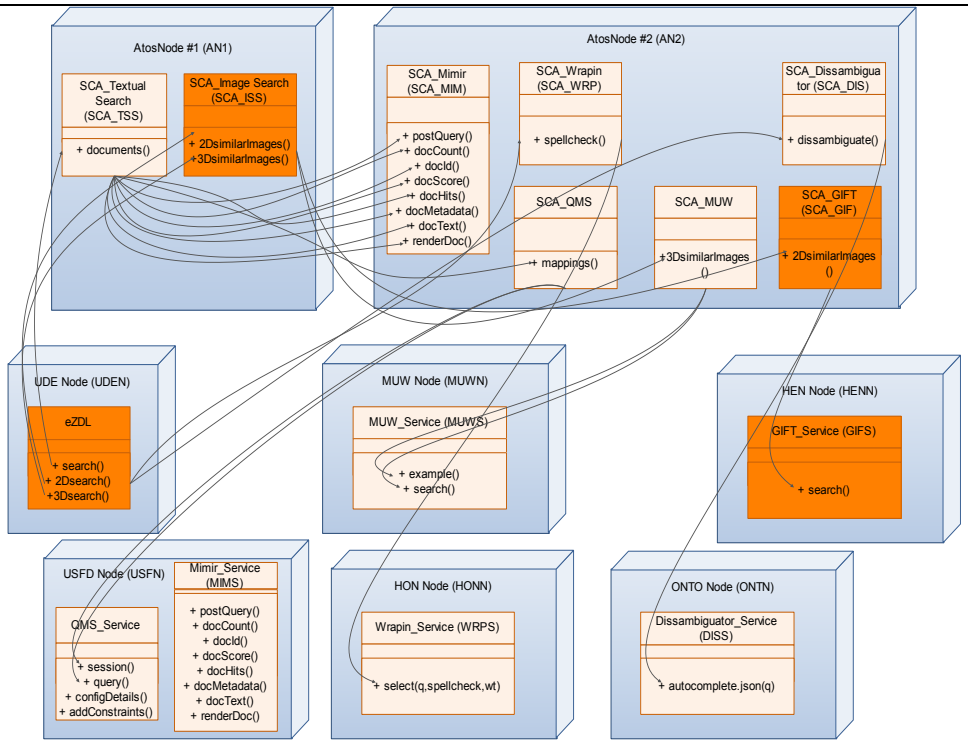


Figure 4. 2D Image Search in Khresmoi SOA.

Therefore, we will approach these topics in the next sections in order to analyse this scenario according to the metric results.

8.2.2.1 Sizes of Input and Output Messages

This metric becomes more important in this scenario and it is supposed to be bigger than that for theTextual Search scenario. This is based on the fact that any image format (binary for example) should be bigger that any text result. However, currently image format returned is not supported and the output image URLs are returned as result of the query. This fact can be checked as correct according to the results below:

- AN1.2DsimilarImages: example query

AIMSO[AN1.2DsimilarImages] = ***874 bytes.***
AOMSO[AN1.2DsimilarImages] = ***5705 bytes.***

In the future, AOMSO is due to have a higher value at the moment of returning the image so it is important to have it into account for next system architecture design iteration.

8.2.2.2 Messages Rates

We can see here the value of the input/output messages per minute of 2D Image Search which is performed by the method 2DsimilarImages. As we can see there are no dependencies to external nodes and this search is performed totally in AN1. Therefore, is it important to assess this node and service in terms of availability for next architecture design iterations.

- AN1.2DsimilarImages : example query

IMRN[AN1] ≈ *OMRN[AN1]= 241 mes./min*
MRN[AN1] = 241 + 241 = ***482 mes./min***

8.2.2.3 Network Load

This metric aims to measure the bandwidth consumption for 2D image searching for current image URLs results. Also, for future system architecture design updates, it is very important to ensure the range of sizes for images and use a proper network bandwidth that will be able to deal with these images as result.

ITN[AN1] = N × (874+5705) = ***N calls/min. × 6579 bytes/call***
OTN[AN1] = N × (5705+874) = ***N calls/min. × 6579 bytes/call***

8.2.3 3D Image Search Scenario.

