Studienarbeit

An Agent-oriented Architecture for Semantic Extraction and Ontology Evolution from Multimedia

submitted by

Kamil Sokolski

(21731)

Supervisor: Prof. Dr. Ralf Möller

Atila Kaya

Contents

Acknowledgements

Personally, I would like to deeply thank Atila Kaya, who helped me improve the quality of this work and also helped me debugging nasty Bugs.
Second I would like to thank Prof. Möller for making this work possible and showing me the potentials of agents.
I also would like to thank Thomas Hirsch for interesting inspirations on the work with Agents and my girlfriend Bettina Scholze for her support.

Chapter 1

Introduction

1.1   Motivation

This work investigates the applicability of state-of-the-art agent-oriented software development methodologies in developing the application logic of complex, distributed service oriented software systems. Furthermore it discusses the advantages and disadvantages of the agent-oriented software engineering process. To this end, agent-oriented software development is compared to the standard object-oriented software development.

1.2   The BOEMIE Project

We consider the development of an agent-oriented application logic in the BOEMIE project as an example for the development of a complex service oriented software system.

The BOEMIE project aims to acquire knowledge from multimedia content by evolving multimedia ontologies using the extraction of information from content in networked sources. The acquisition of knowledge happens in an ongoing semi-autonomous process called "Bootstrapping"

1.3   The BOEMIE Bootstrapping Process

The BOEMIE process has two major activities a) "Semantics Extraction", and b) "Ontology Evolution"

"Semantics Extraction" tries to extract information from multimedia docu-

ments. These documents can be text, images, audio, video files or even combinations of them like a website. Semantics Extraction itself has three sub processes "Analysis", "Interpretation", and "Fusion". During "Analysis" MLCs (mid level concepts: they contain low level information that can be extracted form a multimedia document e.g. a horizontal bar or a human body) are found in single modalities (one medium e.g. text). Furthermore relations between the MLC instances, (e.g. a human body near a pole) are found during analysis. "Interpretation" tries to compute explanations for MLC instances and their constellations and therefore hypothesizes HLCs (high level concepts), which are more abstract aggregate concepts, combinations of some MLCs. Due to the ambiguity involved in multimedia analysis, interpretation results may become ambiguous as well. Therefore the result of interpretation can be one single file when the interpretation is obvious according to the domain ontology, more then one file, when the interpretation is not clear and the medium could be interpreted in different ways (e.g. a horizontal bar and a jumper in the picture could be interpreted as a high jump or as a pole vault) or even no file at all, if no HLCs could be constructed, because there were too many unknown instances in the analysis phase or the ontology is not rich enough to explain the concepts. When a multimedia document has been analyzed and interpreted with respect to different modalities, then Fusion aims to identify common instances in modality-specific interpretations. (What is the goal of doing so? i) to gain more information which will enhance retrieval ii) to disambiguate information i.e. reduce multiple interpretations if possible.) Once Fusion is completed, the second step oft the BOEMIE process, the Ontology Evolution, starts.

Ontology Evolution, aims to add new concept instances and relations between concepts instances to the domain ontology, "Population", and to enrich the terminological part of the ontology if unknown objects were found during the analysis. This is called "Enrichment". Because changes in the ontology can influence the results of analysis and interpretation, the process of Semantics Extraction has to be repeated with the goal to extract new knowledge. Combining the two major steps namely semantics extraction and ontology evolution together in a cycle, leads us to an ongoing "Bootstrapping"-process. When something unexpected happens during the bootstrapping process (e.g. multiple interpretations were computed) the ontology expert is notified. This may be necessary e.g. if a new MLC concept has been introduced to the ontology. The ontology expert can interact with the system by starting a training process for the analysis tools. Once some cycles of the bootstrapping process occurs, the offline information will differ significant from the one in the live system. The ontology expert can initiate a "play out" of the offline ontology. Then the of-

fline information becomes the live information, and the end user has access to the new information. The offline-stage is going on to acquire new information.
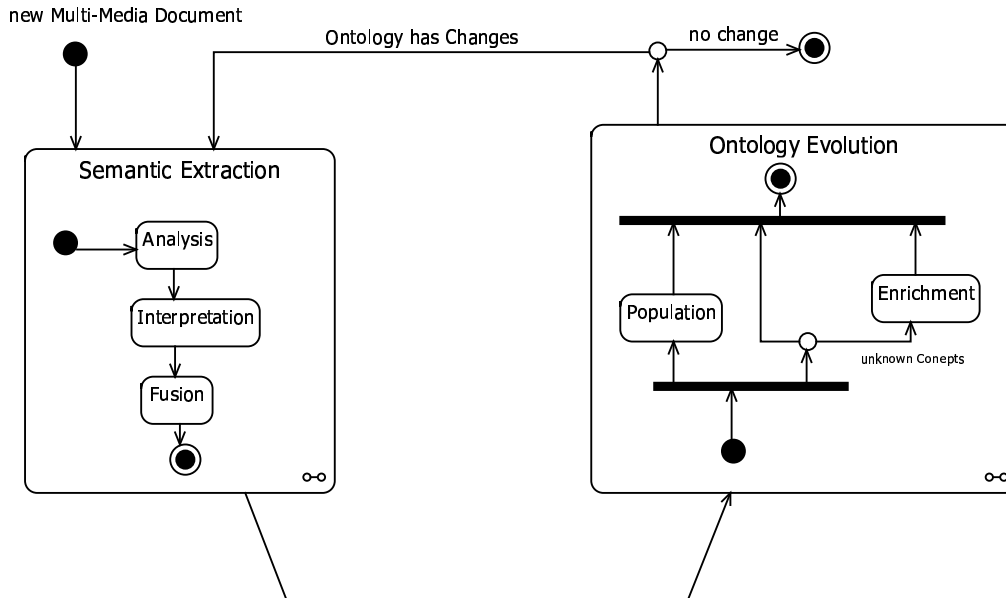

