Engineering JADEX Agents with the MaSE Methodology

Radziah Mohamad, Safaai Deris
Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
radziah@fsksm.utm.my, safaai@fsksm.utm.my

Hany H. Ammar
Computer Science and Electrical Engineering Department, West Virginia University, Morgantown, WV26506, USA
Hany.Ammar@mail.wvu.edu

Abstract

Agent Oriented Software Engineering (AOSE) is one of the fields of the agent domain with a continuous growing interest. The reason is that the possibility to easily specify and implement agent-based systems is of a great importance for the recognition of the add-value of the agent technology in many application fields. In this paper we present an attempt towards this direction, by proposing a kind of roadmap of how one can combine the MaSE methodology for agent-oriented analysis and design and JADEX, a FIPA compliant agent development framework, for an easier analysis, design and implementation of multi-agent systems. Our objective is realized through the presentation of the analysis, design and implementation phases, of a water treatment plant information management system.

1. Introduction

During the last few years, there has been a growth of interest in the potential of agent technology in the context of software engineering. This has led to the proposal of several development environments to build agent systems (for example Zeus [3], AgentBuilder [12], AgentTool [7] and RETSINA [13]) and software frameworks to develop agent applications in compliance with the FIPA specifications (for example FIPA-OS [10], JADE [2] and Jadex [3]). These development environments and software frameworks demanded that system analysis and design methodologies, languages and procedures would support them. As a consequence, many of these were proposed along with a methodology (e.g. Zeus [6], AgentTool [7]) while in parallel have been proposed some promising agent-oriented software development methodologies, such as Gaia [14], AUML [1], Tropos [4], MASE [8]. However, despite the possibilities provided by these methodologies, we believe that a further progress must be made, so that agent-based technologies realize their full potential, concerning the full covering of the software life cycle and the proposal of standards to support agent interoperability.

In this paper we present an attempt to use MaSE in order to engineer a multi-agent system (MAS) that is to be implemented with the JADEX framework. The only pretension we have with this paper is to share our experience to conceive and develop a MAS, by combining MaSE and JADEX, in the context of the Water Treatment Plant Information Management System (WTPIMS), with people who are interested in the development of real life agent-based systems. The MaSE methodology can be applied in a high level design. There is no given way to go from a MaSE model to a system implementation. Thus, the aim of this paper is to describe a kind of roadmap for implementing a MaSE model using the JADEX framework. Towards this end, we provide some additional modeling techniques and make some slight modifications to the MaSE original specification, without obviously altering its philosophy and concepts.

This paper is organized in the following way. In sections 2 and 3 we briefly present the MaSE methodology and JADEX framework. In section 4 we provide a sample MaSE model. In section 5 we provide a methodology for converting the MaSE model to a JADEX implementation. Finally, section 6 concludes.

2. MaSE Methodology Overview

MaSE methodology is a methodology for developing heterogeneous multi-agent systems. MaSE covers the complete lifecycle of the system, from the analysis to the design utilizing a number of graphically based models. The models are transformed into diagrams in order to describe system agents, their communications, and the internal structure of each agent detailed in depth.

MaSE is supported by a software engineering tool called AgentTool [7]. AgentTool allows the designer to formally specify all the MaSE models. It also supports automated design transformations and automatic verification of inter-agent communications.

There are two basic phases in the MaSE: analysis and design. The first phase, Analysis, includes three steps: Capturing Goals, Applying Use Cases and Refining Roles. The purpose of the Analysis phase is to produce a set of roles whose tasks describe what the system has to do to meet its overall requirements. MaSE assumes that the development process starts with a requirements specification which includes the whole set of well-defined requisites. Using the initial requirements of the system, in the Capturing Goals step, the goals are identified and structured into a Goal hierarchy diagram. Next, the Applying Use Cases stage identifies the use cases and creates the sequence diagrams to help to define an initial set of roles. These use cases are, like in UML, a basic scenario about the desired behavior of the system. Finally, the Refining Goals phase transforms the goals previously obtained into a set of roles. Together with roles, a set of tasks are created; tasks define the role behavior. This step implied the construction of a Role Model Diagram and several Concurrent Task Diagrams, each one specifying the role behavior for each task, using a finite state automaton.

The main aim of the design stage is to define the overall system organization by transforming the roles and tasks introduced during analysis into agent types and conversations. The design stage is composed of four phases: Creating Agent Classes, Constructing Conversation, Assembling Agent Classes, and System Design.

The first stage maps roles to groups of agent classes and creates the Agent Classes Diagram. Next, in Constructing Conversation, all the conversations are detailed. Description of each conversation needs two Communication Classes diagrams: one diagram for the initiator agent and another for the responder. In the phase of Assembling Agent Classes, the architecture used for agents is decided and then all the internal agent components are defined. Finally, it is in the System Design where the number, kind and location of each agent instance are shown in the Deployment Diagram.

3. JADEX Overview

Jadex [3] is a Java based, FIPA compliant agent environment, and allows to develop goal oriented agents following the BDI model. Jadex provides a framework and a set of development tools to simplify the creation and testing of agents.

Jadex framework consists of an API, an execution model, and predefined reusable generic functionality. The API provides access to the Jadex concepts when programming plans. Plans are plain Java classes, extending a specific abstract class, which provides useful methods e.g. for sending messages, dispatching sub goals or waiting for events. Plans able to read and alter the beliefs of the agent using the API of the belief base. In addition to the plans coded in Java, the developer provides an XML based Agent Definition File (ADF), which specifies the initial beliefs, goals, and plans of an agent. The Jadex runtime engine reads this file to instantiate an agent model, and executes the agent by keeping track of its goals while continuously selecting and executing plan steps, based on internal events and messages from other agents.

