Specification of the Architecture

Version 1.0

Thomas Tikwinski (FHG/IAIS), Carsten Rosche (FHG/IAIS),
George Paliouras (NCSR), Alfio Ferrara (UniMi), Atila Kaya (TUHH),
Vasileios Papastathis (CERTH)

Distribution: Restricted

BOEMIE
Bootstrapping Ontology Evolution with Multimedia Information Extraction

National Centre for Scientific Research “Demokritos” (NCSR)
Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. (FHG/IMK)
University of Milano (UniMi)
Centre for Research and Technology Hellas (CERTH)
Hamburg University of Technology (TUHH)
Tele Atlas (TA)

FP6-027538 D5.4
28/02/2007
<table>
<thead>
<tr>
<th>Security (distribution level)</th>
<th>Restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contractual date of delivery</td>
<td>M12</td>
</tr>
<tr>
<td>Actual date of delivery</td>
<td>30.04.2007</td>
</tr>
<tr>
<td>Deliverable number</td>
<td>D5.4</td>
</tr>
<tr>
<td>Deliverable name</td>
<td>Specification of the architecture</td>
</tr>
<tr>
<td>Type</td>
<td>Report</td>
</tr>
<tr>
<td>Status &amp; version</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>Number of pages</td>
<td>45</td>
</tr>
<tr>
<td>WP contributing to the document</td>
<td>WP5</td>
</tr>
<tr>
<td>WP / Task responsible</td>
<td>FHG/IAIS</td>
</tr>
<tr>
<td>Other contributors</td>
<td>NCSR, CERTH, TUHH, UniMi, TeleAtlas</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Thomas Tikwinski, George Paliouras</td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>George Petasis, Alfio Ferrara, George Paliouras, Sergios Petridis</td>
</tr>
<tr>
<td>EC Project Officer</td>
<td>Johann Hagman</td>
</tr>
<tr>
<td>Keywords</td>
<td>Prototype, Architecture, Specification</td>
</tr>
</tbody>
</table>

**Abstract (for dissemination)**

This document specifies the general architecture of the BOEMIE prototype system. It elaborates on the specification approach used for the architecture, on suitable modeling techniques, technical platforms and integration aspects.
Content

1 Introduction .................................................................................................................................. 5
  1.1 Executive Summary .................................................................................................................. 5
  1.2 Content of the document ......................................................................................................... 5
  1.3 Scope ..................................................................................................................................... 5
  1.4 Document Status ...................................................................................................................... 5
2 Defining the BOEMIE architecture ............................................................................................. 6
  2.1 Challenges ............................................................................................................................... 6
  2.2 Innovation ............................................................................................................................... 6
  2.3 Overall System Design Approach ........................................................................................... 7
  2.4 Modeling ................................................................................................................................. 7
  2.5 Implementation Strategy ......................................................................................................... 8
  2.6 Risk Handling ......................................................................................................................... 8
3 Case Studies .................................................................................................................................. 10
  3.1 Integration with the LIVE project ............................................................................................ 10
  3.2 Cooperation with content providers ....................................................................................... 10
4 Use Cases .................................................................................................................................... 11
  4.1 The System Operator ............................................................................................................... 11
  4.2 The Domain Expert ................................................................................................................ 13
  4.3 The End User ......................................................................................................................... 15
5 Activities and Processes ............................................................................................................... 20
  5.1 Adding new Content ................................................................................................................ 20
  5.2 Bootstrapping ......................................................................................................................... 22
  5.3 Monitor System Behavior ........................................................................................................ 24
  5.4 Maintain System Components ............................................................................................... 25
  5.5 Support Ontology Population ............................................................................................... 27
  5.6 Support Ontology Enrichment ............................................................................................... 29
  5.7 Support Ontology Coordination ............................................................................................ 30
  5.8 Query Content ........................................................................................................................ 31
  5.9 Browse Content / Suggested Reading ..................................................................................... 32
  5.10 Content Location ................................................................................................................... 34
6 System architecture .................................................................................................................... 35
  6.1 High level architecture ............................................................................................................. 35
  6.2 Building Blocks ....................................................................................................................... 36
  6.3 Component Architecture ......................................................................................................... 39
  6.4 Communication ........................................................................................................................ 42
7 Integration Platform ................................................................................................................... 43
  7.1 Requirements .......................................................................................................................... 43
  7.2 Technical survey ...................................................................................................................... 43
  7.3 Results .................................................................................................................................... 44
  7.4 Web Services Infrastructure .................................................................................................... 44
Illustrations

Illustration 1: System operator use cases ................................................................. 12
Illustration 2: Domain expert use cases ................................................................. 13
Illustration 3: End user use cases ............................................................ 15
Illustration 4: Web Site of the IAAF, with suggested-reading links added by BOEMIE 17
Illustration 5: Suggested Reading Overlays: The user may choose to follow the links or continue normal browsing ................................................................. 18
Illustration 6: Example of locating content on a map: The green arrow shows the location of the content in the browser window. Colored circles indicate other content with close geographic relation to the current content... 19
Illustration 7: Activity Diagram: Adding content to the system (System Operator) ................................................................. 20
Illustration 8: Activity Diagram: Bootstrapping ........................................................ 22
Illustration 9: Activity Diagram: Monitor System Behavior ...................................... 24
Illustration 10: Activity Diagram: Maintain System Components (System Operator) ................. 25
Illustration 11: Activity Diagram: Support Ontology Population (Domain Expert) 27
Illustration 12: Activity Diagram: Support Ontology Enrichment (Domain Expert) 29
Illustration 13: Activity Diagram: Support Ontology Coordination (Domain Expert) 30
Illustration 14: Activity Diagram: Query Content (End User) ...................................... 31
Illustration 15: Activity Diagram: Browsing content with BOEMIE suggestions 32
Illustration 16: Activity Diagram: Content Location (End User) ................................. 34
Illustration 17: Block diagram: BOEMIE core system ............................................. 35
Illustration 18: Component Diagram: First Level System Architecture (see next page for large version) 39
1 Introduction

1.1 Executive Summary

For the various tools and frameworks developed in the BOEMIE project to be integrated into a prototype, a state-of-the-art open architecture is required. It must be able to support the bootstrapping process and related prototype use cases as well as allow for straightforward integration with existing systems of professional users. Based on web services, this document specifies an appropriate architecture for the development of the two prototypes.

1.2 Content of the document

This document contains the system architecture specification for the overall prototype system to be built in the BOEMIE research project. The focus of the BOEMIE system is the automatic semantical analysis of multimedia content assets according to ontologies and the automatic population and evolution of these ontologies. Using this incremental knowledge gain to improve the system performance in terms of recognition of concepts and knowledge extraction is referred to as the Bootstraping Process.

BOEMIE uses three types of ontologies: a multimedia ontology which describes the structure of multimedia content (scenes, cuts, commentary, ...), a domain ontology which contains knowledge about the selected application domain, and a geographic ontology which contains additional knowledge about the locations used in the BOEMIE project.

The system to be built in the BOEMIE project is a semantic multimedia analysis and ontology evolution system. It extracts low-level features from multimedia content and uses them to detect instances of concepts of a multimedia-enriched domain ontology and strives to further interpret the detected (mid-level) concept instances, both per modality and from all modalities combined, to recognize higher-level concepts from the domain ontology.

The system uses state-of-the-art reasoning approaches to deduce further knowledge from mid-level concept instances and geographical relations, and populates the corresponding domain ontology instances with the extracted concept instances. Where applicable, the ontology is automatically extended with new instances and concepts. User clients will link multimedia, domain and geographic ontologies and allow the end user to get access to the acquired knowledge.

1.3 Scope

In this document, we cover the overall system architecture of the BOEMIE integrated prototypes. As written in the Technical Annex p 61ff, an open architecture is envisioned for the integration work. The document discusses the advantages and disadvantages of this approach and collects the requirements of the BOEMIE system to verify that an open architecture is the best approach for the purpose of this integration task. It then studies the use cases covered by the system and decomposes them step by step to derive activities, actors and components, and finally the component structure. In further sections, the document discusses the technical aspects of the integration, such as technical platforms, programming languages, communication means and specification languages.