Figure 1.1: Bootstrapping State Diagram

1.4 The BOEMIE Architecture

The BOEMIE Software system has two staging processes. One staging process is the so called "offline-stage" where information is extracted from multimedia documents and stored in an offline "triple store"-database. The second mode is the so called "life-stage" where end-users can query information from the "life triple-store" through a GUI. I.e. end-users have no access to the "offline stage". The "offline-stage" process uses two major toolkits, the semantics extraction toolkit, and the ontology evolution toolkit for services necessary to solve a task. It is a goal of the project to organize the process as autonomous as possible. Only when the system is not able to decide by its own, witch step to take next or witch result to choose, a human expert is informed who then assumes control of the process. A human system operator is also informed when technical issues occur (e.g. one service could not be called). The steps of the BOEMIE semantics extraction (analysis, interpretation, fusion) and ontology evolution process (population, enrichment) are monitored by the ontology expert. For every step of the BOEMIE process there is a tool, which can be accessed

through a web service. The process is controlled by an application logic, which runs the tool for every process activity and checks the result. Depending on the results, the next activity is executed or the ontology expert is notified if something unexpected has happened.

Chapter 2

Analysis

2.1 Requirements for the BOEMIE Application Logic

To realize the process efficiently, we would like to have the system as autonomous as possible on the one hand, but on the other hand we would like to control the process, if there are more than one possible result to choose, or if unrecognized MCLs appear. The application logic should be developed in a modular way, such that not only the information of a web page (e.g. a HTML page) is extracted, but also the semantics extraction of other possible multimedia documents (e.g. blogs or rss feeds, video files etc.) would be possible, by reusing already existing modules. Scalability may also be an issue, because with a constantly changing and growing ontology, the time needed for analysis, interpretation and fusion may increase. Also the application logic should be able to manage several extraction requests at the same time. Therefore the application logic should be able to provide additional resources to the BOEMIE system on demand in order to make the system scalable.

2.2 The Preferred Solution

Since the "BOMIE -Bootstrapping process" has the characteristic of being semi-autonomous, It is reasonable to use a technology for the application logic that supports these characteristics by nature. These characteristics can be found in the software development paradigm referred to as **"agent-oriented programming"**.

Chapter 3

Design

3.1 Agent-Oriented Programming

Agent-oriented programming is based on object-oriented programming. Agents are modules of the software system and have the following properties:

- Autonomy: agents encapsulate some state and make decisions about what to do based on this state.

- Reactivity: agents are situated in an environment, are able to perceive their environment, and are able to respond to changes.

- Proactivity: agents are able to take the initiative (they act self-directed).

- Social ability: agents are able to communicate with other agents and to interact with them in competition are corporation.

3.2 Multi-agent Platforms

If multiple agents work together in a special environment, then such environments are called "multi-agent platforms". There are different multi-agent platforms implemented in different programming languages, but not every platform implements the FIPA-standards.
In this work we chose the Java Agent DEvelopment Framework (JADE) as a Multi-agent platform for the BOEMIE application logic, because it is flexible, the agent and behavior classes can easily be extended; it implements the FIPA-Standards, has ready to use classes for FIPA-communication-protocols,

and supports agent mobility.

JADE has been already used successful in some research and commercial projects. In the JADE framework, every agent- runs in a single thread and schedules his tasks by using a queue and a round-robin non-preemptive scheduler. This means that an agent can block one behavior on purpose to switch to another. It is useful, when an agent has to wait for a message. Then the agent blocks his message-receiving behavior and can execute a different behavior instead. When the expected message arrives the blocked behavior is automatically activated (JADE feature). All this multi threading concepts are implemented in the JADE-Framework and ready to use for the developer.

3.3   FIPA-Standards

Foundation for Intelligent Physical Agents (FIPA) is an international non-profit initiative that aims to produce specifications for the interoperability of agents created by different manufacturers with different technologies. Since they are focused on interoperability issues, the FIPA specifications do not deal with agents' internal structure. Instead the FIPA specifications define a language for communication between agents, called ACL (Agent Communication Language), and the methods of interaction with white and yellow page services that agents can use to find other agents to interact with, when trying to reach their goals. (The white page contains all agents that are on the platform and the yellow page contains all services that agents can process.) ACL relies on an asynchronous message exchange paradigm based on agents' names and consequently allows for "loosely-coupled" connections between components of the system (agents). Agents are completely independent of transport addresses and time frames (the receiver of a message might even not exist at the time it is sent).

