FP6-511723

INFRAWEBS

Intelligent Framework for Generating Open (Adaptable) Development Platforms for Web-Service Enabled Applications Using Semantic Web Technologies, Distributed Decision Support Units and Multi-Agent Systems

Specific Targeted Research Project
Priority 2 “Information Society Technologies”

Deliverable D2.2-3.2
Revised Specification and Full Model OM including the coupling and integration to the SWU / SAM / SIR

Andrei Boyanov
Galya Andonova
Todor Zaynelov
Georg Jesdinsky
Joachim Nern

Due date of deliverable: 31/07/2006
Actual submission date: 18/09/2006

Start date of the project: 01/08/2004
Duration: 30 Months

big7aspa
big7aspa

Version 9, dated 18/09/2006
EXECUTIVE SUMMARY

This deliverable gives the full model description of the general OM including the coupling to and integration with the SWU / SAM / SIR. It describes comprehensively the software realization and its functionality in interoperation with the further framework modules.

In Section 1 and 2 an introduction as well as the integration of the OM in the IIF is illustrated and exemplified.

Detailed information and description about the OM integration and functionality within the Infrawebs system context is given in Section 3.

In section 4 the OM design basis is presented, including the explication of the main OM elements, the filtering procedures, the object clustering and object classification algorithms.

The OM Interfaces as well as the OM Design details are comprehensively specified, depicted and described in the sections 5 and 6.
**DOCUMENT INFORMATION**

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP6 – 511723</th>
<th>Acronym</th>
<th>INFRAWEBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full title</td>
<td>Intelligent Framework for Generating Open (Adaptable) Development Platforms for Web-Service Enabled Applications Using Semantic Web Technologies, Distributed Decision Support Units and Multi-Agent-Systems</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://www.infrawebs.eu">http://www.infrawebs.eu</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Document URL</td>
<td>EU Project officer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dr. Anne-Marie Sassen</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>D2.2-3.2</th>
<th>Title</th>
<th>Full OM Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work package Number</td>
<td>WP2</td>
<td>Title</td>
<td>Organizational Memory (OM) and Recommender Tool (RT)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of delivery</th>
<th>Contractual 31/07/2006</th>
<th>Actual 18/09/2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Version 9, dated 18/09/2006</td>
<td>final</td>
</tr>
<tr>
<td>Nature</td>
<td>Report ☑ Demonstrator ☑ Other ☐</td>
<td></td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>Public ☑ Consortium ☐</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors (Partner)</th>
<th>Andrei Boyanov (big7aspa), Galya Andonova (big7aspa), Todor Zaynelov (big7aspa), Joachim Nern (big7aspa), Georg Jesdinsky (big7aspa)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsible Author</td>
<td>Joachim Nern Email <a href="mailto:nern@infrawebs.info">nern@infrawebs.info</a></td>
</tr>
<tr>
<td>Partner</td>
<td>big7aspa Phone +49 171 4762998</td>
</tr>
</tbody>
</table>

**Abstract (for dissemination)**

This deliverable gives the full model description of the general OM including the coupling integration with the SWU / SAM / SIR. The integration of the OM in the given IIF is illustrated and exemplified. Detailed information and description about the OM integration and functionality especially within the Infrawebs system context is given. The OM design basis is presented, including the description and specification of the main OM elements, the filtering procedures, the object clustering and object classification algorithms. Furthermore the OM Interfaces as well as the OM Design details are comprehensively specified and described.

**Keywords**
Organizational Memory, OM Design Basis, OM Interface as Web Service

**Version log/Date**

<table>
<thead>
<tr>
<th>24-Jul-06 – V1</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Structure</td>
<td>Boyanov</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>25-Jul-06 – V2</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding Screenshots</td>
<td>Zaynelov</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>27-Jul-06 – V3</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtitling of Screenshots</td>
<td>Zaynelov</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Version</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>28-Jul-06</td>
<td>V4</td>
<td>Some formatting</td>
</tr>
<tr>
<td>09-Aug-06</td>
<td>V5</td>
<td>Restructured &amp; enhanced</td>
</tr>
<tr>
<td>15-Aug-06</td>
<td>V6</td>
<td>Optimised Restructed – Several modifications – textual fragments added for discussion</td>
</tr>
<tr>
<td>22-Aug-06</td>
<td>V7</td>
<td>Detail tuning and clarification</td>
</tr>
<tr>
<td>02-Sept-06</td>
<td>V8</td>
<td>Considering the first internal review comments and adjustment to Del 5</td>
</tr>
<tr>
<td>12-09-06</td>
<td>V9</td>
<td>Considering the second internal review and final adjustments</td>
</tr>
<tr>
<td>15-09-06</td>
<td>V9</td>
<td>Review</td>
</tr>
<tr>
<td>18-09-06</td>
<td>V9</td>
<td>Correction</td>
</tr>
<tr>
<td>18-09-06</td>
<td>V9</td>
<td>Correction</td>
</tr>
</tbody>
</table>
# Table of Contents

Executive Summary .......................... 2
Document Information ........................ 3
Table of Contents ............................ 5
List of Abbreviations ......................... 8
List of Figures ............................... 9
List of Tables ................................ 11

1 Introduction ............................... 12

2 The OM in the Context of the INFRAWEBS Framework ....................... 13

3 OM Integration and Functionality in the INFRAWEBS System Context .......... 16
   3.1 OM Integration - Interoperation with other Infrawebs Modules .......... 16
      3.1.1 Interaction with the SWS-Designer ............................ 17
         3.1.1.1 Retrieve WSML ontologies ................................ 17
         3.1.1.2 Retrieve WSDL files ..................................... 17
         3.1.1.3 Retrieve WSMO service ................................... 18
      3.1.2 Interaction with the SWS-Composer ............................ 18
         3.1.2.1 Create composed WSMO based SWS ......................... 18
      3.1.3 Interaction with SWS-Goal Editor ............................ 19
      3.1.4 Interaction with the 1st step Discovery within the SAM .......... 19
      3.1.5 Interaction with DSWS-R ..................................... 19
      3.1.6 Interaction with SIR .......................................... 20
   3.2 OM Functionality – retrieving semantic and non-semantic objects ...... 21
      3.2.1 Retrieving WSMO objects by loaded (imported) ontologies ........ 21
      3.2.2 Retrieving WSMO objects by natural language descriptions ...... 21
      3.2.3 Retrieving WSMO objects by WSMO ontology terms ............... 21
      3.2.4 Retrieving WSMO objects similar to given WSMO descriptions ... 22

4 OM Design Basis ............................. 23
   4.1 Main OM elements ........................ 23
      4.1.1 Knowledge objects ................................. 23
      4.1.2 Classification tree and structure ......................... 25
      4.1.3 System dictionaries .................................. 25
      4.1.4 Centroids .......................................... 28
      4.1.5 Object Feature Vectors ............................. 28
      4.1.6 SOFV components ................................ 29
4.2 Filtering (centroids/ objects generation)
  4.2.1 Filtering of WSMO objects components
    4.2.1.1 Filtering of WSMO services and goals
    4.2.1.2 Filtering of a WSMO ontology
  4.2.2 Filtering of used (imported) ontologies
  4.2.3 Non-functional properties filtering
  4.2.4 Filtering of WSDL files
  4.2.5 Full text filtering
  4.2.6 Additional Metadata Filtering

4.3 Objects clustering

4.4 Objects classification

4.5 Similarity calculation methods
  4.5.1 Weighted intersection similarity method
  4.5.2 Weightless intersection similarity method
  4.5.3 Lexical similarity
    4.5.3.1 Object preprocessing
    4.5.3.2 Analytical description of the lexical similarity

5 OM GUIs & Interfaces
  5.1 Administration Interface
  5.2 Web Service
  5.3 Web Demo interface

6 OM Design Details
  6.1 OM Components
    6.1.1 Data Layer Component
    6.1.2 Objects Retrieval Component
    6.1.3 Classification Component
    6.1.4 Clustering Component
    6.1.5 Similarity Calculator Component
    6.1.6 Filtering Component
    6.1.7 Discovery component
    6.1.8 Web Service Component
    6.1.9 GUI Component
    6.1.10 Web Demo Component
  6.2 Components interactions

7 Work for the Next Project Period

8 References
**LIST OF ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>BPEL4WS</td>
<td>Business Process Execution Language for Web Services</td>
</tr>
<tr>
<td>CBR</td>
<td>Case Based Reasoning</td>
</tr>
<tr>
<td>CFV</td>
<td>Centroid Feature Vector</td>
</tr>
<tr>
<td>CTV</td>
<td>Centroid Term Vector</td>
</tr>
<tr>
<td>DSWS-R</td>
<td>Distributed Semantic Web Service Repository</td>
</tr>
<tr>
<td>IRI</td>
<td>Internationalized Resource Identifiers</td>
</tr>
<tr>
<td>KO</td>
<td>Knowledge Object</td>
</tr>
<tr>
<td>MD</td>
<td>Meta Data</td>
</tr>
<tr>
<td>NFP</td>
<td>Non Functional Properties</td>
</tr>
<tr>
<td>OM</td>
<td>Organizational Memory</td>
</tr>
<tr>
<td>OTV</td>
<td>Object Term Vector</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>SAM</td>
<td>Service Access Middleware (INFRAWEBS module)</td>
</tr>
<tr>
<td>SFS</td>
<td>STREAM Flows! System (INFRAWEBS use case)</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SOFV</td>
<td>Structured Object Feature Vector</td>
</tr>
<tr>
<td>SIR</td>
<td>Semantic Information Router</td>
</tr>
<tr>
<td>STDV</td>
<td>Structured Task Description Vector</td>
</tr>
<tr>
<td>SCT</td>
<td>Subject Classification Tree</td>
</tr>
<tr>
<td>SWU</td>
<td>Semantic Web Service Unit</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Service</td>
</tr>
<tr>
<td>SWS-D</td>
<td>Semantic Web Service Designer</td>
</tr>
<tr>
<td>SWS-C</td>
<td>Semantic Web Service Composer</td>
</tr>
<tr>
<td>SWS-E</td>
<td>Semantic Web Service Executor (INFRAWEBS module)</td>
</tr>
<tr>
<td>TFIDF</td>
<td>Term Frequency Inverse Document Frequency</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>WS</td>
<td>Web Service</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSML</td>
<td>Web Service Modeling Language</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1  Design Time Phase of the Infrawebs System ......................................................... 13
Figure 2  Runtime phase of the Infrawebs System ................................................................. 14
Figure 3  Example for an Object Feature Vector (OFV) .......................................................... 25
Figure 4  OM Taxonomy composed of classification subjects ............................................... 26
Figure 5  System Dictionaries & Recommendation Function ..................................................... 27
Figure 6  Example for the Centroid representing the Subject “booking” and the layer “Natural Language” ................................................................. 28
Figure 7  Object Feature Vector with different weighted components ....................................... 29
Figure 8  Structured Object Feature Vector ............................................................................ 31
Figure 9  Resulting Object Feature Vector after Filtering of a WSMO Object Component ........... 33
Figure 10 Object Feature Vector, Generated by filtering of a WSMO ontology ............................ 35
Figure 11 Clipping of an Object Feature Vector of Filtered Imported Ontologies ....................... 36
Figure 12 Object Feature Vector, received from non-functional properties filtering .................. 36
Figure 13 Classifying by matching feature vector profiles and centroids ..................................... 39
Figure 14 Weighted intersection similarity method .................................................................... 40
Figure 15 Weightless intersection similarity method ................................................................. 41
Figure 16 Analytical description of the lexical similarity ............................................................ 43
Figure 17 Main form of the Administration Module ..................................................................... 44
Figure 18 Subject Classification Tree - tree and flat view ........................................................... 45
Figure 19 Edit Subject Form by selecting subject from subjects tree and "Edit Subject“ from context menu or from main menu "Subjects tree/ Edit selected Subject" ..................... 45
Figure 20 Show classified objects – select subject and "Subjects Tree/ Show classified objects” from main menu, or select subject and "Show classified objects” from context menu on subject’s tree ................................................................. 46
Figure 21 Searching for Objects ............................................................................................... 47
Figure 22 List and Select Objects .............................................................................................. 48
Figure 23 Insert Object .............................................................................................................. 48
Figure 24 Editing objects – by selecting object from objects panel with "Edit” from the context menu or from main menu "Objects/ Edit". ................................................................. 49
Figure 25 Add Multiple Objects .................................................................................................. 49
Figure 26 “Show classification" from context menu on selected objects in objects panel or from main menu “Objects/ Show classification” ................................................................. 50
Figure 27 Matrix Form: Shows similarity between objects ......................................................... 51
Figure 28 Keyword Editing form: "Keywords/ Edit Keywords" menu (KD terms, GE keywords, Abbreviations, Non-info words, Propositions. ................................................................. 51
Figure 29 Form for filtering of selected objects - "Objects/ Filter" from main menu or "Filter" from context menu in objects panel ................................................................. 52
Figure 30 "Centroids/ Propose centroids" .................................................................................... 53
Figure 31 Maintenance Menue for Centroids ............................................................................. 54
Figure 32  Centroid Form: add new centroids from main menu "Centroid/ New" or from context menu on subjects tree, right button click "Edit centroid" for the selected subject.

Figure 33  Show centroids

Figure 34  Show classified objects to centroid

Figure 35  Modified Objects Queue

Figure 36  NFP parse settings

Figure 37  Queue processing settings

Figure 38  OM Web Demo Interface

Figure 39  OM Web Demo Operations

Figure 40  List of retrieved objects, similarity coefficients and annotations

Figure 41  OM Web Demo interface for the operation “New Or Modified Objects Message”

Figure 42  OM Web Demo interface for the operation “Search for objects by query”

Figure 43  OM Web Demo interface for “Search for Similar Objects”

Figure 44  OM Web Demo interface for the operation “Search for ontologies by WSDL file”

Figure 45  OM Web Demo interface for the operation “Search for WSDL file by keywords”

Figure 46  Record and table configuration

Figure 47  MySQL Data-Layer

Figure 48  Class for retrieving Objects out of the DSWS-R

Figure 49  Implementation class for Meta-Classification (Similarity Calculator)

Figure 50  Implementation class for clustering

Figure 51  Implementation Classes for the Similarity Calculator

Figure 52  Classes for the Filtering Component

Figure 53  Discovery Component

Figure 54  Web Service Component

Figure 55  Interaction of the GUI-Component

Figure 56  Web Demo Component

Figure 57  OM Components Internal Interactions
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Search for WSMO objects operation</td>
<td>58</td>
</tr>
<tr>
<td>Table 2</td>
<td>Search for similar objects operation</td>
<td>59</td>
</tr>
<tr>
<td>Table 3</td>
<td>Search for WSDL objects operation</td>
<td>59</td>
</tr>
<tr>
<td>Table 4</td>
<td>New or modified objects operation</td>
<td>60</td>
</tr>
<tr>
<td>Table 5</td>
<td>Is Alive operation</td>
<td>61</td>
</tr>
<tr>
<td>Table 6</td>
<td>Start operation</td>
<td>61</td>
</tr>
<tr>
<td>Table 7</td>
<td>Shut down operation</td>
<td>61</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

The OM module represents a similarity based organizational memory and case based recommender tool. The component classifies the system knowledge objects using statistical and linguistic algorithms such as fuzzy matching functions.

The OM is presented in this deliverable as some kind of “knowledge object indexer”, an entity responsible for providing an indexing of objects to be used for efficient retrieval of such objects. It acts especially for the CBR components of the Semantic Web Service Designer and Composer as a case base memory.

The OM module is fully integrated in the framework and interacts with several framework components. Therefore several algorithms and functionalities are described in detail in the related documents and deliverables – especially the CBR approaches given in WP5 – several references are given inline. In the deliverable related to WP5 as well as the previous deliverable of WP2 the analytically details related to the CBR approach are discussed, whereas in this current deliverable the software realisation approach aspects are mainly pointed out.
2 THE OM IN THE CONTEXT OF THE INFRAWEBS FRAMEWORK

Before describing the full OM model in detail, an overview is given about the Infrawebs Integrated Framework and the chosen models depicting the different aspects of Semantic Web Services.