1.4 Document Status

This document is Version 1.0, it replaces draft version 0.7 final. The document is stable, it has undergone Quality Assurance and is filed as Deliverable D5.4.
2 Defining the BOEMIE architecture

The BOEMIE prototype is going to integrate all software components developed by the partners within the scope of this project plus several further components to implement the bootstrapping process. Furthermore, it implements a set of use cases which demonstrate the practical usability of the system. From the beginning, the project team has envisioned an open architecture to be the foundation for the prototype. The term “open” as it is used in this document generally means “not restricted”, or at least “as little restricted as possible”. There are two sides to this approach: On the input side, open tools and technologies are used wherever possible. All specifications are made using open standards, where integration and communication technologies are used, technologies based on open standards are preferred, and where possible open source implementations of these technologies are applied. On the output side, the architecture itself is designed to facilitate the interaction with the system, making the prototype as open as possible for the integration with further components and other systems. This section discusses the challenges the prototype development faces, the overall system architecture approach and the practical aspects of the architecture definition. Throughout the section, the open architecture approach is explained in more detail.

2.1 Challenges

From a system engineering point of view, the BOEMIE prototype poses a challenging task. It is an experimental system, implementing the bootstrapping process, and being comprised of existing tools as well as components specifically developed for the system. It is likely that the system structure and/or behavior will need to be adapted to the findings from the test runs of this process. At the same time, the list of use cases of the system is not necessarily complete at the time of specification of the system architecture. New use cases may be identified during experimentation with the process, and the project team may decide to change the prototype to integrate them. The system design therefore needs to be flexible to a very high degree to support these changes. Furthermore, the system is envisioned to be integrated into an existing infrastructure of a professional user, service provider or broadcaster as well as with other projects for case studies. Hence the design also needs clear and stable interfaces towards the infrastructure it is integrated with.

As described in the Technical Annex, these requirements call for an open architecture approach which provides a stable framework for integration both to the outward and to the inward part of the system while at the same time being extremely flexible to allow for the system to evolve with growing knowledge about the entire process. The following section sketches the overall design approach taken to cope with these challenges.

2.2 Innovation

When talking about innovation aspects of the system architecture, we need to clarify up-front that the main innovation in the BOEMIE system is not in its architecture. The role of the architecture is to bring together the research results from other work packages and integrate them in a stable prototype system. The architecture has to deal with innovation rather than introduce innovative aspects itself. Nevertheless, there is are two innovative aspects covered by this document:

The first aspect is concerned with the bootstrapping process and the related use cases. As set forth in the previous section, the design of the bootstrapping process is an ongoing research topic, the system architecture has to cope with a task which is at least partly unknown at design time. The bootstrapping-related use cases are therefore tentative and described as we expect them to be. Some

---

1 An open standard is understood in this context as a specification developed by an open consortium and set by an international standards body from where the specification is available for use and implementation free of charge.
may change or become obsolete, others may prove not feasible as expected (or at all), new use cases may need to be introduced. The architecture must be flexible enough to support this.

The second innovative aspect is concerned with the two use cases about suggesting content according to the users focus of interest and with the locating of content on maps. These use cases leverage the ontology built by the BOEMIE system to provide the end user with innovative ways of using the content analyzed by the prototype in conjunction with knowledge from the system ontology.

2.3 Overall System Design Approach

The system to be built must conform to considerable requirements with respect to openness and flexibility. During experiments with the bootstrapping process, requirements to change the system structure or behavior may become visible. This calls for an architecture approach that can cope with this kind of flexibility requirements. The approach presented here relies on the assumption that some parts of the system are likely to change while other parts should remain mostly unchanged throughout the software life cycle. It therefore strives to build a lightweight, stable application framework that provides clear concepts for the building and extension of application structure and behavior, while not making any assumptions with respect to the actual shape and functionality of the application.

The system design chosen for the BOEMIE prototype follows a set of proved design patterns, such as Model-View-Controller for communication or Command Objects for processes. It encapsulates as much of the system structure and behavior in generic components and interfaces as possible. Components implementing these interfaces or deriving from the generic ones can be plugged into the system without much effort.

From the structural point of view, the only two assumptions taken in the system design are that there is

a) exactly one central controller component, called Application Logic, with a well-known interface to the functions it provides, and

b) a common interface from the application logic to all other components of the system.

The dynamic aspects of the system are encapsulated in processes which are implemented as process components that are in turn plugged into the application logic. Each process component instance represents a running process in the system. Processes are started and controlled by external events which are received as messages by the application logic. The processes interact with the rest of the system by themselves sending and receiving messages to and from other application components through the application logic's interface.

This connection pattern leads to a star-shaped communication network with the application logic acting as a hub. The application logic has a 1:n relationship to the n system components, while each component has a 1:1 relationship with the application logic. Each component may choose to provide additional interfaces and to communicate with other software (which may be other system components, but in general should not), for example user interfaces or external data sources.

Following this design, it is possible to specify the architecture through a concise set of components and interfaces, providing the necessary stability as well as the required flexibility for the experimental type of system which BOEMIE represents.

2.4 Modeling

The technical specification of the architecture is split into two parts. The first part, described in this document, contains use cases, processes and architecture models, while the low-level definitions of
the interfaces will be collected in an Interface Dictionary which is a supplement document to the architecture specification.

For the development of the architecture models contained in this document, state-of-the-art software modeling methodology is applied. The Unified Modeling Language (UML) in its version 2.1 is a widely applied technology which offers a suitable palette of diagrams for the purposes of this document. The modeling follows a three-step approach. In the first step, users and use cases are identified and captured in textual descriptions. The second step concentrates on the processes behind these use cases, decomposes them into single tasks and associates them with actors. The final step breaks the actors from the second step down into components, defines the structural framework, and arranges the components into the systems structure model.

From the list of diagrams available in UML 2, three diagram types are used to support this process:

- Use Case Diagrams (Step 1)
- Activity Diagrams (Step 2)
- Component Diagrams (Step 3)

Though with UML 2 it is possible to automatically generate code from the models, a classic non-generative approach to writing code for the BOEMIE prototype is assumed.

### 2.5 Implementation Strategy

The system implementation begins by transforming the UML models of the generic interfaces into a set of WSDL descriptions. Function libraries and XML Schema definitions (XSD) are developed to fine-specify the interfaces of the components; these will be covered by a separate Interface Dictionary document supplementing this specification. Using the generated WSDL files, the necessary infrastructure is set up, and interface test code is written. The test code is then used to thoroughly test the infrastructure and act as placeholder to be replaced later with the actual implementations of the interfaces. Once the interfaces are in place and tested, implementation of the application logic and components can begin. As soon as at least the application logic and one system component are available, test processes can be implemented and integration starts. With more components and processes being finalized over time, integration proceeds until the full system functionality is available.

### 2.6 Risk Handling

With a system of the complexity level of the BOEMIE prototype, several factors may endanger the timely completion of the implementation. In this section, we discuss the most obvious risks, knowing that there may be several other possible issues.

#### 2.6.1 Scarce Resources

The primary goal of the implementation process is to build functionality for all use cases described in this document in a state-of-the-art manner. The project team will use the (limited) resources available for integration to reach this goal as well as possible. However, it may become apparent during the process that it will be necessary to compromise between a state-of-the-art system from an engineering point of view and the aspects relevant for research. Since research and innovation is the main focus of the project, in a situation where resources are scarce, the team will concentrate its efforts on the research-related parts of the system, allowing the parts with less relevance for research to be implemented at a somewhat lower quality.
2.6.2 Interoperability Issues

The system is designed to keep interoperability issues at a minimum. However, especially during the integration of the bootstrapping process, interoperability issues may arise and endanger a timely completion of the system. In such a case, the project team will organize additional face-to-face meetings on short notice to solve these problems as quickly as possible and thereby avoid delays in the project schedule.