This scenario is very similar to 2D Image Search Scenario. However, it is important to note that changing the result type from 2D to 3D may affect several metrics and consequently system performance. We can see below in Figure 5 the components involved in 3D Image Search workflow:

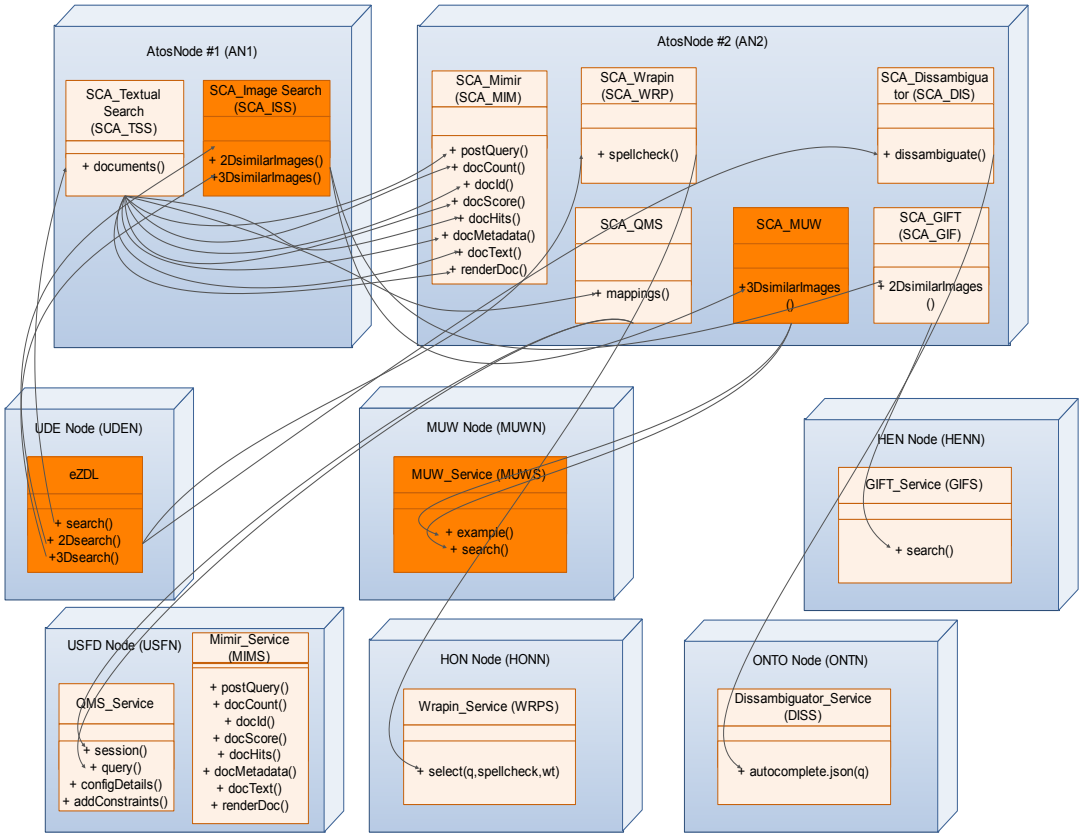


Figure 5. 3D Image Search in Khresmoi SOA.

In principle, we can suppose that the image results will be heavier than previous scenario, so the metrics results must be analysed in order to ensure a good performance of the system according to these new requirements.

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

8.2.3.1 Sizes of Input and Output Messages

Similar to the 2D Image Search, we can realise that the output size of the message is also bigger in this scenario – in fact it is even biggerfor this case. Again we regard it as a key aspect to ensure a good dealing with these outputs size in terms of latency and time responses, so the system finish the full workflow the most optimal possible.

- AN1.3DsimilarImages: example query

AIMSO[AN1.3DsimilarImages] = ***385 bytes.***
AOMSO[AN1.3DsimilarImages] = ***8800 bytes.***

8.2.3.2 Messages Rates

For MRN metrics, we have the same case here as for 2D Image Search: the total number of messages is smaller than for the Textual Search and we can see this reflected in a lower MRN value:

- AN1.3DsimilarImages : example query

IMRN[AN1] ≈ OMRN[AN1]= 168 mes./min
MRN[AN1] = 168 + 168 = ***336 mes./min***

8.2.3.3 Network Load

The Network Load of the system for 3D Image Search scenario is even more critical in this case. The large size of output image results forces us to use here a proper network bandwidth capable to work with these data without producing bottlenecks and any other performance mistakes in terms of availability or latency.