The Infrawebs framework defines two phases: Design Time and Runtime. The components involved in the Design Time phase as well as the Runtime phase are characterized in the following two illustrations. A detailed description is given in the public deliverables of WP 11 [1] and the WP10, which describes the SFS (Stream Flow System – the Infrawebs Use Case).

Figure 1 Design Time Phase of the Infrawebs System

The components involved in the Design Time phase are the following:

- SWS-D (Semantic Web Service Designer)
- Responsible for the design of semantic web service descriptions (in particular the capabilities of the Web Service descriptions)
  - SWS-C (Semantic Web Service Composer)
    - Responsible for the composition of existing WSMO-based semantic web services
  - DSWS-R (Distributed Semantic Web Service Repository)
    - Responsible for the persistent storage of WSMO-based descriptions
  - SIR (Semantic Information Router)
    - Responsible for handling the formal descriptions (based on RDF) that are submitted in the service registration process.
  - OM (Organizational Memory)
    - Represents an organizational memory structured as case base memory with recommender facilities

Figure 2  Runtime phase of the Infrawebs System
The components involved in the Runtime phase are shortly described below:\(^1\):

- **SAM (Service Access Middleware)**
  - Responsible for guiding the user applications through the steps of semantic web service usage, including service discovery, selection and execution.

- **SWS-E (Semantic Web Service Executor)**
  - Responsible for executing the semantic web service descriptions

- **QoS-Monitor (Quality of Service Monitor)**
  - Responsible for collecting monitor data and calculate the metric values for the semantic web services being executed

- **DSWS-R**
  - Same functionality as for design time

- **OM**
  - used during the 1\(^{st}\) step Discovery within the SAM

As clearly shown above, the OM component is concerned with both - the runtime phase and the design time phase. It represents a case-based memory and its main role is to index all available textual documents with respect to description of web services, both non-semantic (WSDL) and semantic (WSML).

\(^1\) Note that some of the components are also active in the design time phase.
3 **OM Integration and Functionality in the INFRAWEBS System Context**

The OM module is devoted to the organization of all non-semantic as well as specific semantic (knowledge) objects (WSMO) handled within the SWS development and life cycle. It supports the creators of Semantic Web Services to retrieve relevant or adequate information for the semantic modeling of web services. By indexing all available textual documents and providing the artefacts to the relevant module achieve this.

WSMO objects represent semantic web services, goals, ontologies and mediators written in WSML language. The WSMO community gives a full overview of the structure of WSMO objects in [2].

The OM functionality is extended to parse all main components of WSMO objects (part of the “Web Service capabilities”).

These components are:

- Imported (loaded) ontologies
- Non-functional properties
- Assumptions
- Preconditions
- Postconditions
- Effects

The OM module is directly coupled and integrated in the SWU (Semantic Web Service Unit) and serves to the following SWU modules:

- SWS-Designer
- SWS-Composer
- SWS- Goal Editor

as well as to the discovery module:

- 1st step Discovery within the SAM (Service Access Middleware)

The OM module also communicates (interacts) with the following modules mainly by indexing their objects:

- DSWS-R (Distributed Semantic Web Service Repository)
- SIR (Semantic Information Router)

3.1 **OM Integration - Interoperation with other Infrawebs Modules**

In this chapter the OM interoperation with the other Infrawebs modules is described.
3.1.1 Interaction with the SWS-Designer

In the following the interoperation of the OM and the service designer is described, especially the retrieval of WSML ontologies, of WSDL files and WSMO services.

3.1.1.1 Retrieve WSML ontologies

The SWS-Designer accesses the OM module as a case base memory for retrieving already created (existing) WSMO objects in design time. These objects are used as a basis for the creation of new “similar” WSMO objects [1].

The capability of the activity “Find WSML ontologies” (see [3]) is to provide the user with a set of WSML ontologies needed for designing a semantic Web service. The activity is implemented as a sequential process of describing the user’s demands

- by filling the request form [3],
- transforming this description into a XML query,
- sending the query for processing to the OM,
- visualising and processing the query results and
- importing the retrieved ontologies.

For the execution of this task the OM provides the following method:

Operation **SearchForWSMOObjects** [query: Query]: SearchResult

For related details see section 5.2 and table “Search for WSMO objects operation”. Details about this activity and the interaction diagram see [6] and appendix 2.

3.1.1.2 Retrieve WSDL files

The SWS-Designer uses the OM module as a case based memory for retrieving a similar WSDL file in order to annotate it semantically.

This activity aims at providing the user with a WSDL file description of a Web service to be converted by the SWS Designer into WSMO semantic Web service. The activity is implemented as a sequential process of describing the user demands (filling the request form), transforming this description into a XML query, sending the query for processing to the OM, visualising and processing the query results, selecting the desired service and importing its WSDL description and some non-functional properties

For the execution of this activity the OM offers the following method:

Operation **SearchForWSDLObject** [query: WsdlQuery]: SearchResult
For further details about this operation see 5.2 and table “Search for WSDL object operation”. The activity itself is specified in detail in [6] resp. illustrated in the appendix 2.

### 3.1.1.3 Retrieve WSMO service

The SWS-Designer uses the OM module as a case based memory for retrieving similar WSMO services with the designing process of SWS.

The “Find Similar WSMO Service” activity is accomplished in the following way:

- the SWS-Designer sends a query to the OM
- the OM sends as a result a set of IRIs identifying the matching semantic services plus annotations
- The end user (of the SWS-Designer) selects one or more of the matching documents and retrieves them out of corresponding storage facility (DSWS-R).

For the execution of this task the OM provides the following method:

**Operation** `SearchForWSMOObjects[query: Query]: SearchResult`

See 5.2 and table “Search for WSMO objects operation” and [6] resp. appendix 2 for further details.

### 3.1.2 Interaction with the SWS-Composer

#### 3.1.2.1 Create composed WSMO based SWS

The SWS-Composer uses the OM module as a case base memory for retrieving already created (existing) WSMO objects in design time, especially during the creation of composed WSMO based Semantic Web Services (Components involved: SWS-D, OM, and DSWS-R, see appendix 2). Within this context the following subactivities occur:

- Search for services with similar set of WSML ontologies
- Search for similar WSMO services (by example)
- Search for services by textual description
- Storing of constructed composed WSMO service

For the execution of this task the OM provides the following method:

**Operation** `SearchForSimilarObjects[obj: xsd:string, objectType: ObjectType, maxResult: xsd:int]: SearchResult`

See 5.2 and table “Search for WSMO objects operation”, [6] and appendix 2 for further details.
3.1.3 Interaction with SWS-Goal Editor

The SWS-Goal Editor\(^2\) [1] is going to use the OM module as a case based memory for retrieving already created (existing) WSMO objects in design time. For the execution of this task OM provides following method:

Operation **SearchForWSMOObjects**(query: Query): SearchResult

Details about this operation are given in 5.2 and table “Search for WSMO objects operation”. The activity and interaction diagram and related descriptions are given in [6] resp. appendix 2.

3.1.4 Interaction with the 1st step Discovery within the SAM

The discovery module uses the OM in run time for executing the first step phase of the INFRAWEBS discovery process. The pre-filtering step of the SWS discovery is implemented with the following steps:

- a pre-filtering query is formulated based on the current WSML goal. The WSMO goal is used as query by example [1]
- the query is sent to the OM.
- the OM responds with a list of web services, which is taken as the initial set of services in logic-based matching.

For the execution of this task OM offers following method:

Operation **SearchForWSMOObjects**(query: Query): SearchResult

For further details about this operation see 5.2 and table “Search for WSMO objects operation”.

The OM was tested with the test data provided by the partner SZTAKI in the discovery testbed [4]. All service capabilities are searchable through the local OM. The peer-to-peer networks of the SWUs are “transparent” for the SAM, and it needs to address only the local OM module for textual queries in the whole network [4].

For more details about this activity and the interaction diagram of it see [6] resp. appendix 2.

3.1.5 Interaction with DSWS-R

The OM module indexes all stored descriptions located in the repository as well as all advertisements located in the registry of the DSWS-R [7]. This is achieved by sending a notification to the OM in the case new descriptions are stored in DSWS-R.

---

\(^2\) At present the goal editor is just specified. According to the consortium’s decision in WP5 it will be realised in the next project phase.
The notification message is defined as:

```xml
<message type = "NewOrModifiedObjects" sender = "IRI of the Local Repository" senderType="WSMORepository">
  <object type="WSMOService" status = "new" identifier = "IRI of the service" />
</message>
```

The notification message sent to the OM only specifies the WSMO object identifier, and then at some predefined interval (depending on the OM configuration) the OM retrieves new descriptions (as WSML files) from DSWS-R so that it can index them offline.

Notifications for new advertisements published in the Registry are also sent to the OM:

```xml
<message type = "NewAdvertisement" sender = "IRI of the Local Registry" senderType "WSMORegistry">
  <object type "WSMOService" status = "new" identifier = "IRI of the service advertisement" />
</message>
```

The OM indexes WSMO objects created in a local SWU and stored in the Local Repository. Hereby the following Components are involved: SWS-Designer or SWS-Composer (Object creator), DSWS-R, OM. The specific tasks are:

- An Object Creator sends a newly created WSMO object to the Local Repository.
- The Local Repository stores the new WSMO object and sends a notification message to the OM.
- The OM creates indexes according to its specific indexing policy

The OM will index the advertisements published in the registry in a similar way like indexing WSMO objects created in a local SWU and stored in the local Repository.

The OM subsequently retrieves the advertisement from the Registry in order to index it via an asynchronous request to the Registry.

For the execution of the upper tasks the OM offers the following method:

**Operation** `NewOrModifiedObjects` [message: Message]: `xsd:boolean`

For details about this operation see 5.2 and table “New or modified objects operation” and [6] as well as appendix 2.

### 3.1.6 Interaction with SIR

The OM module will also index the content and metadata of WSDL files stored in the SIR module [6] and supports other modules to access them in design time.

The interaction with the SIR is characterized by the following sequence:
Indexing of WSDL files – OM

Messages exchanges (via SOAP) between SIR and OM is realised as follows:

- from SIR to OM
  - asynchronous : NewOrModifiedObjects[message]
- from OM to SIR
  - HTTP GET request (IRI) for retrieving WSDL file
  - SPARQL query for retrieving WSDL file metadata (MD)

For the execution of this task the OM offers following method:

Operation **NewOrModifiedObjects[message: Message]: xsd:boolean**

For accessing the metadata SPARQL is used [6]. For further details about this operation see 5.2 and table “New or modified objects operation” and [6] resp. appendix 2.

### 3.2 OM Functionality – retrieving semantic and non-semantic objects

#### 3.2.1 Retrieving WSMO objects by loaded (imported) ontologies

This kind of use of the OM module is a part of the activities for designing new WSMO objects. Furthermore this type of search is used as the first step of retrieving appropriate objects in case of lacking sufficient search criteria. This is the simplest type of queries, which asks for retrieving existing semantic services described by the same (or almost the same) set of ontologies as ontologies imported by the service under creation. The Designer, the Composer and the Goal Editor use this OM functionality.

#### 3.2.2 Retrieving WSMO objects by natural language descriptions

In many cases the easiest way for searching is the natural language description of the given problem. Within the SWU the Designer, the Composer and the Goal Editor use the OM functionalities for retrieving WSMO objects by natural language descriptions. These queries are devoted to retrieve existing semantic services, containing natural language terms in their NFP properties (e.g. title, publisher, description, etc.).

#### 3.2.3 Retrieving WSMO objects by WSMO ontology terms

Within the designed case based tools of the SWU (the Designer, the Composer and the Goal Editor, see Deliverable WP5) the user is enabled to retrieve WSMO objects by searching for selected ontology terms. Two types of queries by ontology terms are designed:

- a structured and
- an unstructured one.
The unstructured queries are used for retrieving WSMO objects similar to ontology terms based on descriptions without regarding the location of the term occurrence.

From the other side the structured queries permit the search for WSMO objects by different criteria – for example searching for semantic web services with preconditions similar to given ontology terms and effects similar to other ontology terms. The analytical query formulation and the use of such ontology-based querying are described in detail in [1].

3.2.4 Retrieving WSMO objects similar to given WSMO descriptions

The case based tools in [3] enable the retrieval of WSMO objects by querying for entire parts of other WSMO objects. These types of queries are called “queries by example”. The different criteria of the structured query could also be given as examples. This type of query is based on the assumption that services containing similar axioms in specific description sections (e.g. preconditions) will probably have the similar axioms in its other parts (e.g. postconditions).

After the completion of an axiom and the assignment to a given role (section, according to the WSMO framework), a querying of the CB memory results in the retrieval of similar axioms assigned to the same roles specified in the query.
4 OM DESIGN BASIS

In this chapter the main OM elements are defined and described as well as the main procedures for filtering different knowledge objects. Apart from the textual description screenshots from the running software package are given as additional illustrations.

4.1 Main OM elements

In this section the OM elements are defined and described, like knowledge objects, the classification tree, the system dictionaries, the Centroids, the object feature vector, the structured feature vector and its (SOFV-) vector.

4.1.1 Knowledge objects

A “knowledge object” (KO) in the context of the OM module is a defined entity, that represents the information contained in one of the types of WSMO objects – web service, ontology, mediator (not used in this project stage), goal as well as the information embedded in WSDL files (object). These knowledge objects represent the system (domain) knowledge.

These knowledge objects are divided into two main groups:

- semantic and
- non-semantic knowledge objects.

Semantic knowledge objects – also called “WSMO objects” - are WSML files containing semantic web services descriptions (capabilities), ontologies, goals etc. according to the WSMO [2] specification.

Non-semantic system knowledge objects are WSDL files containing non-semantic web services descriptions – not defined within the WSMO specification.

For handling these Knowledge objects as machine processable objects they are represented by a “structural feature vector model”. Every knowledge object in the OM is characterized by:

- an object identifier,
- object type
- object annotation and
- object feature vector.

The Object identifier is the IRI of the object (for example “http://www.wsmo.org/ontologies/sfs/flight”).

Several types are defined:

- service
- goal
- ontology
- mediator
• WSDL
• BPEL (BPEL4WS)\(^3\)

The object annotation gives additional (meta) information about the knowledge object. In the case of WSMO objects, the object annotation consists of different annotation elements (description, title, publisher, etc.), whereas the content is extracted out of the non-functional properties section of the file, whose structure is oriented on the DC metadata scheme (see [2]).

The object annotation elements are defined and configured by the user (admin) based on a decision, which elements are to be extracted out of the non-functional properties section of the file and to be located in the object annotation. For example the annotation is formalized as the following:

```
"ANNOTATION:
  Description: Rent a car booking
  Subject: Car, tourismServiceProvider, Car
  Title: Car rental ontology"
```

In the case of WSDL files, the annotation elements are generated out of the objects metadata information, which is accessed via SPARQL out of the NFPs\(^4\) of the object.

The object feature vector represents the “concept” of the object using a vector structure. This OFV\(^5\) consists of weighted terms, which are extracted out of different hierarchical memory layers – for details see 4.1.4 and 4.1.5. An OFV example is shown in Figure 3.

These knowledge objects are treated within several processes like “filtering” (see section 4.2), “clustering” (see section 4.3) and “classification” (see 4.4) and finally indexed and stored within the hierarchical classification tree (see next section 4.1.2).

---

\(^3\) At this project stage the BPEL (resp. BPEL4WS) type is defined, applied and processable in the OM. The final indexing will be done in relation to the concept given by the SIR [5] and further use in the SWS-C in the choreography section.