2.6.3 Delays in Component Development

For several possible reasons, the development of single components of the prototype may be delayed. Where such a delay arises and will affect the entire integration process if not dealt with, the team will re-asses the component in question to determine the best of the following solutions:

- Postpone or drop non-critical functionality
- Re-assign effort and allocate additional resources
- Reduce quality
3 Case Studies

The BOEMIE system provides a multitude of functionalities for several possible users (see section 4). To verify the practical applicability of the overall system, case studies of real-world application scenarios are desirable. Especially the application of several aspects of the BOEMIE system in one case study can help to provide proof-of-concept for the BOEMIE approach.

3.1 Integration with the LIVE project

LIVE will produce in real-time a non-linear multi-stream TV broadcast of the 2008 Olympic Games in Beijing which adapts to the interest of the viewers.

For this innovative TV experience the multiple incoming video signals and the available archive material will be indexed and structured by semi-automatic meta-data extraction tools. The identified video objects will be filtered and visualized to the professional users in the control room of a broadcast station. Additionally, feedback coming in from the TV consumers over a back channel mechanism will be analyzed by a recommender system. At the intelligent media framework layer the semantic connections between the user preferences and annotated video material are made. These results are fed into the control room to guide the production process.

The experiments and the development of an integrated prototype will be carried out at the ORF in Austria.

The project teams of LIVE and BOEMIE see prospective potential for mutual benefit from a cooperation. As a first application of the BOEMIE system in the LIVE scenario, an integration with the recommender system built in LIVE to support the production process with the knowledge gathered by the BOEMIE system is envisioned. As both the BOEMIE prototype and the LIVE Intelligent Media Framework use Web Services as interfaces, the technical integration is expected to create little additional effort.

In a second step, it is envisioned to explore the options of using the BOEMIE system also for the end user part of the LIVE system.

A common meeting between the two teams will be organized to define the exact application scenario.

3.2 Cooperation with content providers

The project team has established contact to major sport content providers such as the IAAF or Eurosport to explore options for a case-study cooperation. These efforts aim at providing prove-of-concept for the BOEMIE approach in a real-world scenario as well as setting up a convincing showcase for the prototype. Both content providers run large web sites with extensive amounts of athletics coverage which could be enhanced by applying the technology developed in the project to provide a better user experience. Interest to cooperate has been expressed by the IAAF, the contact to Eurosport is ongoing. Meetings with representatives of these organizations will be organized to clarify the exact scope of each case study.
4 Use Cases

The first step in the specification of any system's architecture is the identification of its use cases, which consist of the users who interact with the system, and the tasks the users perform with the system. Starting from a verbal description, these use cases are transformed step by step into more formal diagrams and specifications according to the selected modeling language.

To illustrate the use cases and their reflection in the system architecture, we will use the following example. It will be extended and explained in more detail throughout the document.

A large content provider and operator of a sports web site intends to integrate the BOEMIE system into his existing IT infrastructure. The shall analyze the providers content which is stored in a large content management system (CMS); each content item can be accessed via an unique URI reference into the CMS. The system shall be used to enhance the user experience of the content providers web site and thereby to attract more users to ultimately generate more advertising revenues.

4.1 The System Operator

Prospective operators of the BOEMIE system are commercial content owners, portal operators, syndicators, or broadcasters. An expected common characteristic of all expert users is the availability of domain-related content of some form, be it in multimedia archives or extensive web sites, which can be processed by the BOEMIE system to provide customers with improved access to the knowledge contained in this multimedia content.

It is anticipated that the commercial user of BOEMIE will integrate the system into an existing infrastructure to generate an increased value for the end users, creating input connections to available multimedia assets, output connections according to the end users' use cases and control connections for the administration interface. Illustration 1 summarizes the use cases of the BOEMIE system operator, further detailed in the rest of this section.

In our example, the system operator role is enacted by the content providers IT staff. Therefore, the system operator user interface will be used in the IT department. The technical integration aspects will be discussed in section 6.
4.1.1 Add Content

The system operator can add multimedia content to the system, either by actively uploading content through a user interface, or by adding references (URIs) that can be typed through a use interface or selected from a list which is populated by an automatic content discovery component, for instance a web crawler. Content added by the system operator is stored in the multimedia repository and scheduled for semantics extraction.

In the case of the exemplary content provider, content will be added using URI references into the existing CMS. The list of URIs will be extracted from the CMS itself. In the second step, it BOEMIE system as soon as it is added to the CMS.

4.1.2 Control Bootstrapping Process

The central process in the BOEMIE prototype system is the bootstrapping process which repeatedly extracts knowledge from multimedia content and evolves the ontology. With the new ontology version generated in one run of the bootstrapping loop, not only analysis quality for new incoming content can be improved, but also content already analyzed can be re-processed to detect instances of concepts unknown to the ontology at the time of the previous analysis. This process runs automatically most of the time, requiring assistance from a domain expert from time to time (see section 4.2) and is controlled by the System Operator who can start, suspend, resume or abort the process. According to the findings from experiments with the bootstrapping process, additional control options may be added, for instance a control to trigger re-analysis of already processed content.

The system will retrieve content items from the content providers CMS one by one, analyze them to extract semantics and populate and evolve the ontology with the extracted knowledge. The IT department can monitor the progress through the system operator user interface and control the process as necessary.
4.1.3 Monitor System Behavior

The system operator has the option to review the overall system status through a monitor interface which displays key process values from the system, messages and errors. The information is updated on a regular basis so that the operator can obtain up-to-date information about the system at any given point of time. These key values help the operator to decide how to run the system, for instance when to add content, when to start or stop the bootstrapping process, or when to shut down for maintenance.

4.1.4 Maintain system components

Control over the system's components means the ability to start, stop, and re-start individual components, or the entire system, for maintenance or error-recovery. Where components provide individual maintenance tasks (for example a backup or rollback function), these tasks are available to the operator in the system maintenance use case.

Through the same interface as for the bootstrapping process, the IT department can monitor and control all system components.

4.2 The Domain Expert

The second user which has been identified for the system is the domain expert, a human user with extensive knowledge about the application domain. The tasks of the domain expert are related to control of the ontology evolution and coordination process as described in D4.1, section 3.9. Illustration 2 summarizes the use cases of the domain expert, further detailed in the rest of this section.

In our example the content provider has several experts for various types of sport. The experts for the selected domains receive training in using the BOEMIE domain expert interface to control the evolution of the ontology.

Illustration 2: Domain expert use cases

4.2.1 Support Ontology Population

Where the interpretation of an incoming multimedia content is unambiguous, the system will act automatically on the population of the domain ontology. However, if more than one possible interpretation are found, the ontology evolution toolkit will try to disambiguate and select the most promi-
nent interpretation. In that case, the domain expert can interact with this disambiguation process through monitoring or through manual selection of the correct interpretation if required. Ambiguous interpretations are kept in a list of proposals. The domain expert can review this list and for each entry select the correct interpretation.

For instance, a content item analyzed by the BOEMIE system may be a still image showing an instance of high jump. The extraction process may have identified an athlete and a foam mat but no bar. The interpretation is therefore not clear. The most prominent interpretation would be high jump, but an interpretation as a pole vault event could as well be possible. The system generates an interpretation proposal and appends it to the list of proposals to be reviewed by the domain expert, who should accept the proposal, leading to ontology population.

4.2.2 Support Ontology Enrichment

Where the system discovers possible new concepts and/or rules (see D4.1, Patterns P3 and P4), the domain expert has the task to review the evidence found by the system and revise, approve or disapprove the proposals made by the system. System proposals are made also the ontology coordination service (see D4.1, 8.2.4), which allows manual interaction by the domain expert to set one or several of its parameters. Although default values for all parameters exist, changing them is an optional task to be performed when the result of the ontology matching process seems less than satisfactory. The system provides the expert with an interface that allows the setting of these parameters. If the system cannot label new concepts or rules itself, the domain expert also labels these.