ITN[AN1] = N × (385+8800) = ***N calls/min. × 9185 bytes/call***
OTN[AN1] = N × (8800+385) = ***N calls/min. × 9185 bytes/call***

8.2.4 Multilingual Textual Search Scenario.

Due to the unavailability of the Multilingual Translator service was impossible to calculate the metrics for this scenario. In future evaluations we will take into account the requirements related to multilingual support applied for query translation and document translation “on the fly”.

9 Conclusion

In this deliverable, we present a summary of the efforts provided in task T6.2.4 related to "Evaluation of Architecture Design".

After describing the Evaluation Approach chosen for testing the KHREMSOI Architecture together with a set of metrics for each test phase included in the evaluation approach, test cases have been carried out based on the four main scenarios considered in the project.

Metrics proposed in this document are not based on any particular technology, protocol or standard and thus are applicable for various types of services and service compositions. Metrics have been categorized in three main categories which are Complexity, Critically and Reliability and Performance.

Analysis of the results let us to provide solutions to the difficulties with SOA scalability experienced during the first 15 months of the project typically raising concerns relating to:

- the time and resource required to generate the early system specification
- ambiguously defined system specifications resulting in a high occurrence of implementation errors
- the poor visibility of the overall implementation resulting in complexities in on-going system maintenance
- the inability to ensure conformance of the executing process against the originating specifications

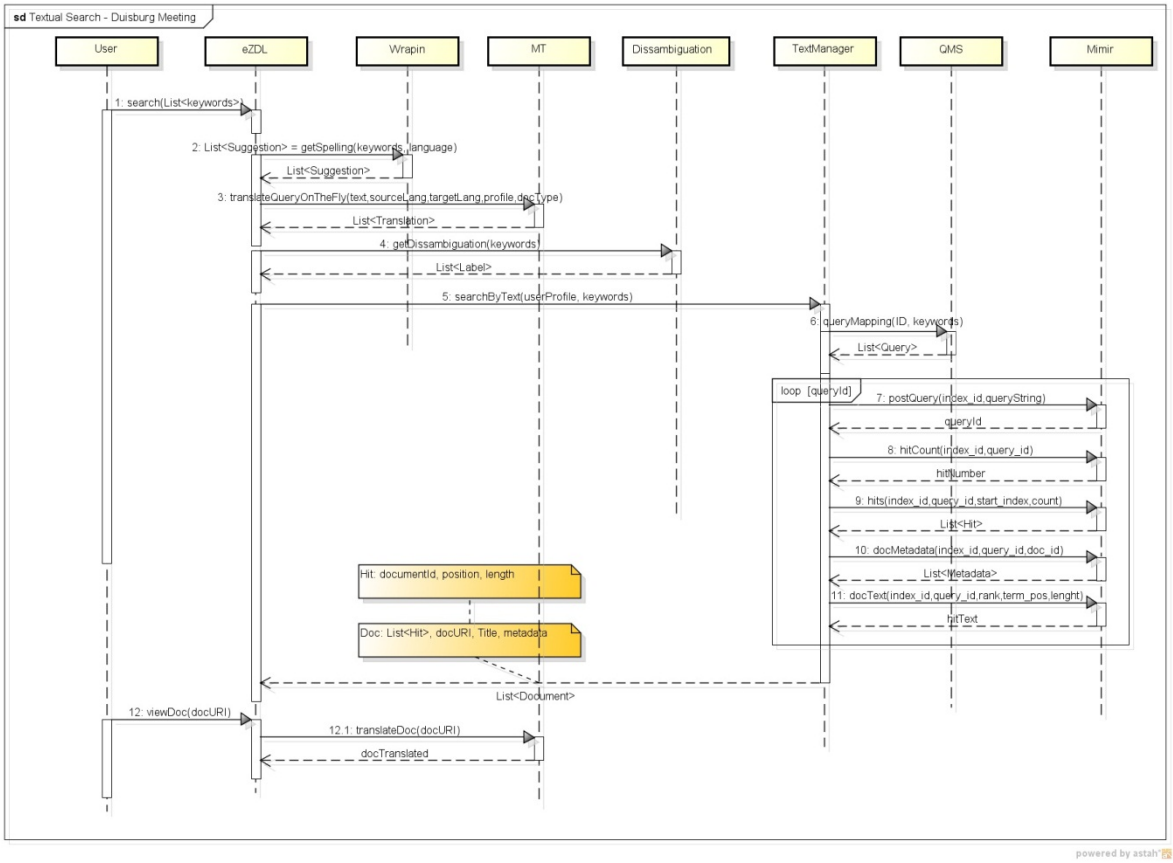
One of the main achievements at the end of the evaluation task has been the provision of overall visibility of the inter-dependencies between co-operating services to assess the impact of an future implementation change before such change is introduced assess in this way the scalability of the system in future iterations.