Declarative interfaces permit agents to enter or exit the system, locate each other and link up dynamically so they can be distributed on multiple platforms or move from one agent platform to another agent platform. This anticipates many of the key aspects of service oriented architecture (SOA).

3.4   Software Development Process

The company "Whitestein Technologies" developed the Agent-oriented Development Methodology (ADEM).

ADEM is a comprehensive agent-oriented software development process for the specification, implementation, deployment, and operation of agent-based systems and applications. It is based on the Rational Unified Process (RUP), and integrates activities from the object-oriented development process. Additional activities are added to the RUP to support MAS development. (E.g. the "Base Business Modeling" is extended to the "Extended Business Modeling" by modeling intentions of business actors, workers, and organization units, modeling of business goals and responsibilities, modeling entity roles, communicative interactions, observations and effecting interactions, services, and mobility.)

ADEM suggest to model first the Business Goals and to derive the goals of agents form them. Second a Deployment model (shows witch agent is on deployed on witch agent platform), Business Services (they become the services of the agents later), agent interactions (communication diagrams), and the agent organization are modeled for the Analysis. For analysing the requirements the business goals are fragmented into smaller subgoals. During the design phase the agent society, agent interactions, agent services, the observation and the actions that effect the environment of agents, agent behaviors and mental states are modeled. Every model artifact can be visualized as a diagram in the Agent Modeling Language (AML).

ADEM has been published in "Issues in Multi-agent Systems" (November 2007) by the company, but did not yet become a common standard. At the time of writing, there are no standards to develop a good agent-oriented software development process. We follow ADEM as a guideline for analysis and design but differ in some process steps and leave out the suggested methods for implementation and software deployment.

3.5 Modeling a Multi-agent System

It is easy to find a modeling language and tool for an object oriented program. The unified modeling language (UML) is a common standard in the software development for object-oriented modeling. There are many graphical tools that support the UML 2.0 specifications.

But UML 2.0 is lacking the support of some of the key aspects that characterize agents (see 3.1). Internal and external goals, plans, beliefs, multi-life-lines, preceptors and effectors, services, agent environments and some more are not supported as default. To define these concepts with UML classes would take much time and would make a model really unclear and the resulting model would probably be involved.

All the above listed requirements can be found in the agent modeling language (AML).

The **Agent Modeling Language (AML)** is an agent-specific extension to the widely known UML 2.0. It is designed to support business modeling, requirements specification analysis, and design of software systems based on software agent concepts and principles. AML supports some important aspects of agent-oriented modeling (e.g. roles, goals, believes, plans and multiple lifelines etc.). Also special icons are defined for agent-specific concepts. Different aspects can be shown in different AML specific diagrams (e.g. a mental diagram shows the connection between goals, beliefs, and plans, a MAS deployment diagram shows which agent is deployed on which agent platform, an AML activity diagram has special icons representing sending and receiving agent messages.) The agent modeling language is well specified in the specification document.

AML is useful to build models that:

- consist of a number of **autonomous**, **concurrent** and/or **asynchronous** (possibly proactive) **entities**,

- comprise entities that are able to **observe** and/or **interact** with their environment,

- make use of **complex interactions** and **aggregated services**,

- employ **social structures**,

- capture **mental characteristics** of systems and/or their parts

There is an AML Profile module for **StarUML** (an existing UML-based CASE tool), where one can draw agent models by pointing and clicking.

3.6   Agents as a Software Module

In complex software system agents can be considered to be modules. One or more agents fulfill a certain goal. By giving an Agent only some tasks, who belong to one domain, we follow the concept of "separation of concerns". In this way the agents stay independent from each other and can be reused in different contexts. If an agent needs help from another agent in fulfilling his task, he can request an action by sending an ACL-message. The receiver of the message can accept it or reject it. This kind of communication makes the agents more flexible than between objects in an object oriented system.

The ACL supports multi-message communication to send an ACL-Message to some agents at the same time (e.g. for informing a group of agents about a certain state or for choosing an agent from a group of agents). Even if the receiver of the request does not exist, the ACL-message can be sent. The receiving agent may also be deployed on a different agent platform. This kind of communication makes it possible to start and stop agents at runtime or to move them to another platform.

Obviously, the same behavior can be implemented as part of an object oriented system, however it is easier to develop in an agent-oriented way by using a proven multi-agent platform and the functionality it offers out-of-the-box.

3.7 Mobility and Distribution

Because agents are independent from each other, it is easy to develop an application that is distributed on more than one agent-platform. E.g. when we have to develop an application logic that uses different web services (like in the BOEMIE-project), or a set of same web services that are implemented on multiple web servers, we can replicate our service agent and deploy it on another platform.