The system keeps a list of all possible new concepts and rules. The domain expert can review this list and can either accept, modify or reject the proposed enrichment in case the supportive evidence does not justify the addition of the proposed items.

The domain expert can also define new concepts and rules without the system having made a proposal. The system provides a user interface that allows the domain expert to create, review, modify, save and delete concepts, relations and rules.

The system installed at the content provider may, for example, find that there are often unknown white shapes which are all located at the lower end of an athlete’s leg. It may inform the domain expert of this finding by creating a proposal for a new concept that represents these white shapes. The domain expert can review the proposal, decide that it is meaningful in the application domain and label it “shoe”. The domain ontology is then enriched by adding the new concept.

4.2.3 Support Ontology Coordination

When the ontology has been enriched, one or several ontology mappings between the internal ontologies of the BOEMIE system and external ontologies may need to be updated for coordination purposes. The validation and selection of the mappings to be updated is interactive with the domain expert. An appropriate interface allows the expert to select and validate these mappings. Also in this case, the domain expert can interact with the ontology matching service (see D4.1, 8.2.4) for setting the appropriate parameter configuration for coordination. Changes in these settings will typically happen when the generation of proposals for new axioms and rules produces many false positives.
4.2.4 Validate Ontology Mapping

When the ontology has been enriched, one or several ontology mappings may need to be updated. The validation and selection of the mappings to update is a manual task which is performed by the domain expert. An appropriate interface allows the expert to select and validate these mappings.

4.3 The End User

The end users can exploit the BOEMIE prototypes by using the knowledge accumulated by the system, hence having improved semantic access to the application domain and the analyzed multimedia content. Since the BOEMIE services have been designed to be used in various application scenarios, end users cannot be classified into particular user groups, rather they could be characterized as a heterogeneous group comprising of members with diverse interests and expertise. In order to specify generic BOEMIE system use cases for the users, we identified as a common factor for all end users the need to query into the systems knowledge for browsing content according to their needs and/or preferences. More specifically, we identified the “Query Content”, “Suggested Reading” and “Content Location” use cases to be apt, as shown in Illustration 3. These can be provided via a properly developed query engine which exploits the semantic links between concepts to allow for advanced content browsing.

*The exemplary content provider plans to enhance his web site with the system. This is done by adding the pages generated by the BOEMIE system to set of pages, or by using the BOEMIE proxy system for browsing the web site through it.*

![Illustration 3: End user use cases](image)

4.3.1 Query Content

The BOEMIE system constitutes a Knowledge Base which can be queried using standard query languages, by specifying search terms. Using semantic links and rules from the Ontologies and Reasoning Mechanisms developed within BOEMIE, the prototype can provide suitable content that satisfies the query criteria. Within the BOEMIE approach, querying the BOEMIE system will differ from queries to well-known search engines in that the system will not be limited to term/thesaurus based search algorithms; instead it will use the domain-specific semantic model to reason about relevance of concepts and instances for the query. Furthermore, this can lead to a more concise result set compared to classical search engines, which is more comprehensive in terms of relevant results at the same time.
According to the envisioned use case, users will be allowed to specify their search criteria either by using a free-form functionality or by selecting the criteria that satisfy their needs from lists of known concepts automatically populated by the BOEMIE system. Then, the BOEMIE system can process the query in order to return a list of results in addition to evidence why the results provided to the user were considered relevant for the particular query. Furthermore, for each result, the user will be able to review the related concepts and refine his search by following links to these concepts, thus creating a new results list. This mechanism allows for navigation on the semantic net formed by the BOEMIE domain ontology.

The end user can open the query page, enter one or several search terms and submit his query. The BOEMIE system will answer the query and enrich it with both the knowledge extracted from the content and links to the relevant content items. The user can then enter new search terms, follow semantic links presented by the system to change or refine the search or view the content items found.

4.3.2 Browse Content / Suggested Reading

It is a fact that from a research point of view, a parameterized search is not considered as a very intriguing problem. In addition to that, it must be acknowledged that as far as the end users are concerned, the numerous advantages of using semantic queries are not easy to be perceived. Therefore, the BOEMIE prototype will provide an advanced interface that grants access to the knowledge in the BOEMIE ontologies in a way appealing to end users. To become more specific, instead of expecting from the user to type in search terms or to select them from a list, the interface will adapt to the user's browsing behavior in order to identify the users' focus of interest so as to leverage their search and content browsing effectiveness and efficiency.

The core of our approach is that each multimedia document may be related to one or several concepts or instances from the domain ontology. By properly managing the user's interaction with the document in terms of activity tracking and processing, information on how relevant these concepts are to the user can be extracted. More specifically, the BOEMIE system will employ a relevance scale in order to rank the results according to the interest focus of every user. In this way concepts will be assigned with a relevance level. Using a properly developed scoring mechanism for assigning relevance values to the concepts, the selection of relevant concepts can be updated and narrowed after the user views a new document. The focus of interest could then be defined as the set of concepts from the domain ontology, which carry the highest relevance values. For instance, if the user visits two specific URLs consequentially, the system could assume that their content is considered as relevant by the user. On the contrary, if a document is reviewed only very briefly and the user returns quickly to the previous page, this may be interpreted as evidence that this page's content is not very relevant to the initial query.

Using this information, the BOEMIE system will strive to supply the user with a properly calculated selection of suggested reading documents. These may come either from the repository of content known to the BOEMIE system or they could be identified through external sources like web directories. The user interface to be developed could emphasize on the web based use of the BOEMIE system and add links for the suggested documents to the original web pages in way of code insertion. In order to realize the necessary user tracking and code insertion mechanisms, an HTTP proxy system will be built so as to allow accessing web pages in a transparent manner while keeping track of the users’ focus of interest and inserting the suggested reading links automatically.

In order to respect the original content providers as well as not to disturb the users, very small icons will be added to the page in places where relevant concepts are represented. In this way, when hovering the mouse pointer over these icons, pop-up layers which contain links to the proposed documents will be activated. Rich Internet Application (RIA) or Browser Plug-In techniques may be
used to allow the user dragging and keeping these windows for later use in case that the user decides to follow other links first.

Consider a user interested in high jump events and the high jumper Yelena Isinbayeva in particular. When searching for information on this athlete, starting from a random document, the navigation path may lead the user to the athletics section first, and then to web pages related to high jump events. If the user browses a certain amount of pages about high jump event, all of which mention Yelena Isinbayeva, the BOEMIE system can conclude that this concept is relevant to the user. Based on this conclusion, the web pages could be enriched with more information considered related to the user’s query and browsing records. Illustration 4 presents a depiction of the IAAF web site, as enriched with suggested reading links added by the BOEMIE system. In a similar vein, Illustration 5 depicts the same site, with suggested reading overlays, where the user is provided with the option to follow additional links related to his interests or to continue his browsing without distressing him.
Illustration 5: Suggested Reading Overlays: The user may choose to follow the links or continue normal browsing.
4.3.3 Locate Content in a Map

The combination of domain specific knowledge from the domain ontology with location information from the geographic ontology leads to the specification of the third end user use case. It aligns the two ontologies in the form of two user interface components, a content window and a map window. Where geographic information has been extracted from multimedia content, the multimedia asset and the corresponding map position can be displayed side by side. With content in one window and the map in another window, this allows the user to navigate in two ways. The content window can be used to navigate through or follow the temporal flow of the content. In this case, the map window will be automatically updated whenever a new geographic concept is found in the content. For example, in the video footage from a marathon event, several landmarks or street signs may be detected. These can be located in the map and the map window updated to show the location of the concept.