\(^4\) In principal the NFPs are specific Metadata of (about) the object – see [2].

\(^5\) In the previous deliverable [10] the short OTV (Object Term Vector) was used.
4.1.2 Classification tree and structure

Knowledge objects are classified to a given pre-configured taxonomy composed out of classification subjects. Each knowledge object is classified based on a similarity calculation to the subjects of this taxonomy (see section 4.5). The specification of this type of classification tree is given in the previous public deliverable D2.1.1.

In Figure 4 an example for this classification tree realized within the software package is given.

4.1.3 System dictionaries

The OM module uses for the filtering, the clustering and the classification procedure different types of system dictionaries:

- knowledge domain dictionary,
- generic keywords dictionary,
- abbreviations dictionary,
- stop terms dictionary and
- synonyms dictionary.
These dictionaries are the basis for calculating the weight of single terms in the relevant feature vectors – see [8] and [10]. The knowledge domain dictionary contains natural language words, specific for the particular domain – e.g. mathematics, philosophy, medicine, etc. These terms are essential for the searching process and due to this they have a maximum weight (role) in the filtering process.

Words out of the “knowledge domain” dictionary, which are elements of the object feature vector as well as the centroid feature vector (see section 4.1.4), are weighted with a weight factor equal to 3. The final weight of the term is calculated out of a combination of term frequency, term frequency inverse document frequency (TFIDF) and the predefined weight in the dictionary.

The “generic keywords” dictionary contains more generic natural language words, that are not specific and representative for the given knowledge domain, but are also important for context of terms. These words are weighted with a weight factor equal to 2 used in the weight calculation process of constructing the object feature vector as well as the centroid feature vector.

---

6 The term “centroid” is defined in detail in section 4.1.4
The “abbreviations” dictionary consists of word expressions and their abbreviations. The filtering process uses this dictionary to quasi unifies the object feature vectors in this way, that they will be substituted and manipulated correctly.

The “Non-info” (resp. “Stop Terms”) Dictionary consists of natural language terms (Stop Terms), which have no concept related impact for the searching results. These words are simply extracted out of the query and have no direct impact to the filtering process.

The “synonyms dictionary” holds all the synonyms for the terms in the Knowledge Domain Dictionary and the Generic Keywords Dictionary. The Synonyms are supplied by the WordNet library [9].

The OM component maintains also a list of keywords recommendations (propositions). These recommendations (propositions) are extracted automatically out of each new object stored in the system. Accordingly each keyword proposition is processed by the Administrator and added to one of the system dictionaries (see Figure 5). Recommendations are just supported for some classification layers (see Classification layers in the following sections). In Figure 5 the OM System Dictionaries are illustrated.
4.1.4 Centroids

The centroids are the formalized representatives of each set of objects (the subjects) for the corresponding classification layer [8], [10], [14]. These centroids are structured as centroid feature vectors (CFV) composed out of the weighted and substituted terms out of the system dictionaries for the corresponding layer. Centroids are mapped to the classification subjects based on the system taxonomy and furthermore mapped to exactly one layer of the OM Classification Layers.

In the Figure 6 an example is shown for mapping the centroid for the natural language layer to the “Booking” subject from the classification tree.

4.1.5 Object Feature Vectors

Every knowledge object is represented by a structured object feature vector (SOFV). The Structured Object Feature Vector represents a hierarchic collec-
tion of one-dimensional feature vectors (OFV), whereby the one-dimensional feature vector (OFV) is structured as a list of weighted keywords.\footnote{In the previous deliverable [10] a one-dimensional OTV was specified. Due to the complexity of the WSMO context a more structured feature vector has been designed – the SOFV.}

The structure of the SOFV in the context of the Infrawebs system is represented in Figure 7 in the current section.

A process called “filtering procedure” creates (semi-) automatically object’s feature vector (see section 4.2). For every one-dimensional feature vector a different filtering algorithm is applied.

In the following illustration (Figure 7) the feature vector contains terms with different weights related to different layers: “Ontology Based” layer, “Imported ontologies” layer and “Natural language” layer.

<table>
<thead>
<tr>
<th>Term</th>
<th>Weight</th>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>offohotelRoombookingRequest</td>
<td>1</td>
<td>Ontology based</td>
</tr>
<tr>
<td>offohotelRoombookingRequest:hotelCI</td>
<td>1</td>
<td>Ontology based</td>
</tr>
<tr>
<td>offohotelRoombookingRequest:check1</td>
<td>1</td>
<td>Ontology based</td>
</tr>
<tr>
<td>offohotelRoombookingRequest:check2</td>
<td>1</td>
<td>Ontology based</td>
</tr>
<tr>
<td>offohotelRoombookingRequest:runbe1</td>
<td>1</td>
<td>Ontology based</td>
</tr>
<tr>
<td>offohotelRoombookingRequest:hotelSt</td>
<td>1</td>
<td>Ontology based</td>
</tr>
<tr>
<td>offohotelRoombookingRequest:runbe2</td>
<td>1</td>
<td>Ontology based</td>
</tr>
<tr>
<td>offohotelRoombookingRequest:smokin</td>
<td>1</td>
<td>Ontology based</td>
</tr>
<tr>
<td>df</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
<tr>
<td>loc</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
<tr>
<td>bo</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
<tr>
<td>ho</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
<tr>
<td>ticket</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>booking</td>
<td>12</td>
<td>Natural language</td>
</tr>
<tr>
<td>seller</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>aos</td>
<td>6</td>
<td>Natural language</td>
</tr>
<tr>
<td>smoking</td>
<td>3</td>
<td>Natural language</td>
</tr>
</tbody>
</table>

Figure 7  Object Feature Vector with different weighted components

4.1.6 SOFV components

As mentioned before the OM component classifies textual objects to a given classification tree. Considering the semantic feature of the objects they are classified according to different semantic criteria. This imposes the necessity of multiple classification layers, where knowledge objects are represented in each layer with different parts of their SOFV and classified to the layers centroids.
different set of system dictionaries and classification algorithms are supported for every classification layer.

The OM supports nine layers for semantic objects and two layers for non-semantic objects. The layers for non-semantic objects are (described in detail in section 4.2.4):

- “Non-semantic content” and
- “Non-semantic metadata”

According to the structure given by the WSMO specification the layers for the semantic objects are

- “Natural Language”,
- “Ontology Based”,
- “Imported Ontologies”,
- “Capabilities”,
- “Assumptions”,
- “Preconditions”,
- “Postconditions”,
- “Effects” and
- “Service Functionality”

In Figure 8 the SOTV is illustrated.

4.2 Filtering (centroids/ objects generation)

The OM component in the Infrawebs’ context supports two types of centroids:

- centroids based on natural language words;
- centroids based on ontology terms.

In this section the different filtering procedures for the different types of objects are described.

4.2.1 Filtering of WSMO objects components

The initial state of the filtering procedure of the WSMO objects is a parsing process. The Filtering of WSMO ontologies is performed in a different way than filtering of WSMO services and goal files. This distinction is due to the different structure of the WSMO web service and goal files on one hand and WSMO ontologies on the other hand.
4.2.1.1 Filtering of WSMO services and goals

The parsing method used to filter WSMO services and goals is able to distinguish the different parts of the WSMO description – assumptions, preconditions, postconditions, effects. For these separated parts ontology terms are extracted considering ontology terms containing the symbol ‘#’. By parsing the expression enclosed by the symbols ‘[’ and ‘]’ and following an ontology term (a word containing the symbol ‘#’) additionally the attribute names are extracted.

The attributes are populated in the feature vector prefixed with the ontology term found before the symbol ‘[’.
For example consider the following WSMO expression:

```owl
?FlightBooking memberOf fb#flightBooking[
    bookingIdentifier hasValue ?BookingIdentifier,
    bookingTicket hasValue ?FlightBookingTicket,
    airTripInfo hasValue ?AirTripInfo,
    buyer hasValue ?Buyer,
    seller hasValue ?IBFlightServiceProvider,
    paymentMethod hasValue ?Points
] and
```

The filtering of this expression will extract the following terms:

```plaintext
fb#flightBooking
fb#flightBooking.bookingIdentifier
fb#flightBooking.bookingTicket
fb#flightBooking.airTripInfo
fb#flightBooking.buyer
fb#flightBooking.seller
fb#flightBooking.paymentMethod
```

After the filtering process of the WSMO object component (web service or goal), the extracted terms are stored in the object feature vector. The terms are assigned to different weights, which are calculated as described before.

The extracted terms are furthermore assigned to one of the following layers: Assumptions, Preconditions, Postconditions, Effects, Service Functionality. Below is shown a part of feature vector which consists of the resulting terms of the filtered WSMO object (see illustration in Figure 9).
4.2.1.2 Filtering of a WSMO ontology

In the case of filtering a WSMO ontology the parsing method is able to retrieve concepts and to extract them. The algorithm simply detects concepts and their attributes and saves the attributes prefixes by the name of the concepts (see the example below). The parsing algorithm also creates namespace prefixes for the ontology IRI and saves these prefixes and the corresponding full names of the ontology.

For example, the ontology “http://www.wsmo.org/ontologies/sfs/shuttle” takes prefix “sh”. This prefix is created out of the first two letters after the last “/” symbol. In the case the received prefix already exists it is suffixed with the appropriate sequential number (for example 1, 2, 3).

Consider the following extracted component of an ontology file:

```java
concept shuttle

nonFunctionalProperties

  dc#description hasValue "A shuttle"
  dc#rentalRate hasValue "The shuttle rate"
  dc#hasAirConditioning hasValue "Indicates if the shuttle has air conditioning"
```

![Figure 9 Resulting Object Feature Vector after Filtering of a WSMO Object Component](image)
endNonFunctionalProperties
shuttleIdentificator ofType _string
rentalRate ofType _float
doorCount ofType _integer
shuttleSize ofType _integer
miliageLimited ofType _boolean
hasChildSeat ofType _boolean
hasAirConditioning ofType _boolean

The parsing procedure of this expression will return the following terms:

sh#shuttle
sh#shuttle.shuttleIdentificator
sh#shuttle.rentalRate
sh#shuttle.doorCount
sh#shuttle.shuttleSize
sh#shuttle.miliageLimited
sh#shuttle.hasChildSeat
sh#shuttle.hasAirConditioning

In the following figure the object feature vector corresponding to a WSMO ontology file is shown. The extracted ontology terms are stored in Ontology based layer of the object feature vector.

4.2.2 Filtering of used (imported) ontologies

The WSMO objects are parsed to extract the part containing the imported ontologies. The Ontologies are stored in the OM component with their prefixes based on a table containing the full names of the ontologies and their prefixes. In case a new (unknown) ontology is detected (an ontology for which the OM module doesn’t know a prefix) the prefix is extracted from the ‘namespace’ section of the WSMO file\(^8\). In this particular case the namespaces are actually ontologies. So if the extracted namespace prefix (ontology prefix) is already used by the OM module then it is suffixed with a sequence number.

---

\(^8\) The namespace section in the file contains namespaces and their prefixes.
During the retrieval process ontologies in the query are manipulated in a specific way that replaces them with prefixes stored in the OM. This replacement is necessary to ensure the correct work of the algorithms, which are responsible for the comparison with stored objects in order to satisfy real search user needs. If OM doesn’t support prefix for the specific ontology, this ontology is not used in the searching objects procedure.

After the filtering process the retrieved imported ontologies are stored in the object feature vector Figure 10 using their prefixes. They are presented in the feature vector with weight 1 and layer “Imported ontologies”.

In the following Figure 11 a part of an object feature vector is shown, which consists of terms received out of the filtering of used (imported) ontologies.

4.2.3 Non-functional properties filtering

Non-functional properties are extracted out of WSMO object by parsing it and concatenating all nonFunctionalProperties sections. Out of this concatenated string only terms enclosed in quotation marks (the symbol ‘”’) are used for generation of the feature vector. This string is processed in the same manner as described in 4.5.3.1. This natural language filtering uses five system dictionaries which are described in 4.1.3.
Figure 11  Clipping of an Object Feature Vector of Filtered Imported Ontologies

<table>
<thead>
<tr>
<th>Term</th>
<th>Weight</th>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
<tr>
<td>sh</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
<tr>
<td>bo</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
<tr>
<td>tb</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
<tr>
<td>loc</td>
<td>1</td>
<td>Imported ontologies</td>
</tr>
</tbody>
</table>

Figure 12  Object Feature Vector, received from non-functional properties filtering

<table>
<thead>
<tr>
<th>Term</th>
<th>Weight</th>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>jose</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>clara</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>pezuela</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>location</td>
<td>9</td>
<td>Natural language</td>
</tr>
<tr>
<td>service</td>
<td>21</td>
<td>Natural language</td>
</tr>
<tr>
<td>buyer</td>
<td>18</td>
<td>Natural language</td>
</tr>
<tr>
<td>preferences</td>
<td>9</td>
<td>Natural language</td>
</tr>
<tr>
<td>required</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>information</td>
<td>18</td>
<td>Natural language</td>
</tr>
<tr>
<td>precondition</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>email</td>
<td>6</td>
<td>Natural language</td>
</tr>
<tr>
<td>pk.cup</td>
<td>6</td>
<td>Natural language</td>
</tr>
<tr>
<td>doors</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>aconditioning</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>send</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>describes</td>
<td>3</td>
<td>Natural language</td>
</tr>
<tr>
<td>email destination</td>
<td>3</td>
<td>Natural language</td>
</tr>
</tbody>
</table>

The usage of the Knowledge Domain and Generic Keywords Dictionaries is necessary for retrieving the most appropriate objects during the search and retrieving process. The occurrence of terms in the query with minimal meaning for the search results are corrected in this way that these words will not have a strong impact in calculating the object similarity coefficients.

The weight of each natural language term, which is added to the object feature vector, depends on the number of occurrences of the term (term frequency) in the file and also depends on the type of the term (consideration of TDIF). For
example, knowledge domain terms (terms of the Knowledge Domain Dictionary) have a weight factor 3, generic terms (terms of the Generic Keywords Dictionary) have weight factor 2, and propositions (according to the procedure in 4.5.3.1 terms which are not found in the Generic Keywords or Knowledge Domain are added to the propositions) have a weight factor 1. In the Figure 12 an object feature vector is shown, which consists out of natural language layer terms. These terms are extracted out of the object file by the “non-functional properties filtering procedure”.

The filtering of Non-functional properties enables also collocation parsing (parsing of combined terms). Here fore the admin sets the non-functional parse properties settings in the OMManager / Settings. The admin marks which parts of the non-functional properties section should be parsed (which Dublin Core (DC) elements, for example dc#description, dc#title, dc#publisher and etc.) and the way in which they will be parsed – like a collocation or like a single words. For example, if the property “description” is set to be parsed like a collocation then in the feature vector the whole content of “description” will be presented and handled like an undivided unit (conjunction of terms). This is due to cited characters of some term expressions, which are irrelevant when searched separately.

4.2.4 Filtering of WSDL files

The OM filters “WSDL files” by saving natural language content keywords and metadata keywords in an object feature vectors. This filtering is very close to the non-functional properties filtering. The procedure consists of parsing WSDL files and extracting out of them the “important” natural language keywords. The XML attributes, which content is taken are:

- name,
- type and
- message.

4.2.5 Full text filtering

The OM component supports natural language feature vector generation out of the overall text of the object. This kind of filtering extracts all natural language terms out of the file and then follows the procedure described in section Object preprocessing (4.5.3.1.)

Full text filtering is performed not only on text documents but also on binary files. The binary files text filtering is based on strings extraction out of the content of the file.

This filtering it is yet not used for the purposes of Infrawebs framework, but it is very useful and is intended to use it in future OM implementations as stand-alone component.
4.2.6 Additional Metadata Filtering

Additionally to the WSMO objects and full text filtering the OM component has the capability to create OFVs based on the metadata associated with the objects. Metadata are automatically extracted out of the file containing the object. This is achieved using the libextractor software library. Supported file formats are: ms word, open office, html, pdf, ps, gif, jpeg, png, mp3, mpeg, zip and many others.