By using FIPA-standard protocols the agents can negotiate and select an agent that has to do the work. With this approach we get load balancing without any additional programming costs.

The same principle makes agent-mobility possible. E.g. one of our agents from BOEMIE application A could move to BOEMIE application B and work together with agents of application B.

Figure 3.1 shows an AML Deployment Diagram where an interpretation agent is moving form BOEMIE Agent Platform A to BOEMIE Platform B and interacting with agents on Platform B. The hat above an agent indicates that he is hosted by the displayed agent platform.
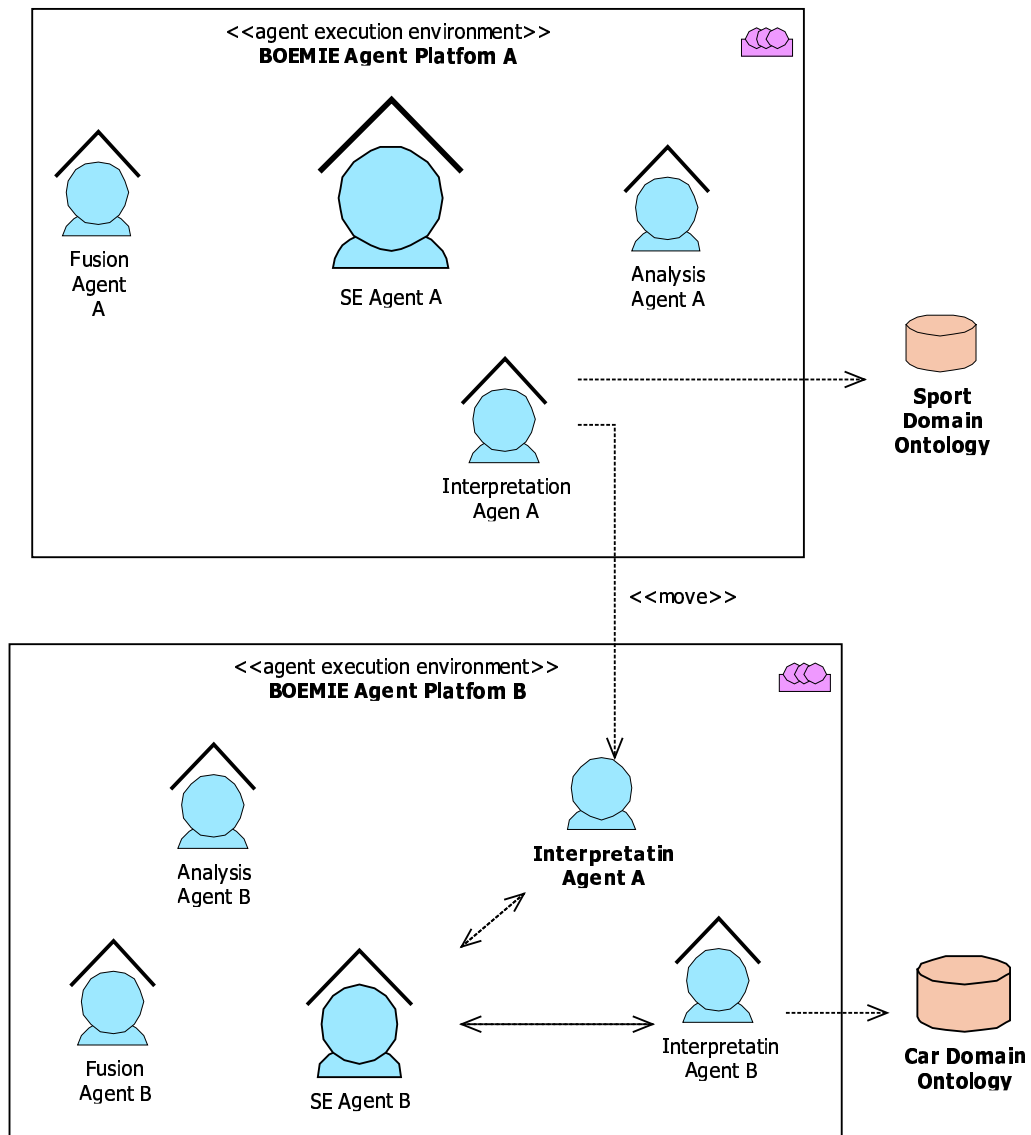
Figure 3.1: AML-Diagram: Model of a mobile Agent

Chapter 4

Implementation

4.1 The BOEMIE Application Logic

In the BOEMIE application logic we have two roles of agents. One of them is the role of a *coordinator*, who has a complex goal (e.g. satisfy the customer). To fulfill this goal, some smaller sub-goals have to be fulfilled. Therefore the coordinator has a complex plan which defines the sub-set of goals and the order of their execution. He himself does not have any plans to achieve these sub-goals, but he has plans to coordinate the work that has to be done. The other role is the role of a *servant*, who does not know anything about the complex goal, but only knows one of the sub- goals. The servant has a plan to achieve this goal and to report the result to the coordinator.