On the other hand, the map window can be used to move to another geographic position; in this case, the domain ontology may be used to identify content which contains geographic concepts that are close to the current map position, and these content items can be shown as points of interest in the map, updating the content window when the user selects such a point of interest. Switching between the two windows, the user can freely navigate through the combined domain/location space.

For instance, when reading web content about the Osaka Athletics GP in the browser window, the map window is updated to show the location mentioned in the content. At the same time, other content available from the same region is marked in the map window. By clicking on the marked spots in on the map, the user can update the browser window to display the content (see Illustration 6).

Illustration 6: Example of locating content on a map: The green arrow shows the location of the content in the browser window. Colored circles indicate other content with close geographic relation to the current content.
5 Activities and Processes

With the use cases in place, the next step in the architecture development is to analyze the use cases and identify the tasks involved in each of them and the actors that carry out these tasks. The tasks and actors are modeled using UML Activity Diagrams. These diagrams form the basis for the subsequent modeling of the systems component architecture.

The following sections show the activity diagrams resulting from the first decomposition step and describe the tasks and actors in each use case.

5.1 Adding new Content

The process of adding and analyzing new content can follow two patterns. On one hand, the system operator can explicitly add new content to be analyzed, either by physically uploading the content to the multimedia repository or by registering uniform resource identifiers (URIs) which point to the multimedia documents. On the other hand, the system's automatic content acquisition function (a web crawler for instance) can identify interesting new content and suggest it for being added to the system. In the latter case, the system operator can choose the URIs to add from the list of proposals generated by the automatic content discovery tool.

In the example, a member of the IT staff opens the system operator user interface and chooses the “Add content” function. He is presented with a text input box for typing in a URI, a file selector box for uploading content and a list of URIs retrieved from the System Opera-
tor Module, which may have a textual list or a direct connection to the CMS. Selecting a few URIs from the list and starting the operation, the selected URIs are added to the list of content items to be analyzed by the Semantics Extraction Toolkit through the System Operator Module.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Operator User Interface</td>
<td>is responsible for the communication with the human system administrator.</td>
</tr>
<tr>
<td>System Operator Module</td>
<td>contains all particular functions required for the communication between the core system and the system administrators user interface.</td>
</tr>
</tbody>
</table>
5.2 Bootstrapping

Once new content has become available, it is passed to the semantics extraction toolkit for analysis. This process is described in detail in D 2.1, section 2. Results of this process are ontology A-boxes containing Mid Level Concept Instances (MLCis), High Level Concept Instances (HLCis) (explanations) and relations between these MLC/HLC instances.

According to the explanations found for the identified MLCis, the ontology evolution toolkit will populate the domain ontology, requesting support from the domain expert where necessary. Deliverable D 4.1 describes this process in detail. Where the background knowledge was insufficient for an explanation of the instances of an ABox, this ABox is classified as “unknown”. In the next step, the system tries to identify clusters of concepts/relations contained in ABoxes classified as un-
known that may be candidates for new concepts/relations. If such clusters can be identified, the ontology evolution toolkit tries to enrich the ontology with new concepts with the help of the domain expert (see section 4.2).

If new Mid-Level Concepts have been added to the ontology in the process, then the semantics extraction toolkit can retrain its extractors to learn how to identify instances of these new concepts, improving the recognition quality. If new High-Level Concepts were added, the system can use this newly gained knowledge to re-interpret the extracted knowledge. This loop is continued until no new knowledge can be extracted any more.

Consider an image about a pole vault event from the CMS undergoing the bootstrapping process: First it is passed to the Semantics Extraction Toolkit which extracts low-level features and tries to identify concepts from the domain ontology. The Semantics Extraction Toolkit identifies a horizontal bar, an athlete, a foam mat and a pole and classifies the image as pole vault image. Assume that it also finds an oval white shape close to the lower end of the athlete which it cannot identify. It generates an A-Box for the image and passes it to the Ontology Evolution Toolkit. The Ontology Evolution Toolkit populates the ontology with the detected concept instances. The Semantics Extraction Toolkit may have found several unknown concepts in various images and suggest an attempt to identify new Mid-Level Concepts (MLCs) after having populated the ontology. It clusters the unknown concepts and may find, among other clusters, several instances of oval white shapes that are located at the lower end of an athlete instance. It proposes a new Mid-Level Concept which is later identified to be a shoe by the domain expert. The ontology is evolved by adding the new concept and the Semantics Extraction Toolkit is triggered to retrain the extractors so that the new concept can be identified. If the system is idle, it can re-analyze existing content to try and find more instances of the new concept.

The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantics Extraction Toolkit</td>
<td>is the toolkit for analyzing multimedia assets as described in deliverable D2.1.</td>
</tr>
<tr>
<td>Ontology Evolution Toolkit</td>
<td>is the toolkit responsible for population and enrichment of the domain ontology as described in deliverable D4.1.</td>
</tr>
<tr>
<td>System Operator User Interface</td>
<td>see section 5.1</td>
</tr>
<tr>
<td>System Operator Module</td>
<td>see section 5.1</td>
</tr>
</tbody>
</table>
5.3 **Monitor System Behavior**

To monitor the system behavior, the system operator will open a System Operator User Interface which provides an overview of the overall system status and indicates if there are errors that need attention. This information is gathered by a system operator module that accesses other components to monitor their status. If there are errors, the system operator module logs all error information and the user interface allows the system operator to review these details in order to check if the error is recoverable or not. If the error can be recovered, then the system maintenance process can be used to recover the error (see next section). Otherwise, administrative tasks outside the scope of the systems use cases may be required (e.g. a restart of the component, network infrastructure or operation system).

*For example, the content provider's IT staff may open the System Operator User Interface and select the “system status” function. The user interface will retrieve status information from the System Operator Module which in turn constantly collects information from all known system modules. This may reveal that the multimedia repository is not working. By clicking on the corresponding link, the operator can request additional information through the System Operator Module, which may tell him that the network connection to the repository has produced timeouts.*
The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Operator User Interface</td>
<td>see section 5.1</td>
</tr>
<tr>
<td>System Operator Module</td>
<td>see section 5.1</td>
</tr>
</tbody>
</table>

5.4 *Maintain System Components*

System maintenance is executed through a corresponding interface which gives the system administrator access to all maintenance functionalities of the system and its components. The components may either provide standardized tasks through a web service, in which case a default interface is used to trigger and control these tasks, or specialized tasks through dedicated web pages which are provided by the component itself and linked from the maintenance interface. To perform a maintenance task, the system operator selects the component to be maintained from the system operator interface, which lists the maintenance tasks available for this component. The operator then chooses the task to perform, upon which the interface provides a form that allows configuration of the tasks settings. Having configured the maintenance task to his needs, the operator starts the task and re-
views the result which is presented in the user interface. If the result is not satisfactory, the operator may choose to execute another maintenance task.

Continuing the previous example, the operator may click on the multimedia repository's “maintenance” link to see what maintenance options the repository component offers. From the list of options, he selects “restart HTTPS server” and the System Operator Module will trigger this function on the repository component. The result of the function is reported back and the operator can see that the server has successfully been restarted. Using the system monitoring function, he can see that the timeouts have ceased.

The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Operator User Interface</td>
<td>see section 5.1</td>
</tr>
<tr>
<td>System Operator Module</td>
<td>see section 5.1</td>
</tr>
</tbody>
</table>
5.5 **Support Ontology Population**

Where more than one explanation for a set of identified MLCs from a multimedia document are available, the ontology evolution toolkit will check the explanations for similarities/contradictions with the existing ontology and refine the explanations automatically before populating the ontology. The refinements made by the system may be reviewed by the domain expert by opening the list of proposed refined explanations in the domain expert user interface and selecting single proposals from the list to see the assigned explanations and refinements made. The domain expert has the option to overrule the assumptions made by the system, select an appropriate HLCi and assign it as explanation. The system will then populate the ontology with the HLCi as defined by the domain expert.