4.3 Objects clustering

The clustering algorithm is principally described in the previous deliverable [10] resp. [8] and in the papers [12] and [14].

The objects clustering procedure is processed by using stepwise the MinMax-Algorithm and processing the assignment matrix of the objects given in the collection. As a result the objects of the given collection are grouped in clusters. The assignment matrix of the objects consists of similarity coefficients calculated between every objects. For more details about similarity calculation see [10].

The objects clustering algorithm is implemented in two main steps. Here are briefly described these two phases.

During the first step of clustering algorithm every row of the assignment matrix is processed by the MinMax-Algorithm (for more details see [9]). The max class of objects for each row generates one cluster of objects. Finally the collection of objects is transformed into a collection of clusters.

The second step of the algorithm works with the clusters created in the previous step\(^9\). The assignment matrix of clusters is created. Then this matrix is processed in the same way as described in the first step of the algorithm. Every row of the assignment matrix of clusters is manipulated by the MinMax-Algorithm. The max class of clusters for each row generates one cluster of clusters of objects. At last the repeated clusters are deleted.

The clustering procedure is the basis for the semi-automatic centroids generation. This is furthermore illustrated in the section of the Administration interfaces (section 5.1).

4.4 Objects classification

Objects are classified to subjects of the classification tree based on processing the assignment functions (similarity function). The subjects are characterized and represented by centroids, whereby the centroids are formalized as feature vectors (CFV), which are structural identical with the object feature vectors (OFV). During the classification procedure the SOFV of the objects are matched against the centroids representing the subjects (existing sets of objects). The matching is based on a similarity calculation, which are described analytically in section 4.5. Each matching procedure results in a similarity coefficient. Accord-

\(^9\) A similar strategy is used in [12]
ingly the objects are assigned to the subjects (see [10]). Furthermore a meta clustering [12] is performed, whereby the similarity coefficients between the objects and the given centroid are clustered in three similarity sets – min, middle or max. The min set contains objects with minimal similarity coefficients between them and the subject, the max set contains objects with maximum similarity coefficients between them and the subject and the middle set contains objects with middle coefficients.

The following picture illustrates the classification procedure.

**Figure 13** Classifying by matching feature vector profiles and centroids

### 4.5 Similarity calculation methods

The main analytical formulations implemented by the different similarity calculation methods are given hereafter. These formulas are explained in more detail in [1] and [13]. The following abbreviations are used in the analytical descriptions:

- \( V^1 \) - The feature vector of the first object
- \( V^2 \) - The feature vector of the second object
- \( k_{i1} \) - the weight of the \( i \)-th word from the first feature vector
- \( w_{i1} \) - the \( i \)-th word of the first feature vector
- \( k_{i2} \) - the weight of the \( i \)-th word from the second feature vector
- \( w_{i2} \) - the \( i \)-th word of the second feature vector
4.5.1 Weighted intersection similarity method

The following represents an asymmetric formula (Figure 14) for similarity calculation between 2 feature vector objects. This formula \(^{10}\) (see formula 3 in [13]) is used for calculating the similarity between the different parts of the feature vector of an object and a centroid resp. between the feature vector of an object and a query.

\[
\begin{align*}
  k_{j1}w_{j1} &= <k_{11}w_{11}, ..., k_{n1}w_{n1}> \\
  k_{j2}w_{j2} &= <k_{12}w_{12}, ..., k_{n2}w_{n2}>
\end{align*}
\]

\[
sim (V^1, V^2) = sim (k_{j1}w_{j1}, k_{j2}w_{j2}) = \sum_{j=1}^{n} \frac{1}{\sum_{j=1}^{n} k_{j1}} \sum_{j=1}^{n} \delta (k_{j1}w_{j1}, k_{j2}w_{j2}),
\]

\[
\delta (k_{j1}w_{j1}, k_{j2}w_{j2}) = \begin{cases} 
  1 - \frac{k_{i2} - k_{j1}}{k_{i2} + k_{j1}}, & \text{if } w_{j1} = w_{j2} \\
  & \text{and } k_{j2} \geq k_{j1} ; l_c = \{1, ..., n_c\} \\
  1 - \frac{k_{i2} - k_{j1}}{k_{j1}}, & \text{if } w_{j1} = w_{j2} \\
  & \text{and } k_{i2} < k_{j1} ; l_c = \{1, ..., n_c\} \\
  0, & \text{if } w_{j1} \neq w_{j2}
\end{cases}
\]

Figure 14 Weighted intersection similarity method

4.5.2 Weightless intersection similarity method

The following formula in Figure 15 is an asymmetric similarity calculation formula, which doesn’t take into account the weights of the words in the feature vector. As depicted in [13] (see formula 2) such similarity function calculates the size of intersection between two sets of ontological words and normalize it by the size of the set of words of the query. It’s used in the case of imported ontology queries [3].

\(^{10}\) In [13] this analytical description is formulated for the case of “Calculating similarity for ontological queries by example”. In the current description the general case is considered.
\[
\text{sim}_i(V^1, V^2) = \text{sim} \left( <w_{i1}, \ldots, w_{i1}>, <k_{12}w_{i1}, \ldots, k_{n2}w_{i2} > \right) = \\
= \frac{1}{n_i} \sum_{j=1}^{n_i} \delta \left( w_{j1}, <k_{12}w_{i1}, \ldots, k_{n2}w_{i2} > \right), \\
\text{where} \quad \delta \left( w_{j1}, <k_{12}w_{i1}, \ldots, k_{n2}w_{i2} > \right) = \begin{cases} 
1, & \text{if } w_{j1} = w_{i1}, \quad l_2 = \{1, \ldots, n_2\} \\
0, & \text{otherwise} 
\end{cases}
\]

Figure 15 Weightless intersection similarity method

4.5.3 Lexical similarity

The lexical similarity is calculated in a more complex way. Before applying the similarity calculation formula a sophisticated preprocessing of the object is performed.

4.5.3.1 Object preprocessing

The natural language query or the natural language text extract (for example in the case of non-functional properties filtering of object) passes through the following preprocessing steps:

1. Non-alphabetical letters excluding
2. Composite words tokenization
3. Substitute Abbreviations
4. Stop words exclusion
5. Synonyms substitution
6. Similar words substitution

The feature vector of the query or the feature vector of the object (in the case of non-functional properties filtering of object) is composed by the words given as result of this preprocessing.

Non-alphabetical letters excluding

The following symbols are replaced with spaces: `~!@#$%^&*()-_=+|}{\[\]’";,./?.,><.

Composite words tokenization

In some situations the query (or the natural language text extract) can contain words composed by two or more other words. This is the case when variable, methods or services names are cited in the query or in the file. An example for such composite word can be findHotelByLocation.

Capital letters are used as separators for the tokenization. The given above example will be separated in the following simple words: retrieve, hotel, by and location.

It has to be mentioned here that other forms of composite words as minus-separated composite words will be covered by the first step of the preprocessing.
Substitute abbreviations
The OM module supports dictionary of abbreviations. This dictionary consists of abbreviation words and their full equivalents. The Administrator of the system fulfills it. In this step the query (or the natural language text extract) is checked for abbreviations and if such are found they are substituted with their full equivalents.

“Stop Words” exclusion
Stop words are words that have no importance for the search. They are also called non-informational words – too generic words that don’t influence the meaning of the query or the object description. Such words are ‘and’, ‘or’, ‘as’, ‘not’ etc. For the specific domain additional non-informational words can be added to the system dictionary. All occurrences of stop words listed in the system dictionary “Non-info Words” are replaced with empty string in the query (or in the natural language text extract).

Synonyms and similar words substitution
The OM module uses a list of standard words. These standard words could be knowledge domain words or generic words depending on the specific domain. All objects in the system are described by these words. The OM supports Knowledge Domain Dictionary and Generic Keywords Dictionary which are used in this procedure.

To have relevant similarity calculation the query’s words (the natural language extract words) have to replaced with relevant words out of the system dictionaries – Knowledge Domain Dictionary or Generic Keywords Dictionary.

The first technique is to achieve that is the replacement of the query’s (natural language text extract) words with synonyms out of the system dictionary. In the case where no synonym to the given word is found in the system dictionary, the word is added to the Prepositions Dictionary. The similarity between the words is calculated using the WordNet (http://wordnet.princeton.edu/) library.

The Administrator of the system could transfer the words from the propositions Dictionary to another system dictionary – Knowledge Domain Dictionary or Generic Dictionary.

The Synonyms Dictionary is actualized every time a new word is added to the Knowledge Domain Dictionary or to the Generic Keywords Dictionary.

4.5.3.2 Analytical description of the lexical similarity
As described in [13] the lexical similarity between a natural language query and an OFV of a WSMO object is carried out in 2 stages:

1) preprocessing the query and
2) matching the preprocessed query to the OFVs.
3) The preprocessed query is matched against the “lexical part” of the OFV.
The following analytical formulation (Figure 16) represents the similarity calculation between the OFV of the query and the OFV of the corresponding NFP section of the WSMO object.

\[
sim(V^1, V^2) = \frac{1}{\sum_{l=1}^{n} k_{ml} + \sum_{j=1}^{n} \Delta(k_{j1}w_{j1}, k_{j2}w_{j2},...,k_{n2}w_{n2})} \sum_{j=1}^{n} \delta(k_{j1}w_{j1}, k_{j2}w_{j2},...,k_{n2}w_{n2}),
\]

where \( \delta(k_{j1}w_{j1}, k_{j2}w_{j2},...,k_{n2}w_{n2}) = \begin{cases} k_{j1} + k_{j2}, & w_{j1} = w_{j2}, l = \{1,...,n\} \\ 0, & w_{j1} \neq w_{j2} \end{cases} \)

and \( \Delta(k_{j1}w_{j1}, k_{j2}w_{j2},...,k_{n2}w_{n2}) = \begin{cases} k_{j1}, & w_{j1} = w_{j2}, l = \{1,...,n\} \\ 0, & w_{j1} \neq w_{j2} \end{cases} \)

Figure 16 Analytical description of the lexical similarity

The neglection of the weights of co-occurring words yields in an asymmetrical similarity function [13].

---

11 In [13] the second part of the matching procedure is equal to the NFP section of a stored "case". In this context the stored case is equivalent to stored SOFV of WSMO object.
5 OM GUIs & Interfaces

In this chapter the main GUIs and interfaces of the realized software package is described and illustrated by screenshots.

5.1 Administration Interface

In the following the OM GUIs and Interfaces are illustrated and described. Hereinafter the Administrator interface is shown (Figure 17).

![Figure 17 Main form of the Administration Module](image)

The main form provides several manipulation capabilities like: search, add, edit and delete subjects. Also classified objects can be shown in there assignments to selected subjects resp. centroids, whereby the subjects are organized in the subject classification tree [8], [10]. Each object is assigned to a centroid via a similarity coefficient and the centroid itself belongs to a given node of the subject tree. The subject is characterized with its parent ID subject, subject ID and subject title.

The main functions for manipulating subjects are DeleteSubject(string subjectId) – deletes the subject with ID “subjectId” and returns true or false depending on the success of the operation, AddSubject(OMSubject subject) – adds a new subject in the data store and SaveSubject(OMSubject subject, string subjectId) – updates the information about the subject in the local data store. The second parameter subjectId represents the old ID of the subject and is required to be able retrieve the modified subject in the data storage.

The next screenshot in Figure 18 depicts the tree view of the subject’s tree.
In following Figure 19 the form for add/edit subject is shown. The parent subject ID can be selected from the available subjects or entered manually. Save button populates the changes in the database. Cancel button closes the form.
The classified objects can be shown in different relations: e.g. selected subject with similarity coefficients and class type max, middle, min as illustrated in Figure 20.

![Figure 20](image)

Figure 20
Show classified objects – select subject and "Subjects Tree/ Show classified objects" from main menu, or select subject and "Show classified objects" from context menu on subject’s tree

The acquired knowledge objects are automatically filtered and classified. Objects can be shown in the objects panel by different views – “Details”, “List”, “Large icons”, “Small icons”. The object panel, shows object’s identification, object’s type and annotation.

The main method used for searching objects is SearchObjects(ObjectFilter filter). In the filtering procedure the searched object ID, object type, annotation and the amount of the returned results can be specified. After a performed search operation the objects selected in the objects panel are populated regarding their results.

From the “Search Objects” panel (Figure 21), users can retrieve features for each objects and retrieving them in relation to different fields: - object ID, type and annotation.
In Figure 21 is exemplarily shown the search for objects containing the term “flight” in their object identification.

All the operations with objects are executed after the selection of one or more objects out of the list. New and existing objects can be added, deleted, edited, filtered, classification and/or similarity shown and centroids proposition.

In Figure 22 several functions are shown in the context menu.

Different views to the objects list

The Figure 23 shows the form for edit (add) a single object. Objects could be added by selecting wsml/wsdl file and the text boxes will be automatic filled with the information extracted out of the files. Supported are object identification, type and annotation. Upon pressing save button after retrieving an object file several sequential actions are triggered: first, the Filtering Component generates (semi-) automatically an Object Feature Vector representation.(Prefilter Object (string fileContent)).

Then a new object is created and passed through to the “Data Layer Component” to be stored in the local database. Afterwards the object is classified against the classification tree. For this purpose the “Classification Layer” is used. The results of the classification process are processed to the Data Layer Component to be stored for further usage.
Figure 22 List and Select Objects

Figure 23 Insert Object
Figure 24 Editing objects – by selecting object from objects panel with "Edit" from the context menu or from main menu "Objects/ Edit".

The form in Figure 24 and Figure 25 for adding multiple objects performs the steps of inserting object described above for all the objects contained in the supplied directory.
Figure 26 shows the classification results of selected objects. An object is classified to the classification tree based on the similarity calculation, which yields an assignment function resp. similarity coefficient. Depending on the "min max algorithm" for objects clustering, three meta groups are constructed - max, middle or min group.

The user can change manually the similarity coefficients in case of unfeasible results (semi-automatic classification). There's possibility to classify objects from the left part again, by a selection and using the "Classify" button (apart from automatic classification of initially received objects). In this case a recalculation of the similarity and assignment function is triggered by execution of "min max algorithm" in the objects classification routine. All classifications are performed using the Classification Component of the OM.

The other OM module functionality gives the similarity matrix of selected objects, which is generated in the Clustering Component. "Show Similarity matrix" button opens a form for showing the similarity matrix (Figure 27).

The similarity matrix is also shown just by selecting objects and clicking on "Show similarity matrix" button in the context menu. In this way the clustering procedure is skipped.

In Figure 27 a matrix is shown, which contains similarity coefficients between the selected objects. The degree of similarity between objects is represented by a normalised coefficient in the range of 0 to 1. 1 represents identical objects.
OM module keeps different types of dictionaries and related terms and keywords: knowledge domain keywords, generic keywords, abbreviations, Stop Terms, synonyms and propositions. Keywords are mainly used within creation process of object feature vectors Figure 29. The topic “Keywords/ Edit Keywords” from main menu provides comprehensive maintenance facilities (Editing, Adding, Deleting) of the different dictionaries.

In Figure 28 also the function “Propositions” is shown: Each proposition could be added to one of the different types of keywords.
During the creation of an object, metadata information is processed and new propositions are presented.

Users can filter object, although an automatically filtering is triggered, in the case a new – unknown – object is retrieved.

An administration section provides facilities for modifying and tuning the automatically generated object feature vectors.

The OM module provides comprehensive clustering facilities. In the simplest way it clusters objects manually selected by the Admin. By clicking the “Create Centroid” the creation of centroids (Figure 30) is triggered, which yields in the determination of a feature vector representing the cluster.
A subject Id, subject title, centroid name, layer, keywords and the Feature Vector itself characterize each centroid. Each subject have multiple centroids, due to the multilayer classification tree.