Both processes of the BOEMIE bootstrapping process can be modeled as a *finite state machine*. These state machines can be translated directly into a plan of the coordinator agent, because the JADE framework provides a behavior class for a finite state machine. Behaviors are the plans of agents in the JADE-framework. In finite state machine behavior (**FSM-behavior**) we register sub-behaviors as states and transitions from one state to another. E.g. the analysis state has a transition to the interpretation state if the result of the analysis was successful (this is indicated by an integer value returned from the sub-behavior), and a transition to the report failure state if the analysis has failed. This is an end state and the semantics extraction process terminates for that particular multimedia file.
In every state of the FSM the coordinator tries to assign a servant agent for

the particular work that has to be done.

By implementing the "Contract Net Protocol", the coordinator can negotiate with several servant agents, and decide which one of them is going to execute the job. In the Contract Net Protocol an initiator agent sends a request message (called: "call for proposal" (**CFP**)) to several responder agents. The content of the CFP message is a description of the work that the *initiator agent* wishes to be done. The *responder agent* now has the possibility to either refuse the request by sending a refuse-message or to send a proposal-message in witch he sends an offer that indicates him as a good candidate for the job (e.g. the amount of time the responder is going to need for the job). After having received all the responses of the CFP, the initiator chooses the responder with the best proposal and sends him the needed data for the job. Now the responder executes the job and responds the result in an inform-message or sends back a failure-message if he failed.

In the BOEMIE application logic the coordinator agents implement the behavior of a contract-net-initiator and the servant-agents implement the behavior of a contract-net-responder. E.g. the "Semantics Extraction Agent" (Coordinator) wants to have an analysis done of a website that contains an image, a caption, and a text section. He sends a CFP to all "Analysis Agents" (Servants), and requests the analysis of an image. All "Analysis Agents" respond by sending back the number of jobs they have in their queue. *Let us assume these responses: AnalysisAgent1: 1, AnalysisAgent2: 0, AnalysisAgent3: 0.*

The "semantics Extraction Agent" chooses either AnalysisAgent2 or AnalysisAgent3 and sends him the image data to analyse. *Let us assume he has chosen AnalysisAgent3.* While the AnalysisAgent3 is working on the image data, the "Semantics Extraction Agent" requests the analysis of the text section in the same way he did for image analysis.

Now the response of the Analysis Agents is: AnalysisAgent1: 1, AnalysisAgent2: 0, AnalysisAgent3: 1 . Of course the "semantics Extraction Agent" chooses AnalysisAgent2 to analyse the text section, because AnalysisAgent2 is the most performativ. In the exact same manner all other tasks are distributed to different servant agents, in every state of the semantics extraction process (analysis, interpretation, fusion). In this way the tasks are performed parallel.

The description facilitator (DF) is the yellow page service of the Multi-agent platform. It is an agent where other agents can register their services they can perform. (An agent can register more than one service at the DF.) As long as the agent registers with the description facilitator of the application logic, it does not matter where the agent remains. All local and remote agents, which

can perform certain service, can be found by requesting from the DF a list of
agents that have registered the service.

If every activity of the semantics extraction process (including fusion) has
been performed successful, the last state of the semantics extraction behavior,
the "InitiateOntologyEvolutionBehaviour" is reached. In this state the "se-
mantics Extraction Agent" informs the "Ontologies Evolution Agent" that new
information is available. Now the "Ontologies Evolution Agent" (coordinator)
is responsible, to manage population and if necessary enrichment of the ontol-
ogy. In the agent oriented application logic prototype, "Ontologies Evolution
Agent" advises a "Population Agent" to store the results of the fusion activity
in a knowledge database. The "Population Agent" first checks if the fused A-
Boxes are consistent with the ontologies that are already stored. If consistency
checking succeeds, he tries to store the fused A-Boxes in the "offline triple-store
database". (At this time the application logic prototype only checks whether
the A-boxes could successfully be stored. It would be useful to check if the
existing ontology has been really extended by new concepts.)
The "Ontologies Evolution Agent" is informed by the "Population Agent" about
the population results. When, at least one, storage was successful, the ontol-
ogy evolution agent informs the semantics extraction agent that the ontology
has changed. Now the semantics extraction agent repeats his work for all the
multimedia files he has already extracted. This concludes one cycle in the boot-
strapping process.
(Figure 4.1 shows the organisation structure of the BOEMIE Application Logic)

4.2   Enhancements for BOMIE

During this work some additional ideas for enhancements to BOMIE Application
Logic came up that could be investigated in research:

4.2.1   Probabilistic Semantics Extraction Agents

Using a learning algorithm for Analysis and Interpretation could provide the
resulted ontology instances with a probability value, which indicates how likely
they are in the given context.
    we could extend the agents, involved in the semantics extraction process,
in such a way that they do not consider instances of a likelihood value, that is

below a certain threshold. This method would lower the production of multiple documents in a autonomous semantics extraction process.