*An expert for Athletics events at the content provider could for example open the Domain Expert User Interface and choose to review the list of identified sports instances. In the list, he selects an instance of the High Jump concept. In the user interface, he can see the content item that contained the concept instance, a list of Mid-Level concept instances identified in the content and the refined explanation. Spotting a small portion of a pole in the image, the domain expert can overrule the automatic explanation and assign “Pole Vault” as correct interpretation manually. The Ontology Evolution Toolkit updates the ontology accordingly.*
The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Expert User Interface</td>
<td>is responsible for all interactions with the human domain expert user</td>
</tr>
<tr>
<td>Domain Expert Module</td>
<td>contains all particular functions required for the communication between the core system and the domain expert user interface.</td>
</tr>
<tr>
<td>Ontology Evolution Toolkit</td>
<td>is the ontology population and enrichment toolkit as described in deliverable D4.1</td>
</tr>
</tbody>
</table>
5.6 Support Ontology Enrichment

Where ontology evolution toolkit identifies candidates for new Mid-Level Concepts or High-Level Concepts, the domain expert is required to review and/or label the proposals generated by the toolkit before the ontology is enriched. The domain expert can access a list of all proposals generated by the ontology evolution toolkit through the domain expert user interface. By opening a proposal, he can review the evidence used by the ontology evolution toolkit to generate the proposal, and decide whether the proposal is valid, in which case the expert can use the user interface to define a new high-level or mid-level concept and trigger enrichment of the ontology, or reject the proposal as false positive.

In the case where the system has identified a possible new concept of white oval shapes at the lower end of an athlete, the athletics expert reviews the proposal and decides that this is indeed a valid new concept. He enters the concept details into a form and submits the data to the domain expert module which uses the Ontology Evolution Toolkit to enrich the ontology.

The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Expert User Interface</td>
<td>see section 5.4</td>
</tr>
<tr>
<td>Domain Expert Module</td>
<td>see section 5.4</td>
</tr>
<tr>
<td>Ontology Evolution Toolkit</td>
<td>see section 5.4</td>
</tr>
</tbody>
</table>
5.7 Support Ontology Coordination

To adjust settings for the ontology matching and mapping processes, the domain expert may use the domain expert user interface to access the mappings and parameters, review the current values, adjust these values and apply the changes. The domain expert module is responsible for getting and setting the values in the ontology evolution toolkit.

The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Expert User Interface</td>
<td>see section 5.4</td>
</tr>
<tr>
<td>Domain Expert Module</td>
<td>see section 5.4</td>
</tr>
<tr>
<td>Ontology Matching Service</td>
<td>Part of the Ontology Evolution Toolkit, see section 5.4</td>
</tr>
<tr>
<td>Ontology Mapping Tool</td>
<td>Part of the Ontology Evolution Toolkit, see section 5.4</td>
</tr>
</tbody>
</table>

Illustration 13: Activity Diagram: Support Ontology Coordination (Domain Expert)

To adjust settings for the ontology matching and mapping processes, the domain expert may use the domain expert user interface to access the mappings and parameters, review the current values, adjust these values and apply the changes. The domain expert module is responsible for getting and setting the values in the ontology evolution toolkit.

The actors identified in this step are:
5.8 Query Content

The first step in querying content from the BOEMIE system is the specification of query parameters. The user can do this through a form in the End User's user interface. The parameters are formulated into a query to the inference service by an End User specific module. The query is answered by the reasoner, the returned results are processed by the End User module again to add the identified multimedia assets and the user is presented with the results through the user interface.

An end user can open the BOEMIE-specific query page provided by the End User Interface which has been integrated into the content providers web site and type in a query about pole vault events in London in 1999. The terms are submitted to the End User Module which uses the reasoner module to answer the query, adds links to the relevant content in the multimedia repository and displays the result in the End User Interface.

The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>End User's User Interface</td>
<td>is responsible for all interactions with the human end user.</td>
</tr>
<tr>
<td>End User Module</td>
<td>contains particular functions required for the communication between the end user's user interface and the core system.</td>
</tr>
<tr>
<td>Reasoner</td>
<td>can answer queries into the domain knowledge gathered by the system.</td>
</tr>
</tbody>
</table>
5.9 **Browse Content / Suggested Reading**

By using the normal web browser, the user specifies a series of URLs of pages he/she looks at. The BOEMIE system, acting as proxy server, reads these URLs and forwards the request to the specified server to obtain the content. The content returned from the original server is then looked up against the list of already analyzed content\(^2\). If the content is known to the system, the associated concepts are identified and used to update the user's focus of interest. According to the updated focus of interest, relevant concepts and associated multimedia assets are looked up for the user. The original server's answer is rewritten to add links to the suggested content, and the answer is served back to the user.

*For instance, an end user may be interested again in pole vault events in London in 1999. From the content providers web site main page, he follows a link to the “Athletics” section. The BOEMIE system, working as proxy, analyzes the users navigation behavior. The end user module forwards the request to the content providers server and looks up the URI in the list of known content. It finds several athletics-related concepts mentioned there and*

---

\(^2\) In an improved version with real-time analysis capabilities, the lookup can be exchanged for an on-the-fly concept extraction.
adds them to the users focus of interest. Since the focus is very unspecific, no recommenda-
tions are made so far. Next, the user navigates to the marathon section and starts reading
an article about the last London Marathon. Following the same pattern, the end user mod-
ule updates the users focus of interest to marathon and adds London to the list. The system
adds suggested reading links for other athletics events in London and for other marathon
events outside London. It will also suggest past London Marathon coverage from the
archives, results from athletes mentioned in earlier London Marathons etc. The result is
delivered to the normal browser window of the user.

The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Acquisition</td>
<td>Connection to the “Adding New Content” use case, see section 5.1</td>
</tr>
<tr>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>Multimedia Repository</td>
<td>stores multimedia documents and references to already-analyzed content.</td>
</tr>
<tr>
<td>End User's User Interface</td>
<td>see section 5.8</td>
</tr>
<tr>
<td>End User Module</td>
<td>see section 5.8</td>
</tr>
<tr>
<td>Reasoner</td>
<td>see section 5.8</td>
</tr>
</tbody>
</table>
5.10 Content Location

When the content location interface is opened, the user needs to select which window to interact with. If the content window is selected, a multimedia file can be opened and viewed. Location information in the content can be identified and the map window updated accordingly. If on the other hand the map window is interacted with, the user can specify or navigate to a location, content assets with locations close to the selected one can be selected and the content window is used to open these content items.

The actors identified in this step are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>End User's User Interface</td>
<td>see section 5.8</td>
</tr>
<tr>
<td>End User Module</td>
<td>see section 5.8</td>
</tr>
<tr>
<td>Map Module</td>
<td>responsible for the interaction with the geo-location provider component.</td>
</tr>
</tbody>
</table>
6 System architecture

With the specification of the systems’ processes as input, the next step is the definition of the system architecture. Starting from a high level architecture view, the system components, interfaces and communication are elaborated.

6.1 High level architecture

The considerations made in chapter 2 regarding an open architecture have led to the general principle of an open communication system to which all system components are connected. By adding to the communication system the core components that have been identified in the process definitions, the following block diagram of the overall system architecture can be drawn:

![Block diagram: BOEMIE core system](Illustration 17)

In this diagram, the major building blocks of the system are visible. The prototype application consists of four parts:

1. **Core Components (red)**
2. **Use Case Modules (green) and User Interfaces (orange)**
3. **Integration Components (yellow)**
4. **A common communication bus**

The core components are those parts of the system that are indispensable for the bootstrapping process itself: Semantics Extraction Toolkit (developed in WP2), Ontology Evolution Tool (developed in WP4), Application Logic (developed in WP5), Reasoner (provides by TUHH) and Location Provider (provided by TeleAtlas).

For each use case, one or several use case modules and user interfaces may be added to the system. These components implement additional functionality required by the use cases' processes. They are
connected to the common communication bus and modules are registered with the application logic. After these two steps, the components can be used by all process components.