A further admin tool allows the maintenance (add, delete, edit or propose) of the centroids. The next two figures show “Centroid Edit” form, which is used to add or edit centroid (Figure 31, Figure 32).
Figure 31 Maintenance Menu for Centroids

Figure 32 Centroid Form: add new centroids from main menu "Centroid/ New" or from context menu on subjects tree, right button click "Edit centroid" for the selected subject.
Figure 33 and Figure 34 shows the panel for visualizing of existing centroids for further maintenance. The context menu of the centroids provides the ability to the relation of the objects to currently selected centroids.

Adding new objects to the DSWS-R or SIR or some of the existing ones are modified a notification message is sent to OM.

This message is received from the Web Service Component and the information is stored in queue for further maintenance. The next form shows the records made in the queue and provides the ability to process them Figure 35.

Pressing “Process Queue” button a sequence of actions is triggered.

First of all, using the Objects Retrieval Component, for the first record in the queue the corresponding file is retrieved from its storage location. Afterwards the filtering component is used to construct the object feature vector. Then the newly created object is stored in the database server using the data layer component.

After that this new object is classified to the already existing ones from the classification component and the result is entered in the data store too. These steps are performed for each record in the queue.

In the following form (Figure 36) the admin provides information, in which way the non-functional properties have to be manipulated. There are two options – single and set. Selecting the single option means that the whole phrase that is enclosed in quotes will be treated as single word. In this form also the structure of the annotation is defined too. The annotation contains all NFP, for which the checkbox “Use in Annotation” is checked.

The OM module contains and Windows Service Figure 37, which is used to process the modified objects queue, when the application is not running. Here can be set the time, when to manipulate the queue – usually the time when the load of the server is lowest. The example settings mean that the queue will be processed every month, every day at 6:00 and 22:00 o’clock.
Figure 33 Show centroids

Figure 34 Show classified objects to centroid
Figure 35 Modified Objects Queue

Figure 36 NFP parse settings

Figure 37 Queue processing settings
5.2 Web Service

The OM module exposes OMSearch Web Service to the other INFRAWEBS components. They will interact with the OM through this Web Service. The list of supported operations that could be executed through the OMSearch Web Service are shown and described in the following tables:

<table>
<thead>
<tr>
<th>Interface name</th>
<th>OMSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation name</td>
<td>SearchForWSMOObjects</td>
</tr>
<tr>
<td>Component</td>
<td>OM (Organizational Memory)</td>
</tr>
<tr>
<td>Operation Description</td>
<td>Web method for searching objects by structured query.</td>
</tr>
<tr>
<td>Operation Signature</td>
<td>Operation SearchForWSMOObjects[query: Query]: SearchResult</td>
</tr>
<tr>
<td>Complex Types</td>
<td>Name: SearchResult</td>
</tr>
<tr>
<td></td>
<td>Consists of list with objects, similar to the given query. Contains statistical data about the search. For XSD description see appendix.</td>
</tr>
<tr>
<td></td>
<td>Name: Query</td>
</tr>
<tr>
<td></td>
<td>Consists of elements defining how the found objects should look like – the type of the searched objects, nonfunctional properties, capability properties, imported ontologies. Also contains information about the maximum number of the elements to be returned in the search result, similarity threshold and WSDL object, used for searching ontologies. For XSD description see appendix.</td>
</tr>
<tr>
<td>Additional Info</td>
<td>This method is also used for retrieving ontologies by WSDL descriptions. The WSDL file content is set in Query object. The Designer, Composer and Goal Editor will use this OM operation as case based memory for retrieving already created WSMO objects. These objects will be used as basis for the creation of new similar WSMO objects. The discovery module will use this OM operation for the first step discovery.</td>
</tr>
</tbody>
</table>
### Table 2  Search for similar objects operation

<table>
<thead>
<tr>
<th>Interface name</th>
<th>OMSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation name</td>
<td>SearchForSimilarObjects</td>
</tr>
<tr>
<td>Component</td>
<td>OM (Organizational Memory)</td>
</tr>
<tr>
<td>Operation Description</td>
<td>Web method for searching objects similar to a given WSMO object. The object is represented by its string content and object type.</td>
</tr>
<tr>
<td>Operation Signature</td>
<td>Operation SearchForSimilarObjects[obj: xsd:string, objectType: ObjectType, maxResult: xsd:int]: SearchResult</td>
</tr>
<tr>
<td>Complex Types</td>
<td>Name: SearchResult</td>
</tr>
<tr>
<td></td>
<td>Consists of list with objects, similar to the given search parameters. Contains statistical data about the search.</td>
</tr>
<tr>
<td></td>
<td>For XSD description see appendix.</td>
</tr>
<tr>
<td>Additional Info</td>
<td>The Designer, Composer and Goal Editor will use this OM operation as case based memory for retrieving already created WSMO objects. These objects will be used as basis for the creation of new similar WSMO objects. The discovery module will use this OM operation for the first step discovery.</td>
</tr>
</tbody>
</table>

### Table 3  Search for WSDL objects operation

<table>
<thead>
<tr>
<th>Interface name</th>
<th>OMSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation name</td>
<td>SearchForWSDLObjects</td>
</tr>
<tr>
<td>Component</td>
<td>OM (Organizational Memory)</td>
</tr>
<tr>
<td>Operation Description</td>
<td>Web method for searching WSDL objects by structured query.</td>
</tr>
<tr>
<td>Operation Signature</td>
<td>Operation SearchForWSDLObjects [query: WsdlQuery] : SearchResult</td>
</tr>
<tr>
<td>Complex Types</td>
<td>Name: SearchResult</td>
</tr>
<tr>
<td>Complex Types</td>
<td>Name: WsdlQuery</td>
</tr>
</tbody>
</table>

| Interface name | OMSearch |
| Operation name | NewOrModifiedObjects |
| Component | OM (Organizational Memory) |
| Operation Description | Web notification message, sent to the OM module, when new or modified objects in SIR or DSWS-R components appear. |
| Operation Signature | Operation **NewOrModifiedObjects**[message: Message]: **xsd:boolean** |
| Pre condition | This operation is called when in the DSWS-R or SIR new objects appear or existing objects are modified or deleted. |
| Post condition | The OM component maintains objects queue with new, modified or deleted objects. This queue is fulfilled when notification is received and is processed periodically depending on local configuration settings. |
| Complex Types | Name: Message | Contains message type, message sender URI, message sender type and objects information. For XSD description see appendix. |
| Additional Info | This operation will be used by DSWS-R and SIR components. |
### Table 5  Is Alive operation

<table>
<thead>
<tr>
<th>Interface name</th>
<th>OMSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation name</td>
<td>IsAlive</td>
</tr>
<tr>
<td>Component</td>
<td>OM (Organizational Memory)</td>
</tr>
<tr>
<td>Operation Description</td>
<td>This operation checks if OM Web Service is alive. Returns “true” for success and “false” for failure.</td>
</tr>
<tr>
<td>Operation Signature</td>
<td>Operation IsAlive[] : xsd:boolean</td>
</tr>
</tbody>
</table>

### Table 6  Start operation

<table>
<thead>
<tr>
<th>Interface name</th>
<th>OMSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation name</td>
<td>Start</td>
</tr>
<tr>
<td>Component</td>
<td>OM (Organizational Memory)</td>
</tr>
<tr>
<td>Operation Description</td>
<td>This operation is used for starting the OM Web Service. Returns “true” for success and “false” for failure.</td>
</tr>
<tr>
<td>Operation Signature</td>
<td>Operation Start[] : xsd:boolean</td>
</tr>
</tbody>
</table>

### Table 7  Shut down operation

<table>
<thead>
<tr>
<th>Interface name</th>
<th>OMSearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation name</td>
<td>ShutDown</td>
</tr>
<tr>
<td>Component</td>
<td>OM (Organizational Memory)</td>
</tr>
<tr>
<td>Operation Description</td>
<td>This operation is used for shutting down the OM Web Service. Returns “true” for success and “false” for failure.</td>
</tr>
<tr>
<td>Operation Signature</td>
<td>Operation ShutDown[] : xsd:boolean</td>
</tr>
</tbody>
</table>

The following table describes the types used in the OM Web methods.
| SearchResult | <xsd:complexType name="SearchResult">
| ResultSet |   <xsd:sequence>
| ResultSet |     <xsd:element name="SearchTime" type="xsd:float" minOccurs="1" maxOccurs="1"/>
| ResultElement |     <xsd:element name="ResultSet" type="xsd1:ResultSet" minOccurs="1" maxOccurs="1"/>
| | </xsd:sequence>
| | </xsd:complexType>
| | <xsd:complexType name="ResultSet">
| |   <xsd:sequence>
| |     <xsd:element minOccurs="0" maxOccurs="unbounded" name="ResultElement" type="xsd1:ResultElement"/>
| | </xsd:sequence>
| | </xsd:complexType>
| | <xsd:complexType name="ResultElement">
| |   <xsd:sequence>
| |     <!-- OntologySimilarity is float number from 0 to 1 or "undefined"! --&gt;
| |     <xsd:element name="ontologySimilarity" type="xsd:float"/>
| |     <!-- LexicalSimilarity is float number from 0 to 1 or "undefined"! --&gt;
| |     <xsd:element name="lexicalSimilarity" type="xsd:float"/>
| |     <!-- Pointer to the object description in the repository - identifier --&gt;
| |     <xsd:element name="identifier" type="xsd:string"/>
| |     <!-- The type of the object --&gt;
| |     <xsd:element name="objectType" type="xsd1:ObjectType"/>
| |     <!-- Metadata --&gt;
| |     <xsd:element name="description" type="xsd:string"/>
| | </xsd:sequence>
| | </xsd:complexType>
| Query | <xsd:complexType name="Query">
| Query |   <xsd:sequence minOccurs="1" maxOccurs="1">
| Query |     <xsd:element name="objectType" type="xsd1:ObjectType" minOccurs="0" maxOccurs="1" default="service"/>
| Query |     <xsd:element name="maxResult" type="xsd:int" minOccurs="0" maxOccurs="1" default="10"/>
| |     <!-- main similarity threshold; only services which similarity (ontological - in the case of ontological or complex query or lexical - in a case of pure natural language query) is greater or equal then this threshold will be included in to the list with the query results -->
| Query |     <xsd:element name="similarityThreshold" type="xsd:float" minOccurs="0" maxOccurs="1" default="0.80"/>
| Query |     <xsd:element name="nfp" type="xsd1:NFP" minOccurs="0" maxOccurs="1"/>
| Query |     <xsd:element name="importedOntologies" type="xsd1:ImportedOntologies" minOccurs="0" maxOccurs="1"/>
| Query |     <xsd:element name="capability" type="xsd1:Capability" minOccurs="0" maxOccurs="1"/>
| Query |     <xsd:element name="wsdlObject" type="xsd1:WsdlObject" minOccurs="0" maxOccurs="1"/>
| | </xsd:sequence>
| WsdlQuery | </xsd:complexType>
| WsdlQuery | <xsd:complexType name="WsdlQuery">
| WsdlQuery |   <xsd:sequence minOccurs="1" maxOccurs="1">
| WsdlQuery |     <!-- keywords matching against short names of operation, messages, datatypes etc. used in WSDL files--&gt;
| WsdlQuery |     <xsd:element name="content" type="xsd1:WeightedKeywordsString" minOccurs="0" maxOccurs="1"/>
| WsdlQuery |     <!-- keywords matching against DC annotations added by SIR--&gt;
| WsdlQuery |     <xsd:element name="metadata" type="xsd1:WeightedKeywordsString" minOccurs="0" maxOccurs="1"/>
| WsdlQuery |     <!-- main similarity threshold; only services which lexical similarity is greater or equal to this threshold will be included in to the list with the query results -->
| WsdlQuery |     <xsd:element name="maxResult" type="xsd:int" minOccurs="0" maxOccurs="1" default="100"/>
| | </xsd:sequence>
| | </xsd:complexType>
then this threshold will be included in to the list with the query results:

```xml
<xs:element name="similarityThreshold" type="xsd:float" minOccurs="0" maxOccurs="1" default="0.00"/>
</xs:sequence>
</xs:complexType>

**ImportedOntologies**

```xml
<xs:complexType name="ImportedOntologies">
<xs:sequence>
<xs:element name="ontologyName" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

**Capability**

```xml
<xs:complexType name="Capability">
<xs:sequence>
<xs:element name="assumption" type="xsd1:Assumption" minOccurs="0"/>
<xs:element name="precondition" type="xsd1:Precondition" minOccurs="0"/>
<xs:element name="postcondition" type="xsd1:Postcondition" minOccurs="0"/>
<xs:element name="effect" type="xsd1:Effect" minOccurs="0"/>
<xs:element name="serviceFunctionality" type="xsd1:ServiceFunctionality" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

**NFP**

```xml
<xs:complexType name="NFP">
<xs:sequence>
<xs:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
<xs:element name="content" type="xsd:string"/>
</xs:sequence>
</xs:complexType>

**Assumption**

```xml
<xs:complexType name="Assumption">
<xs:sequence>
<xs:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
<xs:element name="byExample" type="xsd:boolean" minOccurs="0" maxOccurs="1" default="false"/>
<xs:element name="content" type="xsd:string"/>
</xs:sequence>
</xs:complexType>

**Precondition**

```xml
<xs:complexType name="Precondition">
<xs:sequence>
<xs:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
<xs:element name="byExample" type="xsd:boolean" minOccurs="0" maxOccurs="1" default="false"/>
<xs:element name="content" type="xsd:string"/>
</xs:sequence>
</xs:complexType>

**Postcondition**

```xml
<xs:complexType name="Postcondition">
<xs:sequence>
<xs:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
<xs:element name="byExample" type="xsd:boolean" minOccurs="0" maxOccurs="1" default="false"/>
<xs:element name="content" type="xsd:string"/>
</xs:sequence>
</xs:complexType>

**Effect**