The results of the evaluation metrics have allowed us to establish new requirements, among which we may highlight: a) need a high bandwidth to meet the requirements of network load calculated in each of the scenarios (mainly for 2D and 3D image search scenarios), b) AN1 and AN2 nodes are critical in architecture design because they represent the main system workflows and c) a better response time or a higher message rate is required for operations node AN1 related to textual search because the current ratio is far from expected.

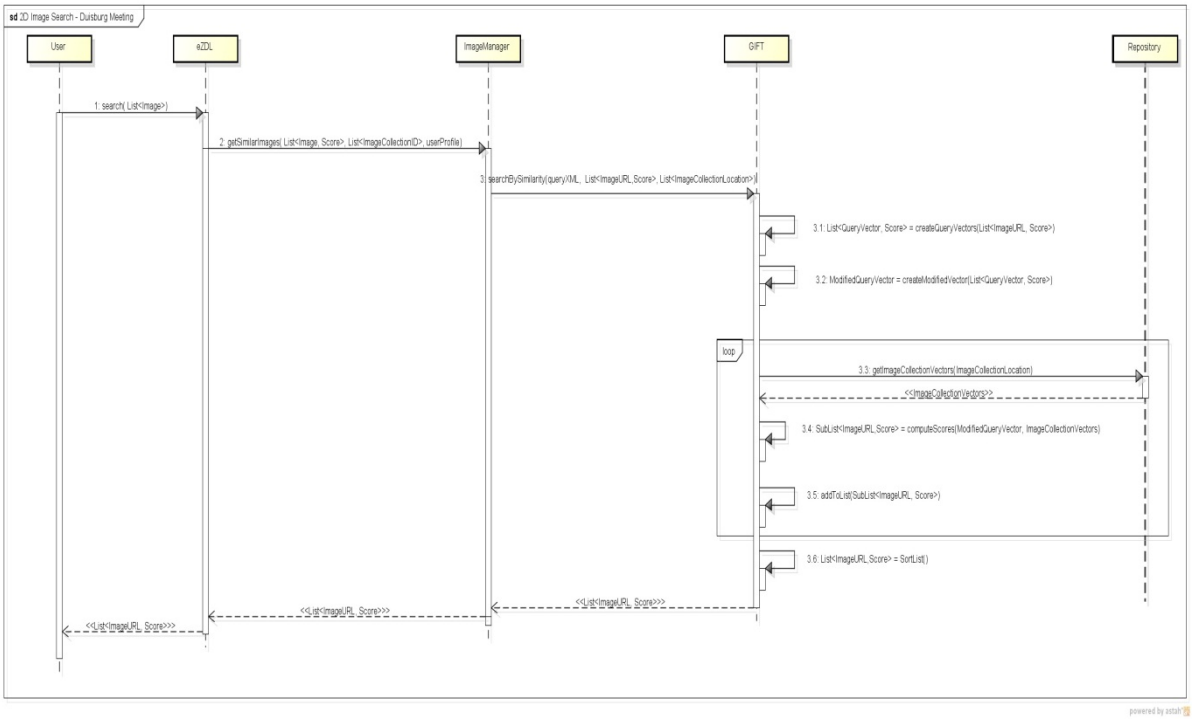
10 Appendix

WP6 Integration Sequence diagrams:

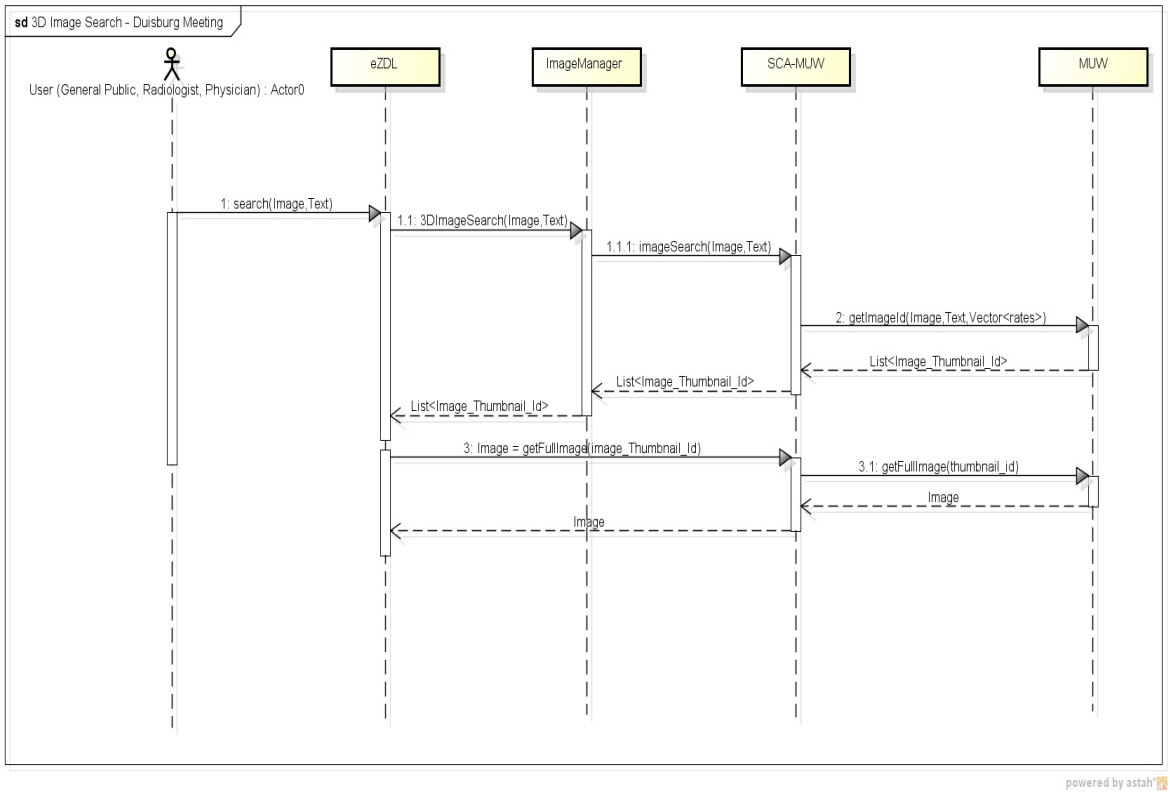
- Textual Search sequence diagram:



- 2D Image Search sequence diagram:



- 3D Image Search sequence diagram:



11 References

[1] (2007) Tony Harris Business Solutions: SOA Testing Methodology. www.thbs.com/soa.html

[2] 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

[3] D. Rud, A. Schmietendorf, and R. Dumke, "Performance modelling of WS-BPEL-based web service compositions," in Proceedings of the Modeling, Design, and Analysis for Service-oriented Architecture Workshop 2006 (mda4soa'2006), (Chicago, USA), September 2006.

[4] D. Rud, A. Schmietendorf, and R. Dumke, "Performance annotated orchestration models for service oriented architectures," in Proceedings of the 22nd Annual UK Performance Engineering Workshop 2006 (UKPEW'06), Bournemouth University, Poole, Dorset, UK, July 2006.