4.2.2 Ontology Enrichment

When unknown instances are found during the semantic extraction process a human ontologies expert needs to be involved to explain these instances. Having an ontology enrichment agent which is able to compare and merge the own ontology with other ontologies that are stored in the ontology repository or in other BOEMIE systems. This could be done by implementing an Ontology Evolution Agent by either using an enrichment service or implementing the ability to retrieve information about the taxonomies by querying a reasoner, to the agent. Integrating such an Agent to the existing Application logic would produce minimal additional costs, because of the flexibility and independence of agents, mentioned above.

4.2.3 Web Service Integration Gateway (WSIG)

At this time the semantics extraction process of the application logic prototype is initiated by a starting agent who sends an ACL-Message to the semantic extraction agent. With the JADE add-on "web service integration gateway" (WSIG) we could expose services provided by agents and published in the JADE DF as web services. This extension would make agent- services (e.g. semantic extraction of a web site) usable from the outside world.

WSIG is a web application composed of two main elements:

- WSIG Servlet

- WSIG Agent

The WSIG Servlet is the front-end toward the internet world and is responsible for:

- accepting incoming HTTP/SOAP requests

- Extracting the SOAP message

- Preparing the corresponding agent action and passing it to the WSIG Agent

Moreover once the action has been proceeded:

- Converting the action result into a SOAP message

- Preparing the HTTP/SOAP response to be sent back to the client

The WSIG Agent is the gateway between the Web and the Agent worlds and is responsible for:

- Forwarding agent actions received from the WSIG Servlet to the agents actually able to serve them and getting back responses from them.

- Subscribing to the JADE DF to receive notifications about agent registrations/deregistrations.

- Creating the WSDL corresponding to each agent service registered with the DF and publish the service in a UDDI registry if needed.
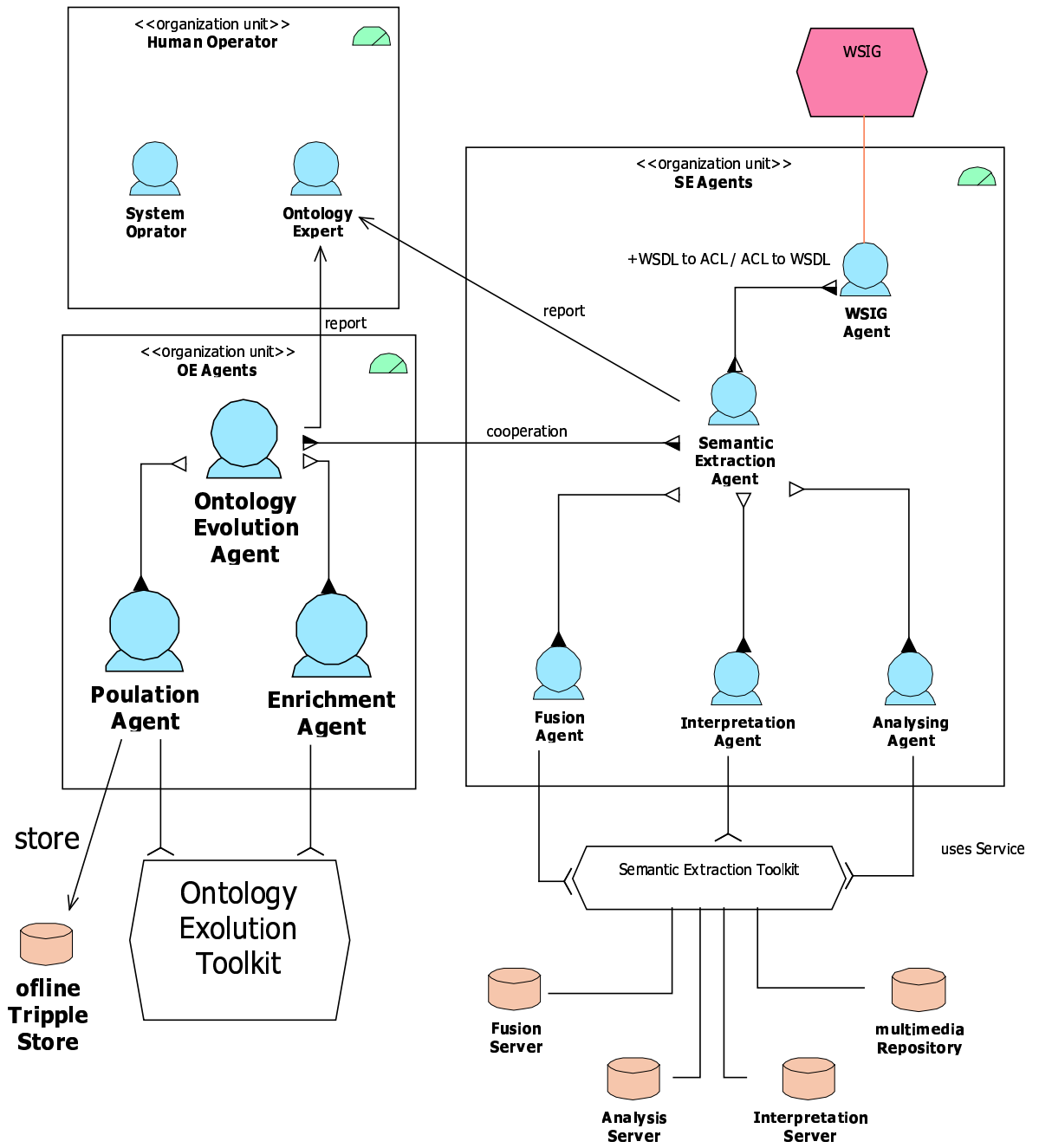
Figure 4.1: AML-Diagram: Organisation Structure of the BOEMIE Application Logic