```xml
<xs:complexType name="Effect">
<xs:sequence>
<xs:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
<xs:element name="byExample" type="xsd:boolean" minOccurs="0" maxOccurs="1" default="false"/>
</xs:sequence>
</xs:complexType>
<table>
<thead>
<tr>
<th>Component</th>
<th>XSD Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceFunctionality</td>
<td><code>&lt;xsd:complexType name=&quot;ServiceFunctionality&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:sequence&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;weight&quot; type=&quot;xsd:int&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot; default=&quot;1&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;content&quot; type=&quot;xsd:string&quot; /&gt;</code></td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:sequence&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:complexType&gt;</td>
</tr>
<tr>
<td>WsdlObject</td>
<td><code>&lt;xsd:complexType name=&quot;WsdlObject&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:sequence&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;weight&quot; type=&quot;xsd:int&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot; default=&quot;1&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;byExample&quot; type=&quot;xsd:boolean&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot; default=&quot;true&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;content&quot; type=&quot;xsd:string&quot; /&gt;</code></td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:sequence&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:complexType&gt;</td>
</tr>
<tr>
<td>WeightedKeywordsString</td>
<td><code>&lt;xsd:complexType name=&quot;WeightedKeywordsString&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:sequence&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;keywords&quot; type=&quot;xsd:string&quot; /&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;weight&quot; type=&quot;xsd:int&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot; default=&quot;1&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:sequence&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:complexType&gt;</td>
</tr>
<tr>
<td>Message</td>
<td><code>&lt;xsd:complexType name=&quot;Message&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:sequence&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;type&quot; type=&quot;xsd1:MessageType&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot; default=&quot;NewOrModifiedObjects&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;senderURI&quot; type=&quot;xsd:string&quot; minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;senderType&quot; type=&quot;xsd1:SenderType&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot; default=&quot;Repository&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;objects&quot; type=&quot;xsd1:MessageObject&quot; minOccurs=&quot;1&quot; maxOccurs=&quot;unbounded&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:sequence&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:complexType&gt;</td>
</tr>
<tr>
<td>MessageObject</td>
<td><code>&lt;xsd:complexType name=&quot;MessageObject&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:sequence&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;type&quot; type=&quot;xsd1:ObjectType&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot; default=&quot;service&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;status&quot; type=&quot;xsd1:ObjectStatus&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot; default=&quot;added&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsd:element name=&quot;iri&quot; type=&quot;xsd:string&quot; minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:sequence&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:complexType&gt;</td>
</tr>
</tbody>
</table>
Enumerations:

| ObjectType      | <xsd:simpleType name="ObjectType">
|-----------------|-----------------------------------
|                 |   <xsd:restriction base="xsd:string">
|                 |       <xsd:enumeration value="all"/>
|                 |       <xsd:enumeration value="service"/>
|                 |       <xsd:enumeration value="goal"/>
|                 |       <xsd:enumeration value="mediator"/>
|                 |       <xsd:enumeration value="ontology"/>
|                 |       <xsd:enumeration value="wsdl"/>
|                 |       <xsd:enumeration value="bpel"/>
|                 |   </xsd:restriction>
|                 | </xsd:simpleType> |

| MessageType     | <xsd:simpleType name="MessageType">
|-----------------|-----------------------------------
|                 |   <xsd:restriction base="xsd:string">
|                 |       <xsd:enumeration value="NewOrModifiedWSMOObjects"/>
|                 |       <xsd:enumeration value="NewOrModifiedObjects"/>
|                 |       <xsd:enumeration value="Advertisment"/>
|                 |   </xsd:restriction>
|                 | </xsd:simpleType> |

| SenderType      | <xsd:simpleType name="SenderType">
|-----------------|-----------------------------------
|                 |   <xsd:restriction base="xsd:string">
|                 |       <xsd:enumeration value="Repository"/>
|                 |       <xsd:enumeration value="Registry"/>
|                 |       <xsd:enumeration value="SIR"/>
|                 |   </xsd:restriction>
|                 | </xsd:simpleType> |

| ObjectStatus    | <xsd:simpleType name="ObjectStatus">
|-----------------|-----------------------------------
|                 |   <xsd:restriction base="xsd:string">
|                 |       <xsd:enumeration value="added"/>
|                 |       <xsd:enumeration value="modified"/>
|                 |       <xsd:enumeration value="deleted"/>
|                 |       <xsd:enumeration value="unchanged"/>
|                 |   </xsd:restriction>
|                 | </xsd:simpleType> |
5.3 Web Demo interface

With http://infrawebs.aspasia-systems.de/ the OM Web Demo Interface can be accessed.

The main purpose of the Web Demo Component is to show the usage of the OM Web Service. In this demo all web service operations can be tested. The interface is explicitly described and easy to use. The first page of the Web Demo is given in Figure 38.

Infrawebs OM Module demo site v0.4

Welcome to demo site of the Infrawebs & FCS-OM Module!

Infrawebs system represents an application-oriented software "tool set" for creating, maintaining and executing WSMO-based Semantic Web Services within the whole life cycle.

The FCS-OM (Fuzzy Concept Set - Organizational Memory) module is an important part of Infrawebs project. This module represents a similarity based organizational memory and case based recommender tool. The component classifies the system knowledge objects using fuzzy logic techniques and statistical and linguistic algorithms as fuzzy matching functions. In this demo site you can perform some testings on the FCS-OM module based on semantic web services searching. The searching is executed among dozens of classified semantic web services.

1. In the "OMSearch service" you can see the operations for accessing FCS-OM module functionality. For now we support five operations - "SearchForWSMOObjects", "SearchForSimilarObjects" and etc. The operation "SearchForWSMOObjects" finds appropriate objects depending on different queries that might be used as searching parameter. The operation "SearchForSimilarObjects" finds objects that are similar to a concrete WSML object. You can read more about these and the others supported service operations here.

2. Here you can find demo web client for testing the above web service operations - OMWebClient

Figure 38  OM Web Demo Interface

The user can test the following OM functionalities (Figure 39) by selecting the desired link for chosen operation, entering different data, executing the operation and examine the results:

- search for objects by query;
- search for similar objects;
- search for ontologies by WSDL file;
- search for WSDL file by keywords;
- NewOrModifiedObjectsMessage;
The Web Demo user could also Figure 39:

- check if, the OM Web Service is alive - button “IsAlive”;
- shut down the service from “ShutDown” button;
- and start the service, if it is closed from “Start” button;

The execution of first four operations “Search for objects by query”, “Search for similar objects”, “Search for ontologies by WSDL file” and “Search for WSDL file by keywords” produces a list of objects like following (Figure 40):
The last operation “NewOrModifiedObjects Message” shows the effect of passing messages to the OM. These notification messages could be passed by SIR or DSWS-R components. If the messages are received correctly the OM Web Demo will return “success”, otherwise it will return “failure”.

This operation executes the OM Web Service method `NewOrModifiedObjects [message: Message]: xsd:boolean`. This method is described in 5.2. The user could try this operation by using the following Web Demo interface (Figure 41).
Here you can view the effect of passing NewOrModifiedObjects message!
Try with different objects and see the corresponding results.

**Message Type:** NewOrModifiedObjects

**Sender URL:** http://ortodemo.sima.bg/dsws/refwebservices/DSWS-RService

**Sender Type:** DSWSRepository

**Object 1:** Type: service
Status: added
URL: http://example.org/sfs/ShuttleManWS

**Object 2:** Type: ontology
Status: modified
URL: http://example.org/sfs/booking

The operation “Search for objects by query” executes following Web Service method:

**Operation** SearchForWSMOOObjects[query: Query]: SearchResult

This operation is described in details in 5.2.

The user is able to enter different types of data in query interface and examine the found objects results. The OM Web Demo offers following interface for this operation:
The operation “Search for similar objects” executes following Web Service method (Figure 42):

**Operation SearchForSimilarObjects**

- **objects**: `xsd:string`
- **objectType**: `ObjectType`
- **maxResult**: `xsd:int`

SearchResult

This method is described in details in 5.2.

The user could enter different WSML file contents and see their similar objects and calculated similarity coefficients.
The operation “Search for ontologies by WSDL file” represents the OM Web Service operation `SearchForWSMOObjects`[query: Query]: SearchResult

This operation is described in details in 5.2 (Figure 43).

In this situation the user may enter different WSDL files contents and search ontologies by them. In the next figure is shown the interface that Web Demo offers for the execution of this search operation.

The OM Web Demo presents an operation called “Search for WSDL file by keywords”. It is a demonstration to the usage of OM Web Service operation `SearchForWSDLObjects` [query: WsdlQuery] : SearchResult.

This operation is described in detail in 5.2. The user could apply content and metadata keywords in the following form and search for WSDL files (Figure 44, Figure 45).
Here you can test the operation for finding ontology objects by WSDL file! You have to paste WSDL file content in the text area to search for ontology objects.

Try with different WSDL files and see the corresponding results.

Max Results: 10  
Similarity Threshold: 0.01

Ontology Description:  Weight: 1
booking
flight

WSDL file content:  Weight: 1

<types>
    <element name="TravelDetails">
      <sequence>
        <element name="TravelerName" type="string"/>
        <element name="TravelType" type="string"/>
      </sequence>
    </element>
  </schema>
</types>

Figure 44 OM Web Demo interface for the operation “Search for ontologies by WSDL file”

Infrawebs OM Module demo site v0.4

Here you can test the operation for finding WSDL files by keywords!

Try with different keywords and see the corresponding results.

Content keywords:
weather fetcher

Weight: 1

Metadata keywords:

Weight: 1

Max Results: 100  
Threshold: 0.0

Figure 45 OM Web Demo interface for the operation “Search for WSDL file by keywords”
6  OM DESIGN DETAILS

In the following chapter the OM design details are pointed out by describing on one hand the OM components and on the other hand the Components interaction and interoperation.

6.1  OM Components

6.1.1 Data Layer Component

This software component is responsible for storing and retrieving of the OM data objects. The implementation uses MySQL database server. The basic objects stored in the database are the objects with feature vectors, the centroids with feature vectors, the similarity between the centroids and the objects, the system dictionaries etc..

The main operations are:

- SaveObject,
- DeleteObject,
- FillObjectFeatureVector,
- SaveCentroid,
- DeleteCentroid,
- FillCentroidFeatureVector,
- GetSimilarity,
- SaveClassifications,
- GetObjectClassification,
- FillKeywords,
- SaveKeywords,
- FillSynonyms,
- UpdateSynonyms

Figure 46 gives the record and table configuration of the used Database. The MySQL Data-Layer is given in Figure 47.
Figure 46 Record and table configuration
Figure 47 MySQL Data-Layer
6.1.2 Objects Retrieval Component

The Objects retrieval component deals with retrieving the framework objects out of their data stores – in case of WSMO objects this is the DSWSRepository and for non-semantic objects such as WSDL and BPEL the SIR. For this purpose the Web service exposed by the Integrated Framework is used.

If new objects are stored in the DSWSRepository or SIR a notification message is sent to OM. OM stores the information in a queue for later processing. The queue is stores also information about the modified and deleted objects.

There are two basic operations supported (Figure 48):

- AddModifiedObjects
- ProcessModifiedObjects

The queue can be processed explicitly or periodically depending on configuration settings.

6.1.3 Classification Component

This meta classification component is used to divide the similarities between different objects in three classes – MIN, MAX and MIDDLE, Figure 49. For that purpose the similarities calculated from the Similarity Calculator component are
used. Furthermore this component is used by the Clustering component for creating centroids and by the Discovery Component for retrieving similar objects by different criteria.

6.1.4 Clustering Component

The Clustering Component (Figure 50) is responsible for the Centroids generation and maintenance. The clusterization process is done in two steps. First – from the input objects a similarity matrix is created. Each row is classified using the Classification component and new Cluster is created with the objects from the MAX class. Then the repeated clusters are cleared.

At the second step from the remaining clusters a new Similarity Matrix is created and processed. The clusters from the MAX class are taken to create new clusters. The repeated clusters are cleared and the remaining ones are the final result.
6.1.5 Similarity Calculator Component

This component (Figure 51) calculates the ontology and lexical similarity between two feature vectors using different algorithms. This Component is used from the Classification Component. It implements both symmetric and asymmetric formulas for similarity calculations.

![Diagram of Implementation Classes for the Similarity Calculator]

Figure 51 Implementation Classes for the Similarity Calculator
6.1.6 Filtering Component

The filtering component (Figure 52) is responsible for the generation of the feature vectors on objects, centroids and queries.

6.1.7 Discovery Component

This component (Figure 53) is used by the Web Service Component for retrieving WSMO objects or non-semantic (WSDL, BPEL) objects similar to a given query or another object.

Out of the Query a Feature Vector is created using the Filtering Component. This OFV is classified against the existing centroids and the most similar objects of the most similar centroid are returned.
6.1.8 Web Service Component

The Web Service Component (Figure 54) exposes the functionality of the OM module to the other INFRAWEBS modules. It is used for retrieving similar objects to a given object or structured query, for notification about new, modified or deleted objects from SIR or DSWS-R and for administrative purposes – to check the status (IsAlive method), to start and stop the service (Start and Shut-Down methods).

6.1.9 GUI Component

The Graphic User Interface (GUI) Component (Figure 55) is used for administrative purposes of the OM module. The GUI Component provides also the ability to edit clusters and objects, import new objects, classify the objects, organize the objects to a specific subject. The view of the subjects can be tree-
like or list-like. The GUI also deals with the management of the system dictionaries.
6.1.10 Web Demo Component

This component (Figure 56) is used for testing and demonstrating the functionality of the OM module. It is developed in ASP.Net as web site and demonstrates the main abilities of the OM. The interaction between the Web Demo Component and the OM is done through the Web Service Component.

Figure 56 Web Demo Component
6.2 Components interactions

The OM Components are grouped in three layers – Data Layer Components, Business Logic Components and Interface Components. The Figure 57 shows interactions between the components from different layers and the interactions between components from one layer (see also Appendix 1).

Figure 57  OM Components Internal Interactions
7  WORK FOR THE NEXT PROJECT PERIOD

Most of the tasks given in the road maps are successfully processed. The Kernel of OM is has to be tested for stability under various conditions.

Based on the testing and application results of the coupled OM tuning and optimization tasks have to be done:

- Optimization of the query interface for the use by the graphical design tools of the Infrawebs (SWU) system\(^\text{12}\)

- Performance related optimization of the classification system to serve as case based memory applied to the SWU as well as the SAM

- Performance optimization of the of the OM for the case based retrieval\(^\text{13}\)

- Finalizing the Demonstrator

\(^{12}\) Depends on the development of the SWU in WP5

\(^{13}\) Depends on the final development of the CBR Query interface
References

[1] A Lopez, J. Gorroñogoitia; Deliverables: D11.2.3 INFRAWEBS Architecture and Evolution Plan & D11.3.4 IIF Design


[8] Micsik, Kovacs; Deliverable D6.2.2, Realised and Coupled Discovery Component


[10] H Joachim Nern, A.Boyanov, D2.1.1-Specification and Realised Reduced OM


APPENDIX 1 - INTERNAL COMPONENTS OF THE OM

Internal Components of the OM and their interoperation (see [1])

Interaction with internal components:

Internal components for the OM are:

- Data Layer Component;
- Objects Retrieval Component;
- Classification Component;
- Clustering Component;
- Similarity Calculator Component;
- Filtering Component;
- Discovery component;
- Web Service Component;
- GUI Component.

The internal interactions provoked by the OM web methods are described hereafter.

The **SearchForWSMOObjects** and **SearchForSimilarObjects** methods provoke the following internal interactions:

- The Web Service Component invokes a method of the Discovery Component;
- The Discovery Component invokes the Filtering Component to create a feature vector. The input to the filtering component is one of the following:
  - structured query;
  - WSMO object;
  - WSDL objects;
- The Discovery Component invokes the Classification Component to classify the feature vector created in the previous step. The result is a set of centroids – one per classification layer, that best fit the given feature vector;
- The Discovery Component invokes again the Classification Component to retrieve the best classified objects to the given centroids in the previous step;
- The Discovery Component invokes the Similarity Calculator to evaluate the similarity between the input query and the retrieved objects;
- The Discovery Component returns the result set to the Web Service Component;
• The Web Service Component returns the result set to the client.

The **NewOrModifiedObjects** message results in the following internal (and external) communications:

- The Web Service Component invokes the Object Retrieval Component and sends him the new or modified objects identifiers;
- The Object Retrieval Component retrieves the objects (see the next section – Interaction with external component for details);
- The Object Retrieval Component passes the retrieved objects through the Filtering Component to obtain their feature vectors;
- The Object Retrieval Component stores the filtered objects in the data storage through the Data Layer Component;
- The Object Retrieval Component informs the Clustering Component that new objects were stored in the system;
- The Clustering Component stores the information needed for centroids reconfiguration;
- The Clustering Component schedules a centroids reconfiguration depending on the configured OM policy;

The internal interactions may be further developed according the iterative development approach we are following.

*Interaction with external components:*

In the case of new or modified objects stored in the DSWS-R or SIR (**NewOrModifiedObjects** message received) the OM component interacts with these external components to retrieve the new or modified objects.

When the initiator of the message is the DSWS-R component then retrieval is done by invoking one of the following API methods (depending on the object type):

- Goal getGoal(IRI id);
- Ontology getOntology(IRI id);
- WebService getWebService(IRI id).