Integration Components are additional building blocks which provide functionality required to connect and combine core components, use case modules and interfaces.

The application logic has the task to coordinate the cooperation between all other components. The application logic component itself has little knowledge about the BOEMIE processes. Instead, processes are modeled as process components which are plugged into the application logic. The application logic component acts as intermediary between these processes and other system components, providing common services like message queues, network interfaces or similar. This distinction between application logic and process components provides for later addition of new processes, either for new use cases or new internal functionality. It is one of the key features that contribute to the system's openness.

All communication between components is run through the common bus to which all components are connected. The bus itself is not managed, all components communicate as equal peers.

The components of this high level architecture diagram are described in more detail in the next subsection.

6.2 Building Blocks

This section adds detail to the building blocks defined in the previous high level architecture concept. In particular, the specific functionalities are captured in textual form. The mapping to technical interfaces is part of the supplementary Interface Dictionary.

6.2.1 Core Blocks

6.2.1.1 Semantics Extraction Toolkit

The semantics extraction toolkit technically encapsulates the work done in Work Package 2 and provides three functionalities towards the integrated system:

- It processes multimedia documents, extracts low-level features, identifies mid-level concept instances and relations between these instances, and tries to find an explanation for the identified instances among the known high-level concepts of the domain ontology. This function schema is called “Analysis” (see D2.1, section 2.3.1)

- It uses manually labeled content to train its internal (modality specific) analysis modules to improve analysis accuracy. Evolved ontology versions generated by the ontology evolution toolkit are used as input for this training. This function schema is labeled “Training” (see D2.1, section 2.3.2)

- It uses clustering techniques to discover prospective new modality-specific mid-level concepts that are unknown to the domain ontology so far. The identified clusters are used by the Ontology Evolution Tool. This mode of operation is called “Discovery” (see D2.1, section 2.3.3)

The semantics extraction toolkit uses multimedia documents as input and generates OWL ABoxes as output.
6.2.1.2 **Ontology Evolution Toolkit**

Like the semantics extraction toolkit for Work Package 2, the ontology evolution toolkit encapsulates the output of Work Package 4 and provides its functionalities towards the integrated prototype through a uniform interface. It uses input from the semantics extraction toolkit to populate and enrich the domain ontology, generating new versions of the domain ontology. Specifically, document interpretations from the semantics extraction toolkit are used to populate the ontology, while discovered new concepts (either from the semantics extraction toolkit or from the ontology evolution toolkit) are used to enrich the ontology. The new ontology versions generated by the ontology evolution toolkit are used by the semantics extraction toolkit for training of the internal analysis modules.

6.2.1.3 **Application Logic**

The application logic interconnects the functionalities of the entire BOEMIE prototype. Its main purpose is to provide stable interfaces between the flexible set of system components and the system behavior. The application logic uses *process components* which implement the use case processes identified above. Each process component captures exactly one process. The process components are plugged into the application logic and use the infrastructure provided by the application logic to control the system behavior.

A standardized interface allows the application logic to access process components and use case modules and integrate them into the system dynamically. Each use case module defines its own interface with respect to function names, input and output. Process components can use the functionality by sending messages to the modules using the standardized use case module interface. The messages must contain the module name, functions name and all required parameters. Each module sends back a response message to the application logic through the application logic's well-known interface, including a reference to the sender process, the function call and the result.

One of the process components in the application logic is the *bootstrapping* process component which controls the behavior of the bootstrapping process. Together with its structural counterpart, the bootstrapper module, it implements the core aspect of the BOEMIE system, the bootstrapping loop.

6.2.1.4 **Reasoner**

The reasoner provides standard and non-standard reasoning services towards the Semantics Extraction Toolkit, the Ontology Evolution Toolkit and to Use Case Modules.

6.2.1.5 **Location Provider**

The location provider is a web service offered by TeleAtlas through which geographic references can be located on a map, map data can be displayed, points of interest can be located and routing functions can be used.

6.2.2 **Use Case Blocks**

Use cases are implemented in the BOEMIE prototype by writing process components which contain the use case behavior; these have been introduced in the previous section. The structural counterpart to these process components are Use Case Modules which provide necessary functionality not covered by the core system, and User Interfaces which access the system via these use case modules.
6.2.2.1 Use Case Modules

Use case modules add process-specific non-core functionality to the system. They can encapsulate arbitrary functions which can be used by arbitrary processes. Each module has a unique module name and provides a set of functions which are identified by the module name and the function's name. The input and output data formats of the functions are defined through the XML Schemata of the Interface Dictionary. Use case modules may interact with other pieces of software to achieve their functionality; they may also be wrappers for existing tools. If necessary, use case modules can leverage core components or other use case modules, though this is not recommended as it may destabilize the architecture approach.

6.2.2.2 User Interfaces

User Interfaces present the user with the information, interactions and components required to fulfill the task represented by each use case. They access the BOEMIE system via the corresponding use case modules. User interfaces may interact with several use case modules and combine several use cases. One user interface will be built per user group (system operators, domain experts, end users). More user interfaces may be added as further use cases are identified or when it seems necessary to have a separate interface for a single use case.

6.2.3 Integration Blocks

6.2.3.1 Multimedia Repository

The multimedia repository is a local storage for multimedia documents and single-media assets. It is used by the semantics extraction toolkit for analysis of the contained documents and can be used for storage of split-modality assets. It can also be used by use case modules and interfaces to present the user with multimedia content where appropriate.

6.2.3.2 XML Database

The XML Database is a universal information storage for XML-formed documents generated by various components of the BOEMIE system. It provides a standard interface to store, update and query XML documents.

6.2.3.3 Ontology Repository

The task of the ontology repository is to keep track of the ontology evolution. It stores all versions of the domain ontology and provides access to the latest version as well as to earlier versions.

6.2.4 Common Communication Bus

The common bus connects all components of the BOEMIE system and enables communication between the components. The bus will be implemented as an IP network between the servers hosting the components so as to allow for the envisioned Web Services infrastructure to work.
6.3 Component Architecture

In this section, the building blocks identified in section 5.2 are further decomposed and detailed to yield structure, communication relations, high level interfaces and generalization between components. The following diagram takes the system architecture down to an aggregate component diagram level.

Illustration 18: Component Diagram: First Level System Architecture (see next page for large version)
6.3.1 Semantics Extraction Toolkit

The semantics extraction toolkit is an encapsulated component, developed and maintained in Work Package 2. It provides a standard interface through which application logic and process components can control the toolkit operations and direct communication with the ontology evolution toolkit can happen. It accesses multimedia content placed in the multimedia repository (and other sources), the XML Database to store and retrieve intermediary extraction results and the application logic's standard interface to call process components, send system messages or signal errors.

6.3.2 Ontology Evolution Toolkit

The ontology evolution toolkit is an encapsulated component, developed and maintained in Work Package 4. It provides a standard interface through which application logic and process components can control the toolkit operations and direct communication with the semantics extraction toolkit can happen. It uses the ontology repository to read and store the semantic model, create new versions of the ontologies and store ontology evolution logs. It also uses the application logic's standard interface to call process components, send system messages or signal errors.

6.3.3 Application Logic

The application logic acts as a hub. It provides a central standard interface which is used by most other system components for synchronous and asynchronous communication and uses in turn the standard interfaces of all other components to control system operations. The component diagram (Illustration 18) shows process components for all described use cases and use case modules for all three user roles, together with the corresponding aggregations. For readability reasons, relations between application logic and specialized components are not repeated but modeled once at the more general component level.

6.3.4 Process Components

As shown in Illustration 18, the processes of the prototype have been modeled as sub-classes of a parent general process component class with a standard interface towards the application logic. The standardization of the interfaces allows the later addition of further process components when the system functionality should be extended. The process components in turn use the application logic's host interface to access other components and functions in the system. The application logic works as proxy and forwards the process components' requests to the actual system components (and vice versa). In the diagram, process components for all use cases identified before have been modeled.