Chapter 5

Conclusion and Outlook

5.1 Conclusion

We showed that agent-oriented programming has a lot of advantages when it comes to implement an automatic/semiautomatic process.

Following the development process proposed in ADEM worked fine as a guide line and together with models in AML we ware able to develop good models of the application during every step of the engineering process. Because of the similarity to social interaction and natural language speaking, developing a model of the system in The an agent oriented modeling language (like AML) was intuitive. The model helped to keep track of the system components and assign roles to agents (and so reduce code redundancy).

We used JADE as an agent platform that maintains FIPA-Standards to ensure a communication capability with other agent platforms and (extended) FIPA-Protocols (like the Contract Net Protocol) for inter-agent-communication that helps to avoid communication deadlocks and to cover all possible communication results. Because of useful JADE classes for agent behavior we had low effort to implement the complex BOEMIE Bootstrapping Process. The usage of the Agent-Paradigm made it possible to develop an application that is autonomous in every component.

5.2   Outlook

5.2.1   Interesting Extensions for Agent-oriented Systems

We discus now some interesting extensions to multi-agent systems that ware investigated during the project work and make a proposal for a project.

Believe Desire Intention
Believe Desire Intention (BDI) is a model for a bounded rational software agent, that views the system as a rational agent having certain mental attitudes of Belief, Desire and Intention (BDI) representing, respectively the information, motivational, and deliberative states of the agent. These mental attitudes determine the system's behavior and are critical for achieving adequate or optimal performance when deliberation is subject to resource bounds. [1]

- Beliefs represent the informational state of the agent, including itself and other agents, and may not necessarily be true.

- Desires (or goals) represent the motivation of an agent.

- Intentions represent the deliberative state of the agent, what he has choosen to do.

- Plans are sequences of actions that an agent can perform to achieve one or more of its intentions.

The Jadex reasoning engine follows the Belief Desire Intention (BDI) model and allows for programming intelligent software agents in XML and Java and can be deployed on different kinds of middleware such as JADE. It is a research project form Distributed Systems and Information Systems Group ,Computer Science Department, University of Hamburg, Germany.
URL: http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

Semantic Web Integration With Agents
FIPA defines ontologies as a representation of agent knowledge, but till now JADE support for ontologies or agent knowledge modeling is based on the Java classes. Agents can use Ontologies to communicate about structured information. The *JADEOWLCodec* lets agents (and MAS) communicate about ontologies described in Web Ontology Language OWL DL. It works with the DL-reasoner **RACER**.

Integration of JADEOWLCodec would make development of knowledge based agents with the ability

- to act depending on a state in the knowledgebase,

- to communicate information in description logic,

- to query a reasoner about ontologies

possible.

Project reference:
Friedrich-Alexander University Erlangen-Nuremberg
Department of Computer Science 8
91058 Erlangen
Germany

URL: http://www8.informatik.uni-erlangen.de/en/demosdownloads.html

5.2.2   Proposal for the Simulation of Social Systems

According to "Kommunikationsanschlüsse" [2] every social process can be broke down to single "communication signs" (the content of one singe communication act) connected with each other in a network. These communication signs influence the behaviors of a social system and social individuals more then mental states and individual believes.

Communications in ACL is comparable to communication acts from sociology and therefore particularly agents are suitable for modeling human communications. An agent that is able to extract the information of a single "communication sign" (content of the agent communication message in the form of an semantic ontology) and having the ability to do reasoning about the information in his personal knowledgebase as well as in his social context (social knowledgebase), could be developed in such a way, that he chooses his reaction dependent on the reception of the information.

This could be achieved by the integration of *semantic content to agent communication* by implementing the "JADEOWLCodec"; implementing a *personal knowledgebase* for every agent and a *social knowledgebase* witch can be accessed by every agent of the social system (MAS - Platform); and adept the concepts of Believe Desire Intention (BDI) to be dependent on states of the personal and social knowledgebase.

An approach like that would combine communication focused and agent focused social simulation and would open new ways of proving social theories.

A possible scenario may a complex economic simulation where not only the simple exchange of goods are simulated but also the global state of the economic system and the social dependencies of producers, organizations and consumers are considered.

Bibliography

[1] M. E. Bratman. Intentions, plans, and practical reason. Technical report, Harvard University Press, Cambridge, MA, 1987.