When the initiator is the SIR component then the interaction is done by invoking the Web Method GetObjects()
APPENDIX 2 - OM ACTIVITY DIAGRAMS
OM Activity Diagrams (see [1])

Figure A2-1: Registration of WSDL files

Figure A2-2: Registration of BPEL files
Figure A2-3: Find Similar WSMO Service

Figure A2-4: Find WSDL files

Figure A2-5: Find WSMO ontology
Figure A2-6: Compose SWS

Figure A2-7: Storage of a WSMO file
Figure A2-8: Publish SWS

Figure A2-9: Notify repository update
APPENDIX 3 - WSML DESCRIPTION OF THE OMSEARCH WEB SERVICE

WSML description of the OMSearch web service

```xml
<?xml version="1.0"?>
<!-- WSDL file version: $Revision: 1.46 $ -->
<definitions name="OMSearch"
    targetNamespace="http://infrawebs.aspasia-systems.de/OM/"
    xmlns:tns="http://infrawebs.aspasia-systems.de/OM/"
    xmlns:xsd1="http://infrawebs.aspasia-systems.de/OM/Types"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
        <xsd:schema elementFormDefault="qualified"
            targetNamespace="http://infrawebs.aspasia-systems.de/OM/Types">
            <xsd:complexType name="Query">
                <xsd:sequence minOccurs="1" maxOccurs="1">
                    <xsd:element name="objectType" type="xsd1:ObjectType" minOccurs="0" maxOccurs="1" default="service"/>
                    <xsd:element name="maxResult" type="xsd:int" minOccurs="0" maxOccurs="1" default="10"/>
                    <xsd:element name="similarityThreshold" type="xsd:float" minOccurs="0" maxOccurs="1" default="0.80"/>
                    <xsd:element name="nfp" type="xsd1:NFP" minOccurs="0" maxOccurs="1"/>
                    <xsd:element name="importedOntologies" type="xsd1:ImportedOntologies" minOccurs="0" maxOccurs="1"/>
                    <xsd:element name="capability" type="xsd1:Capability" minOccurs="0" maxOccurs="1"/>
                    <xsd:element name="wsdlObject" type="xsd1:WsdlObject" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
        <xsd:complexType name="WsdlQuery">
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element name="content" type="xsd1:WeightedKeywordsString" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="metadata" type="xsd1:WeightedKeywordsString" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="maxResult" type="xsd:int" minOccurs="0" maxOccurs="1" default="10"/>
                <xsd:element name="similarityThreshold" type="xsd:float" minOccurs="0" maxOccurs="1" default="0.80"/>
                <xsd:element name="importedOntologies" type="xsd1:ImportedOntologies" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="capability" type="xsd1:Capability" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="wsdlObject" type="xsd1:WsdlObject" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </types>
</definitions>
```
or equal then this threshold will be included in the list with the query results -->
</xsd:element name="similarityThreshold" type="xsd:float" minOccurs="0" maxOccurs="1" default="0.00"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Message">
    <xsd:sequence>
        <xsd:element name="type" type="xsd1:MessageType" minOccurs="0" maxOccurs="1" default="NewOrModifiedObjects"/>
        <xsd:element name="senderURI" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="senderType" type="xsd1:SenderType" minOccurs="0" maxOccurs="1" default="DSWSRepository"/>
        <xsd:element name="objects" type="xsd1:MessageObject" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SearchResult">
    <xsd:sequence>
        <xsd:element name="searchTime" type="xsd:float" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="resultSet" type="xsd1:ResultSet" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ResultSet">
    <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" name="resultElement" type="xsd1:ResultElement" />
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ResultElement">
    <xsd:sequence>
        <!-- OntologySimilarity is float number from 0 to 1 or "undefined"! -->
        <xsd:element name="ontologySimilarity" type="xsd:float"/>
        <!-- LexicalSimilarity is float number from 0 to 1 or "undefined"! -->
        <xsd:element name="lexicalSimilarity" type="xsd:float"/>
        <!-- Pointer to the object description in the repository - identifier -->
        <xsd:element name="identifier" type="xsd:string"/>
        <!-- The type of the object -->
        <xsd:element name="objectType" type="xsd1:ObjectType"/>
        <!-- Metadata -->
        <xsd:element name="annotation" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="NFP">
    <xsd:sequence>
        <xsd:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
        <xsd:element name="content" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ImportedOntologies">
  <xsd:sequence>
    <xsd:element name="ontologyName" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Capability">
  <xsd:sequence>
    <xsd:element name="assumption" type="xsd1:Assumption" minOccurs="0"/>
    <xsd:element name="precondition" type="xsd1:Precondition" minOccurs="0"/>
    <xsd:element name="postcondition" type="xsd1:Postcondition" minOccurs="0"/>
    <xsd:element name="effect" type="xsd1:Effect" minOccurs="0"/>
    <xsd:element name="serviceFunctionality" type="xsd1:ServiceFunctionality" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WsdlObject">
  <xsd:sequence>
    <xsd:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
    <xsd:element name="byExample" type="xsd:boolean" minOccurs="0" maxOccurs="1" default="false"/>
    <xsd:element name="content" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Assumption">
  <xsd:sequence>
    <xsd:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
    <xsd:element name="byExample" type="xsd:boolean" minOccurs="0" maxOccurs="1" default="false"/>
    <xsd:element name="content" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Precondition">
  <xsd:sequence>
    <xsd:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
    <xsd:element name="byExample" type="xsd:boolean" minOccurs="0" maxOccurs="1" default="false"/>
    <xsd:element name="content" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Postcondition">
  <xsd:sequence>
    <xsd:element name="weight" type="xsd:int" minOccurs="0" maxOccurs="1" default="1"/>
    <xsd:element name="byExample" type="xsd:boolean" minOccurs="0" maxOccurs="1" default="false"/>
    <xsd:element name="content" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="added"/>
  <xsd:enumeration value="modified"/>
  <xsd:enumeration value="deleted"/>
  <xsd:enumeration value="unchanged"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="MessageType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="NewOrModifiedObjects"/>
    <xsd:enumeration value="Advertisement"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="SenderId">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="DSWSRepository"/>
    <xsd:enumeration value="SIR"/>
    <xsd:enumeration value="Registry"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- Messages used in service operations -->
<message name="SearchForWSMOObjectsInput">
  <part name="query" type="xsd1:Query"/>
</message>

<message name="SearchForWSMOObjectsOutput">
  <part name="return" type="xsd1:SearchResult"/>
</message>

<message name="SearchForWSMOObjectsFastInput">
  <part name="query" type="xsd1:Query"/>
</message>

<message name="SearchForWSMOObjectsFastOutput">
  <part name="return" type="xsd1:SearchResult"/>
</message>

<message name="SearchForSimilarObjectsInput">
  <part name="obj" type="xsd:string"/>
  <part name="objectType" type="xsd1:ObjectType"/>
  <part name="maxResults" type="xsd:int"/>
</message>
<message name="SearchForSimilarObjectsOutput">
  <part name="return" type="xsd:SearchResult"/>
</message>

<message name="SearchForSimilarObjectsFastInput">
  <part name="obj" type="xsd:string"/>
  <part name="objectType" type="xsd:ObjectType"/>
  <part name="maxResults" type="xsd:int"/>
</message>

<message name="SearchForSimilarObjectsFastOutput">
  <part name="return" type="xsd:SearchResult"/>
</message>

<message name="SearchForWSDLObjectsInput">
  <part name="wsdlQuery" type="xsd:WsdlQuery"/>
</message>

<message name="SearchForWSDLObjectsOutput">
  <part name="searchResult" type="xsd:SearchResult"/>
</message>

<message name="NewOrModifiedObjectsInput">
  <part name="message" type="xsd:Message"/>
</message>

<message name="NewOrModifiedObjectsOutput">
  <part name="success" type="xsd:boolean"/>
</message>

<message name="IsAliveInput">
</message>

<message name="IsAliveOutput">
  <part name="isAlive" type="xsd:boolean"/>
</message>

<message name="ShutDownInput">
</message>

<message name="ShutDownOutput">
  <part name="isShuttingDown" type="xsd:boolean"/>
</message>

<message name="StartInput">
</message>
<message name="StartOutput">
    <part name="started" type="xsd:boolean"/>
</message>

<!-- Port for OMSearch Service -->
<portType name="OMSearchPort">
    <operation name="SearchForWSMOObjects">
        <input message="tns:SearchForWSMOObjectsInput"/>
        <output message="tns:SearchForWSMOObjectsOutput"/>
    </operation>

    <operation name="SearchForWSMOObjectsFast">
        <input message="tns:SearchForWSMOObjectsFastInput"/>
        <output message="tns:SearchForWSMOObjectsFastOutput"/>
    </operation>

    <operation name="SearchForSimilarObjects">
        <input message="tns:SearchForSimilarObjectsInput"/>
        <output message="tns:SearchForSimilarObjectsOutput"/>
    </operation>

    <operation name="SearchForSimilarObjectsFast">
        <input message="tns:SearchForSimilarObjectsFastInput"/>
        <output message="tns:SearchForSimilarObjectsFastOutput"/>
    </operation>

    <operation name="SearchForWSDLObjects">
        <input message="tns:SearchForWSDLObjectsInput"/>
        <output message="tns:SearchForWSDLObjectsOutput"/>
    </operation>

    <operation name="NewOrModifiedObjects">
        <input message="tns:NewOrModifiedObjectsInput"/>
        <output message="tns:NewOrModifiedObjectsOutput"/>
    </operation>
</portType>
<operation name="IsAlive">
    <input message="tns:IsAliveInput"/>
    <output message="tns:IsAliveOutput"/>
</operation>

<operation name="ShutDown">
    <input message="tns:ShutDownInput"/>
    <output message="tns:ShutDownOutput"/>
</operation>

<operation name="Start">
    <input message="tns:StartInput"/>
    <output message="tns:StartOutput"/>
</operation>

<operation name="ProcessModifiedWSMOObjects">
    <input message="tns:ProcessModifiedWSMOObjectsInput"/>
    <output message="tns:ProcessModifiedWSMOObjectsOutput"/>
</operation>

<binding name="OMSearchSoapBinding" type="tns:OMSearchPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="SearchForWSMOObjects">
        <soap:operation
            soapAction="http://infrawebs.aspasia-systems.de/OM/SearchForWSMOObjects"/>
        <input>
            <soap:body use="encoded"
                namespace="http://infrawebs.aspasia-systems.de/OM/"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
        <output>
            <soap:body use="encoded"
                namespace="http://infrawebs.aspasia-systems.de/OM/"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
    </operation>
    <operation name="SearchForWSMOObjectsFast">
        <soap:operation
            soapAction="http://infrawebs.aspasia-systems.de/OM/SearchForWSMOObjectsFast"/>
        <input>
            <soap:body use="encoded"
                namespace="http://infrawebs.aspasia-systems.de/OM/"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
        <output>
            <soap:body use="encoded"
                namespace="http://infrawebs.aspasia-systems.de/OM/"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
    </operation>
</binding>
<operation name="SearchForSimilarObjects">
  <soap:operation
    soapAction="http://infracom.aspinas.asianas.de/OM/SearchForSimilarObjects"/>

  <input>
    <soap:body use="encoded"
      namespace="http://infracom.aspinas.asianas.de/OM/
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>

  <output>
    <soap:body use="encoded"
      namespace="http://infracom.aspinas.asianas.de/OM/
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
</operation>

<operation name="SearchForSimilarObjectsFast">
  <soap:operation
    soapAction="http://infracom.aspinas.asianas.de/OM/SearchForSimilarObjectsFast"/>

  <input>
    <soap:body use="encoded"
      namespace="http://infracom.aspinas.asianas.de/OM/
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>

  <output>
    <soap:body use="encoded"
      namespace="http://infracom.aspinas.asianas.de/OM/
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
</operation>

<operation name="SearchForWSDLObjects">
  <soap:operation
    soapAction="http://infracom.aspinas.asianas.de/OM/SearchForWSDLObjectsFast"/>

  <input>
    <soap:body use="encoded"
      namespace="http://infracom.aspinas.asianas.de/OM/
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>

  <output>
    <soap:body use="encoded"
      namespace="http://infracom.aspinas.asianas.de/OM/
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
</operation>
<binding>
  <input>
    <soap:body use="encoded"
      namespace="http://infrawebs.aspasia-systems.de/OM/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </input>
  <output>
    <soap:body use="encoded"
      namespace="http://infrawebs.aspasia-systems.de/OM/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </output>
</binding>

<operation name="ShutDown">
  <soap:operation
    soapAction="http://infrawebs.aspasia-systems.de/OM/ShutDown"/>
  <input>
    <soap:body use="encoded"
      namespace="http://infrawebs.aspasia-systems.de/OM/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </input>
  <output>
    <soap:body use="encoded"
      namespace="http://infrawebs.aspasia-systems.de/OM/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </output>
</operation>

<operation name="Start">
  <soap:operation
    soapAction="http://infrawebs.aspasia-systems.de/OM/Start"/>
  <input>
    <soap:body use="encoded"
      namespace="http://infrawebs.aspasia-systems.de/OM/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </input>
  <output>
    <soap:body use="encoded"
      namespace="http://infrawebs.aspasia-systems.de/OM/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </output>
</operation>

<!--
<operation name="ProcessModifiedWSMOObjects">
  <soap:operation
    soapAction="http://infrawebs.aspasia-systems.de/OM/ProcessModifiedWSMOObjects"/>
  <input>
    <soap:body use="encoded"
      namespace="http://infrawebs.aspasia-systems.de/OM/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </input>
  <output>
    <soap:body use="encoded"
      namespace="http://infrawebs.aspasia-systems.de/OM/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </output>
</operation>
-->

</binding>
<!-- OMSearch Web Service -->
<service name="Search">
  <documentation>
    Web service exposing the OM features to the other Infraweb modules
  </documentation>
  <port name="OMSearchPort" binding="tns:OMSearchSoapBinding">
    <soap:address location="http://82.103.112.243/OMsearch/OMSearch.asmx"/>
  </port>
</service>
</definitions>
APPENDIX 4 - FCS-PAPER - IPMU 2006

FCS-Paper - IPMU 2006:
G. Andonova, G. Agre, H.-J. Nern, A. Boyanov

"Fuzzy Concept Set Based Organizational Memory as a Quasi Non-Semantic Component within the INFRAWEBS Framework."

Fuzzy Concept Set Based Organizational Memory as a Quasi Non-Semantic Component within the INFRAWEBS Framework

Galya Andonova  
Active Solutions Ltd.  
Sofia, Bulgaria  
galya.andonova@activesolutions.bg

Gennady Agre  
Institute of Information Technologies - BAS,  
Sofia, Bulgaria  
agre@inf.bas.bg

H Joachim Nern  
Aspasia Systems Ltd.  
Duesseldorf,  
Germany  
nern@aspasia-systems.de

Andrei Boyanov  
Active Solutions Ltd.  
Sofia, Bulgaria  
andrei.boyanov@activesolutions.bg

Abstract

The intention of this article is to describe the use of fuzzy logic techniques applied to an Organizational Memory (OM) – a module of the INFRAWEBS framework [1] dedicated to the collection, organization, refinement, and distribution of entity specific knowledge. The module is based on the new streams and approaches in the area of Fuzzy Concept Set (FCS) Modeling, especially Fuzzy Concept Matching (FCM) [2]. The paper gives an overview about the INFRAWEBS conceptual model as well as presents the automatic object feature vector generation, objects classification and clustering mechanisms, as well as techniques for retrieving similar objects and ontology based semantic objects descriptions.

Keywords: Semantic Web Service, WSMO, WSDL, WSML, Ontology, Fuzzy matching, Classification, Clustering.

1 INFRAWEBS Framework

The current technology based on SOAP, WSDL and UDDI addresses only syntactical aspects of a Web service. On this basis just “rigid” (static) services can be designed and invoked. Rigid and non-dynamical means, that they cannot adapt and reflect changes (or disturbances in terms of automatic control) of the action environment without human intervention. Web services based on above mentioned technologies are not “really” automatically driven – their action cycles depend mainly on semi-automatic structures, due to missing closed loop features.