6.3.5 Use Case Modules

As with process components, the individual use case modules are shown in the first level diagram (Illustration 18) as sub-classes of the more general use case module class. Three use case modules for the three user roles have been modeled. All use case modules communicate with the application logic through standard interfaces, plus they provide specialized interfaces towards their user interface components and also external interfaces for integration with other systems where applicable. These modules provide functionality that can be used by process components through the application logic's host interface.
6.3.6 User Interface Components

User interface components are relatively independent of the core system, communicating with it either through specific use case module interfaces or the application logic. These components implement input/output functionality where interaction with human users is required. All user interface components follow the Model-View-Controller pattern and provide a listener interface through which updates can be triggered by the application logic.

6.4 Communication

The communication model of the BOEMIE system must comply with the open architecture described in section 2. Especially it must be taken into account that new modules or use cases will be added to the system in a later stage. Also, the application logic will have to track information about the ongoing communication processes to be able to monitor the system. Therefore it is not feasible to have specialized interfaces at every module.

The approach to be used in BOEMIE is based on generic interfaces. With those interfaces, it will be possible to extend the system, adapt to new use cases and implement monitoring functions independent of the modules into the central application logic.

The calls on the generic interfaces will contain information about which module is to be addressed, which method is to be called and a list of required parameter sets mapped into XML. The result is also presented in XML format. A call on the generic interface would therefore have the following layout:

\[
\text{result} = \text{call}(\text{moduleName}, \text{methodName}, \text{parameterSet 1, parameterSet 2, ...})
\]

In this call, moduleName and methodName will be modeled in URI format. The parameter sets are individual XML documents. This way, results of several other methods can be put into a call.

Of course, this generic type of interfaces cannot make any assumptions on the parameter sets that will be passed within the method calls. Therefore, each parameter set needed by or returned by a method must be defined as an XML schema separately. All modules which will make calls on other modules are obliged to pass parameter sets that validate against the appropriate XML schemata. Validation will not happen under responsibility of the communication infrastructure.
7 Integration Platform

7.1 Requirements

As described in sections 2 and 6.4, the use cases for the BOEMIE system are not all clearly known in advance. This implies the necessity to be able to add further use cases later on. Also, the system architecture might undergo adaptions when new use cases are added. Finally, the BOEMIE system includes a lot of different tools for the different scopes, which all have to be made working together. Even yet unknown components might be added and needed to be integrated into the system in a later stage.

These considerations have led to the design of an open architecture to obtain the needed flexibility. Of course, this open architecture must be reflected in the integration platform, which therefore has to allow for all the considerations mentioned before.

7.1.1 Security

The BOEMIE system collects a lot of multimedia content for analysis. This content might be protected by copyright laws, which prohibit its re-distribution. Therefore it is mandatory, that all content within the BOEMIE system is protected against unauthorized access. The protection of content may be achieved by storing it only in an encrypted way. This way, even if someone manages to get access to the content, it will still be of no use for the attacker.

Since BOEMIE is designed as an open architecture, several interfaces between the different modules of the system exist. Some interfaces, like for example the end-user module, will also be exposed towards users outside of the BOEMIE system. Through some of the interfaces, content underlying the restrictions mentioned before will be exchanged between the modules. Furthermore, as BOEMIE will be used to build up a knowledge base, it must also be protected against intentional misuse by manipulating the ontology. Therefore it is essential to protect all internal interfaces against unauthorized access.

Basic interface protection can be achieved by using network infrastructure like firewalls with access lists which limit the access to the system. This way, only authorized machines will be given access to each other. Regarding the communication itself, encrypted protocols will ensure that no information is sent between two modules in readable form, which prevents from attacks which simply gather information by sniffing on the network. To enable secured communication, a basic infrastructure for authentication must be set up.

As the first prototype of the system will not be used for public tests, security will be regarded as far as necessary in the design, though most probably not be implemented fully. Especially the infrastructure needed for secure communication, providing certificates and/or user databases for example, will most probably be implemented in a very basic way only. For the public showcase, all relevant security features will be implemented and enabled to ensure proper protection of the system and the content.

7.2 Technical survey

The BOEMIE system is being developed and implemented by various different partners. To gather information on the implementation requirements, a technical survey has been conducted, in which the partners were asked for their preferences and requirements for the technical implementation of the BOEMIE system. The results of this survey helped in finding an appropriate solution for the architectural requirements.
### 7.3 Results

#### 7.3.1 Programming Language

As the BOEMIE system consists of various different tools, not all of them will use the same programming language. All partners declared that at least some parts of their work will be done in Java. Some of the tools, especially those for analyzing the documents belonging in the various modalities, will be written in C/C++ due to performance issues and because some of them are based on previous work. Furthermore, Lisp and Tcl/Tk will also be used. This leads to a mixture of different programming languages which cannot be avoided.

#### 7.3.2 Operating System Platform

Most partners will use Linux as their preferred operating system, while at least one partner will not be able to use Linux. For Java as a platform independent programming language, it is also possible to directly use the tools on other platforms. For the other languages, a recompilation might be necessary, but at least all partners declared that their tools should also be working on Windows (Win32) based machines. By design of the BOEMIE system, partners will not be limited in their choice of the operating system.

#### 7.3.3 Interface Model

The different tools demand for different programming languages. Also, different operating systems will be used. Furthermore, yet unknown tools might be added later. These preconditions demand for a loosely coupled system, based on modules and interfaces. By using standard open interface techniques, the modules become independent from each other and can therefore be implemented in the preferred language and can be run on the preferred operating system and still achieve the required communication.

Very well known approaches for such interfaces are web services, which use an XML-based messaging for remote procedure calls. This way, they are independent from the underlying architecture or programming language, because all calls are handled in a textual way.

### 7.4 Web Services Infrastructure

#### 7.4.1 SOAP stack

SOAP is a lightweight protocol used in web services to exchange messages. It is a W3C recommendation and defines rules for the message design. For BOEMIE, version 1.2 of SOAP will be mandatory.

There are several SOAP stacks available for different programming languages. Due to the nature of web services, all of them should be able to work together, but to ease interoperability all implementations must at least comply to the WS-I Basic Profile 1.0. This way, all partners are free to choose their preferred SOAP stack. This is especially important, as different programming languages will be used, therefore it will not be possible to use the same SOAP implementation everywhere.

#### 7.4.2 WSDL descriptions of the services

Web services consist of a server and a client part. The client part is specific to the implementing programming language and can normally be generated automatically from a description of the ser-
vice provided by the service provider. This description is written in WSDL (Web Service Description Language) which is based on XML.

In general, there are two approaches to implementing a web service. The first is by starting with implementing the server and generate the WSDL description from it. The other way is to start by writing the WSDL and generating the server stub from it. In either way, a consuming client should be generated from the WSDL.

As the web service interfaces will be kept very generic (see section 6.4), the WSDL descriptions will also be very generic and will not reflect the actual parameters of the service. Therefore, each partner providing modules which are offering services, will have to generate appropriate XML schema definitions of the services besides the WSDL representation of the web service interface. This also applies to monitoring and managing services.

### 7.4.3 Underlying transport protocol

SOAP can make use of several underlying transport protocols. The preferred transport protocol is HTTP, which is used for transmitting web content in the internet. This renders it very compatible with existing network or firewall equipment. For the same reason it is widely spread and well accepted. As the SOAP communication is using text to exchange messages, HTTP is a very convenient protocol for web services. Because of its wide acceptence and usage, HTTP will also be used for the web services of the BOEMIE system. With regard to security issues, as described above, the secure extension of HTTP, HTTPS should be used.

### 7.4.4 Server requirements

On the server side, web services must be run within a suitable environment, called container. In most cases, the choice of the SOAP stack limits the choice of the usable containers. Nonetheless, as long as the implementations comply with the WS-I BP, operation should be independent from the container. This means all partners can use their preferred container and no special enforcement has to be made.