[2] Thomas Malsch. *Kommunikationsanschlsse. Zur soziolgischen Differenz realer und knstlicher Sozialitt.* VS Verlag, 2005.

[3] Dipl.-Inform. Dipl.-Phys. Ralph Depke. *Visuelle Modellierung agentenbasierter Systeme.* PhD thesis, Fakultt fr Elektrotechnik, Informatik und Mathematik der Universitt Paderborn, 2004.

[4] Radovan Cervenka and Ivan Trencansky. *Agent Modeling Language, Language Specification V0.9.* Whitestein Technologies AG, 2004.

[5] Agent-oriented development methodology for ls/ts. Technical report, Whitestein Information Technology Group AG (http://www.whitestein.com), 2006.

[6] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, and Giovanni Rimassa. Jade administrators guide. Technical report, TILab, 27-February-2003.

[7] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, and Giovanni Rimassa. Jade programmers guide. Technical report, TILab,Telecom Italia, 18-June-2007.

[8] Bernhard Schiemann and Ulf Schreiber. Owl dl as a fipa acl content language. Technical report, Artificial Intelligence Division, Department of Computer Science, University of ErlangenNuremberg.

[9] Boemie description of work. Technical report, SIXTH FRAMEWORK PROGRAMME INFORMATION SOCIETY TECHNOLOGIES, Octoberber 2005.

[10] Thomas Tikwinski (FHG/IAIS), Carsten Rosche (FHG/IAIS), George Paliouras (NCSR), and Alfio Ferrara (UniMi)and Atila Kaya (TUHH)and Vasileios Papastathis (CERTH). Boemie specification of the architecture. Technical report, National Centre for Scientific Research Demokritos (NCSR) Fraunhofer-

Gesellschaft zur Frderung 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), 28/02/2007.

[11] Cathrin Wei (cathrin.weiss@gmail.com). Jade und fipa. Technical report, Universitat des Saarlandes, Wintersemester 2006/2007.

[12] S. Petridis, N. Tsapatsoulis, D. Kosmopoulos, Y. Pratikakis, V. Gatos, S. Perantonis, G. Petasis, P. Fragou, V. Karkaletsis, K. Biatov, C. Seibert, S. Espinosa, S. Melzer, A. Kaya, and R. Moller. Boemie methodology for semantics extraction from multimedia content. Technical report, National Centre for Scientific Research Demokritos (NCSR) and Fraunhofer-Gesellschaft zur Forderung der angewandten Forschung e.V. (FHG/IMK) and Hamburg University of Technology (TUHH) and University of Milano (UniMi) and Centre for Research and Technology Hellas (CERTH) and Tele Atlas (TA), December 21, 2006.

[13] Thorben Ole Heins. Bedarfsgerechte subskription und fusion von informationskanalen mit metadatenannotationen. Technical report, TUHH.

[14] A. Rao and M. Georgeff. Bdi agents: from theory to practice. Technical report, The MIT Press. Cambridge, MA, USA, 1995.

[15] Jade web services integration gateway. Technical report, Telecom Italia, 2007.

Appendices

Appendix A

Appendix

A.1 Agent Platforms Implementing FIPA-standards

This section presents an overview, about some multi agent platforms that are implementing FIPA-standards and shows their characteristics.

- **SPYSE: http://sourceforge.net/projects/spyse**
  (Smart Python Simulation Environment)

  - *Programming language:* Python
  - Easy to use, nice for game theory

- **JACK: http://www.agent-software.com/shared/products/index.html**

  - *Programming Language:* Java
  - Commercial, BID-concept, team-based agent capability
  - *Customers:* defense and aerospace organizations (e.g. U.K. Ministry of Defense)
  - Project: "Human Variability in Computer Generated Forces" The project focuses on the representation of human behavior, and representation of the effects of external and internal moderating influences on the Computer Generated Forces (CGF) entity and unit behavior, in an effective and practical manner.

- **JADE: http://jade.tilab.com/**

  - *Programming language:* Java (Open source (LGPL))
  - developed by Telecom Italia to validate the FIPA specifications.
  - JADE is used in many commercial and academic R&D-Projects.

A.2 Projects Implemented In JADE

- E-Commerce Agent Platform (E-CAP)
  o Comprehensive agent-based e-commerce system realizing price negotiations and combining: (a) adaptability and intelligence (change strategy according to history of purchases), (b) mobility (based on transfer of negotiation modules and agents).

- Knowledge on Demand
  o To design and build a multi-role personalised learning platform using collaborative software agents as the integrating component and for the smart Knowledge Package (KP) finding function of the system. Different technologies, based on existing Learning Technology interoperability specifications are used for other aspects of the personalisation process.

- Pellucid
  o To develop a flexible and adaptable platform to assist organisationally mobile employees at middle and higher levels of public sector organisations.

- AgentCities
  o Agentcities is an initiative to create a next generation Internet that is based upon a worldwide network of services that use the metaphor of a real or a virtual city to cluster services.

List of Figures