Accordingly new streams and project initiatives are developed to increase the degree of automation and the flexibility of service providing and service requesting activities by applying soft computing as well as Fuzzy Logic approaches [3, 5, 6, 7, 8, 9, 12].

The INFRAWEBS framework represents an application-oriented software “tool set” for creating, maintaining and executing WSMO-based [10] Semantic Web Services within the whole life cycle. The main users of the INFRAWEBS will be Web service providers as well as semantic Web service application developers, who register their applications into the system. The INFRAWEBS provides a loosely-coupled set of adaptable system components and tools to analyze design and maintain Semantic Web Services (SWS) in the whole service life cycle: discovery, selection, execution and monitoring. The specific approach follows strictly the bottom up paradigm - it combines and couples knowledge management issues (representing the basis) with SWS issues (representing the top of service advertising and offering environments).

The open platform developed within the INFRAWEBS project consists of coupled and linked Semantic Web Units (SWU), whereby each unit provides tools and system components to analyze, design and maintain Web services realized as semantic Web services within the whole life cycle.

The overall design of the INFRAWEBS (Figure 1) is structured in three main layers:

1) Knowledge management layer for handling service related knowledge artifacts realized as an organizational memory coupled to semantic information routing components (OM & SIR) [1];
2) Service development layer for creating and maintaining SWS embedded in a semantic based interoperable middleware, consisting of Semantic Web Service Designer & Composer, Distributed Semantic Web Service Registries, and an agent based discovery module (Semantic Web Service Unit - SWU);

3) Service deployment layer for the execution and monitoring of Semantic Web Services exploiting closed loop feedback information (Quality of Service brokering) provided for monitoring and execution issues.

![Figure 1: INFRAWEBS layered structure](image)

2 INFRAWEBS Conceptual Model

The conceptual model of the INFRAWEBS (Figure 2) is aimed to serve the following categories of users:

- **Web Service Provider** – a provider of an existing Web service, who would like to convert it to the Semantic Web service and to publish it.

- **Web Service Broker** – an entity, who would like to create and publish a Semantic Web Service with some desired functionality via composition of several existing Semantic Web services.

- **Web Service Application Provider** – an entity, who would like to design an own application based on Semantic Web Service Technology.

- **Web Service Application User** – an “ordinary” end-user of a Web Service Application, who would like to use INFRAWEBS for retrieving a Web Service able to satisfy the request (goal).

The INFRAWEBS Conceptual Model consists of two main elements: the INFRAWEBS Semantic Web Unit and the INFRAWEBS Environment. The INFRAWEBS Semantic Web Unit provides users with:

- **Information structures** for effective discovering, storing and retrieving both semantic and non-semantic information needed for creating and maintaining Semantic Web Services.

- **Tools** for creating and maintaining Semantic Web Services and SWS Applications.

- **Library of Methods** used for creating and maintaining Semantic Web Services.

The INFRAWEBS Environment provides means for communicating with different kinds of INFRAWEBS users and other INFRAWEBS Semantic Web Units as well as for executing SWS ensuring security and privacy of these operations.

![Figure 2: The INFRAWEBS Conceptual Model](image)
as well as other data e.g. graphical models of SWS and “natural language” templates for WSMO Goals.

**Tools for creating and maintaining both semantic and non-semantic data:**

- **CBR based Service Designer** is aimed at designing a WSMO-based Semantic Web service from an existing non-semantic Web service.
- **CBR based Service Composer** is aimed at creating a Semantic Web service through composition of existing WSMO-based semantic Web services.
- **CBR based Goal Editor** is aimed at creating predefined WSMO-based goals and their “natural language” templates needed for designing SWS-based applications.
- **CBR based Recommender tool** is a similarity-based tool facilitating operation of all INFRAWEBS “semantic-based” tools by utilizing “past experience”. Non-semantic data stored in OM is used as the problem description for determining the most similar solution (SWS or its graphical model) to the current problem.

**Methods used during problem-solving**

- Logic-based discovery.
- Application-specific decision-support methods used for service composition, compensation, monitoring etc.
- Ontology key words based discovery
- Several methods for calculating similarity and/or assignments - structural, linguistic, statistic etc.

Semantic and non-semantic components of INFRAWEBS are interconnected by a **Semantic Information Router (SIR)**, which is responsible for:

- Locating all resources needed for problem solving either in the local SWU or outside.
- Creation of non-semantic content (knowledge objects) by means of the semantic content stored in the DWSW-R.
- Creating an effective system of indexes allowing fast communication between semantic and non-semantic modules of SWU.

The objective of this paper is to present main design principles of the INFRAWEBS Organizational Memory (FCS-OM), which uses the Fuzzy Concept Set modeling to solve the following tasks:

- Classification and retrieval of semantic and non-semantic knowledge;
- Implementation of a case-based memory providing solutions for natural language and ontology-based queries;
- Non-semantic mapping of semantic web services to the given requester goals as an implementation of the first step of the service discovery process;
- Provision of similarity calculation methods;

**3 FCS-OM Design and Concepts**

The FCS-OM represents a similarity based organizational memory and a case based recommender tool. The component classifies the system knowledge objects using fuzzy logic techniques as well as statistical and linguistic algorithms as fuzzy matching functions.

### 3.1 Main Elements

The system knowledge is represented by semantic and non-semantic knowledge objects. Semantic knowledge objects are the WSML descriptions [17] of semantic web services, ontologies, goals etc. WSDL files are non-semantic system knowledge objects containing non-semantic web services descriptions.

Knowledge objects are classified to a given pre-configured taxonomy composed out of classification subjects. Each knowledge object of the organizational memory is categorized by similarity- (and relationship-) coefficients determining the relation between objects and subjects regarding the given taxonomy.

FCS-OM module uses different types of system dictionaries - knowledge domain keywords, generic keywords, abbreviations, stop terms and synonyms. These dictionaries are used by the algorithms for creating a (meta data) feature vector representation of knowledge objects and “centroids”, whereby the so called “centroids” are formalized meta data representations of subjects. The (meta data) centroids are synthetically generated objects representing the quasi centre of the set of objects.
assigned to a subject (see following section). The module maintains also a list of keywords recommendations that are extracted automatically out of each new object stored in the system. Accordingly each keyword proposition is processed and added to one of the system dictionaries.

The FCS-OM supports multi-dimensional classification of knowledge objects. For this purpose multiple dictionary sets are also supported:

- Ontology based dictionaries
- Natural language dictionaries

A fuzzy mapping between these two kinds of dictionaries is maintained by the system and is used for retrieving semantic information by a natural language query.

Centroids are the representatives of each set of objects. Each centroid is composed of centroid feature vector (CFV) and mapped to a classification subject given by the system taxonomy. Multiple layers of centroids are maintained by the component to allow the multi-dimensional classification of objects.

Every knowledge object is represented by an object feature vector (OFV). An OFV is a list of weighted keywords that is created automatically by a process called filtering. Multiple layers of OFVs are supported by the system, whereas different methods in each layer are used.

The component combines semantic and non-semantic classification of objects. Additionally the semantic objects are classified by different semantic criteria. This imposes the necessity of multiple classification layers, where knowledge objects are represented by different OFVs and classified to the layers centroids.

3.2 Generation of Centroids and OFVs

The FCS-OM supports two types of centroids:
- centroids based on natural language words;
- centroids based on ontology terms.

The centroids based on natural language words are generated by a clustering process. These centroids are dynamically reconfigured by the system in the case a "new" object is stored in the objects' collection.

The centroids based on ontology terms are automatically created from the ontologies stored in the DSWS-R component of the INFRAWEBS system. They are reconfigured automatically in the case a new ontology is stored.

The knowledge object feature vector generation is performed automatically during the process of adding the object to the objects' collection. The feature vector creation is achieved by a filtering process. The filtering of generic objects consists of automatic extraction of object's metadata and relevant keywords. The OFVs generated out of generic knowledge objects are non-semantic object feature vectors (NS-OFV) The metadata processing considers the meaning of different metadata fields. Keywords from different metadata fields are weighted in different way [13].

The filtering and normalization is performed as a fuzzy matching procedure. Subsequently the generated feature vector undergoes a classification process, which is performed using statistical similarity and assignment approaches for determining relationship functions among the objects themselves and pretended subject term vectors (centroids, representing the subjects), to which the objects are assigned and allocated.

A further applied fuzzy matching process using the predefined dictionaries of standard keywords and knowledge-domain terms extracts and weights implicitly terms and normalize the term vector to the given sets of standard keywords and knowledge-domain terms.

Creation of a semantic objects feature vector (S-OFV) is performed by extracting ontology concepts out of WSML files. The Semantic objects feature vector consists of weighted ontology concepts. The system dictionaries used for these feature vectors are populated with ontology concepts too.

3.3 Similarity and Assignment Function Calculations

Similarity calculation methods are used as fuzzy matching functions. Several methods for similarity calculation are provided by the FCS-OM:

- Statistical methods for calculation of the similarity between two objects;
- Linguistic methods for calculating the similarity between terms;
• Combined methods for complex similarity calculations between ontology based and natural language based OFV's.

The result of the assignment function and similarity calculation yields in a similarity resp. assignment function matrix, which couples the processed objects by similarity coefficients (Figure 3).

Figure 3: Calculation of a knowledge content object similarity and assignment matrix

3.4 Objects clustering

A three stage bottom up clustering is processed by using stepwise the MinMax-Algorithm and processing the assignment matrix of the objects given in the collection.

In Figure 4a the membership function $\mu$ is given over the centroids of the median sets $ms$ (R, Q, I, J .... G, Z, N; 2nd layer in the subject tree). $FS_{med}$ gives the fuzzy root concept set, consisting of fuzzy median concept sets, whereby $\mu_{med}(ms)$ gives the assignment function of the fuzzy median concept centroids. Marked in bold is the assignment function $\mu_{ms}$ of the fuzzy concept median set $FS_{K}$. (K).

The set $FS_{k}$ marked in bold with the assignment function $\mu_{ms}(ms)$ is dissolved in Figure 4b as a set with the function $\mu$ (bs) dependent on the fuzzy concept bottom sets bs. Marked in dark bold is the element set $FS_{bs}$ with the assignment function $\mu_{bs}(e)$ dependent on the elements (knowledge content object, concept) in the bottom set $K_{bs}$.

In Figure 4c the element set $FS_{bs}$ (marked in bold) with the assignment function $\mu_{bs}(e)$ is dissolved over the elements of the bottom sets. Marked in dark bold is exemplarily the word set $FS_{wbs}$ with the assignment function $\mu_{wbs}$ dependent on the words in the element set.

The word set $FS_{wbs}$ (marked in dark bold) with the assignment function $\mu_{wbs}$ is dissolved in Figure 4d over the words of the elements. The assignment coefficients of the singular words (Kol.1.6, Kol.1.6, ..., Kol.1.6.9) to the given element Kol.1.6 is equal to the discrete points of $\mu_{wbs}$.

The visualizations in Figure 4b, c, d illustrate the capability of the OM to determine the overall assignment (or membership) of a single "word" to the overall knowledge stored in the OM – the capability to identify the degree of membership of a single word to the overall knowledge (local similarity / global similarity – local membership / global membership). Defined vice versa: the OM has the capability to identify a global fuzzy root set of concepts (global knowledge), consisting of fuzzy median sets of concepts (centroids), consisting of fuzzy sets of elements, consisting of sets of (atomic) words.

3.5 Objects classification

Objects are classified to subjects of the classification subject’s tree based on processing the assignment functions. Each subject is characterized and represented by centroids, whereby the centroids are formalized as feature vectors (CFV), which are structural identical with the object feature vectors (OFV).

During the classification procedure every similarity resp. assignment coefficient between an object and a given centroid is classified in one of three similarity sets – min, middle or max. The min set contains objects with minimal similarity coefficients between them and the subject, the max set contains objects with maximum similarity coefficients between them and the subject and the middle set contains objects with middle coefficients.

4 Using FCS-OM as a Case-based Memory

The FCS-OM acts as a case based memory and recommender tool in the context of the INFRAWEBS system [18]. The component uses the objects classification and clustering for determining appropriate objects for different kinds of case-based queries.

4.1 Query by imported ontologies

This is the simplest type of queries, which requests for existing services described by the same (or almost the same, "quite similar") set of (imported) ontologies.
Figure 4a, b, c, d: Three stage bottom up clustering of concept sets yielding in: root set, medion sets, element sets, word sets.
The rationality behind this is an assumption that services, which used the same set of ontologies, are likely to belong to the same domain. Practically such an option is equivalent to browsing capabilities of available semantic services from the same problem domain and may be useful for receiving some initial ideas of how the desired service capability should look like. It is not expected to receive a precise selectivity but some kind of fuzzy one from such type of the query.

4.2 “Natural language” queries

“Natural language” queries are intended for searching existing semantic services, which descriptions contain words (terms) specified in the query. Since natural language is used mainly for describing nonfunctional properties of a semantic service (e.g., title, publisher, description, etc.) such a query will be matched against such service properties. The queries may be unstructured or structured. In the first case matching is performed against all nonfunctional properties of the service, while in the second case the similarity is evaluated only using nonfunctional properties specified in the query. For example, it can be requested for the “similar” services created by a concrete organization (service publisher) or written in a concrete natural language or even created by a concrete person (service nonfunctional property “contributor”).

In order to guarantee the compatibility of the retrieved service with the already selected set of ontologies, such “natural language” queries are automatically considered in conjunction with the first type of queries (queries by a set of ontologies).

4.3 Ontology-based queries

The mere a-priori information available to the user before starting to create the description of the service’s axiom is an initial set of ontologies, which was selected as the most appropriate for semantic description of the Web service during the generation process of the service grounding. So a single way to represent more clearly the “meaning” of the new axiom is to describe it as a set of concepts and relations that user thinks to include in its description. Since an axiom can exist only as a named part of a Semantic service description, the meaning of such a query is to find a set of existing Semantic Web services, which capability descriptions contain the set of ontological concepts specified in the query.

4.3.1 Unstructured queries

Even in this case it is possible to formulate a set of queries with a different meaning:

- Query for a semantic service, which capability description as a whole contains (or is similar to) a specified set of ontological concepts.
- Query for a semantic service containing one or several axioms, which descriptions separately contain (or similar to) a specified set of ontological concepts.

In the first case the similarity is measured based on the overall ontology concepts accumulated from all axioms participated in the service capability description. In other words the query is matched against a compound axiom constructed by merging all service capability axioms. The most similar service will be a service, which capability axioms have the highest average similarity (overall similarity) with the query.

In the second case the query is matched against each capability axiom (individually). The most similar service will be the one containing an axiom, which has the highest similarity with the query.

4.3.2 Structured queries

The next natural step is, to allow the user to construct structured queries. Such queries are specified not only by the ontological concepts the service capability description should have, but also considering which part of such a description (i.e., postconditions, assumption etc.) should have which set of ontological words.

For example, it will be possible to retrieve an existing semantic service, whereas the preconditions are the most similar to the set of ontological words specified in the query; or to find such a service which preconditions are similar to one ontological word set and which postconditions are similar to another set of such words defined in the query. In such case the most similar service will be the service with the highest average aggregated similarity (overall similarity) to the structural query.

5 INFRAWEBS Application

The INFRAWEBS Consortium is developing a semantic application performing a Frequent
Flyer Program (FFP) [19] in which the customers can create and reuse travel packages. The application is built upon a Service Oriented Architecture, accessing, discovering, composing and invoking Semantic Web Services for the management of the Travel Packages. The composition of semantic services is driven by Choreography, using the Web Service Modeling Ontology (WSMO see http://www.wsmo.org) as a framework to describe both the service capability and the service behavior. The prototype implementation of the FFP is expected to be realized until the end of 2006.

Acknowledgement

This work is supported by the EC funded IST project INFRAWEBS (IST FP6-511723) within the framework FP 6.

6 References