4. A MaSE Model

In order to better understand our proposal on how MaSE and JADEX can be combined to conceive and implement a multi-agent system (MAS) we will present a prototype of the water treatment plant information management system that is currently being implemented. WTPIMS is aimed to automatically generate daily, weekly and monthly quality reports for all of the Water Treatment Plant (WTP) plants. This research used Sungai Johor and Sungai Layang WTPs managed by Strategi Tegas Sdn. Bhd. as the case study. The reports related to an industry's water use seem infinite. A facility's reporting needs vary dramatically and include those reports required by regulation as well as those needed for process control management. The system will automatically generate reports based on a default schedule or set by the authorized person. The system will automatically notify the management for all the reports that are auto generated. The management then can access the automatically generated reports by choosing the report from the list of the available reports. The related reports will be e-mailed to the supervisor and CEO when they are created. We chose the MaSE methodology to analyze and design the WTPRMS application for its architectural independence, simplicity and supporting tool. This system is appropriate to use the agent approach since the report must be proactively provided to the plant staffs that manage the plants. Therefore, the system to be developed requires its components to show a high degree of autonomy. We will show how this system can be analyzed, designed and implemented.

4.1 Analysis Phase

After taking into account the functional requirements, the system goals have been identified. These goals are included in a Goal Hierarchy Diagram as depicted in a Figure 1. In this diagram the main goals were: schedule reports, generate reports, display reports and send notification. With an iterative process all the main goals were discomposed in sub-goals. For instance, the main schedule report has been divided into: schedule daily, schedule weekly and schedule monthly.

In the Role Model Diagram of the water treatment plant information management system, five roles were identified by the developer; Schedule manager, Report manager, Report Generator, Viewer and Notifier. To satisfy each goal, a role develops certain tasks. In Figure 2, the association between report generator role and task is detailed: producing report. Finally, all interactions involving tasks (communication protocols) are shown in the same figure. With these roles and tasks the designer defined the internal behavior of each role in Concurrent Task Diagram Models. Figure 3 depicts an example of concurrent task diagram model for scheduling a report.

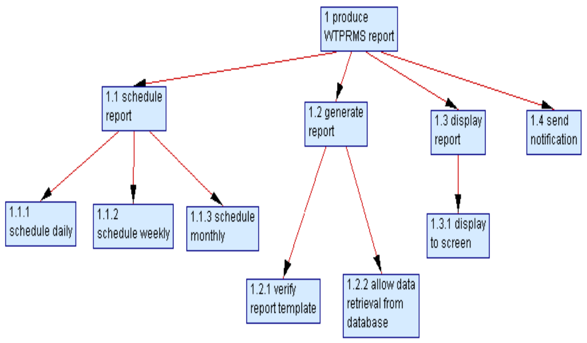

Figure 1 Goal Hierarchy Diagram of the WTPIMS System

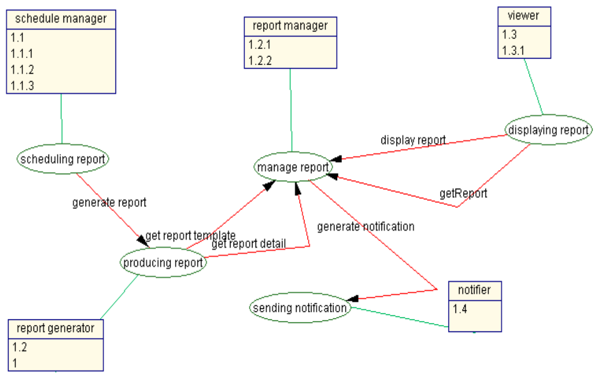

Figure 2 Role Diagram of the WTPIMS System

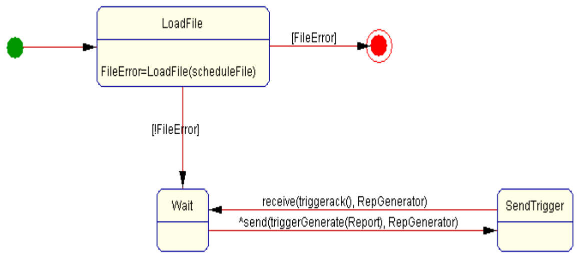
LoadFile
FileError=LoadFile(scheduleFile)
[FileError]
[!FileError]
Wait
receive(triggerack(), RepGenerator)
^send(triggerGenerate(Report), RepGenerator)
SendTrigger

Figure 3 Concurrent Task Diagram for scheduling the report

4.2 Design Phase

The final results of this stage are the Agent Classes Diagram that shows the overall agent system classes and conversations among them. Four agents have been identified: UserInterface, RepManager, RepGeneration and SchedManager. Figure 4 shows four agent classes, their associated roles and the corresponding conversations.

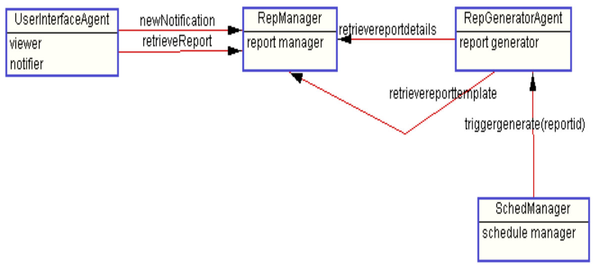

Figure 4 Agent Class Diagram

In MaSE, a conversation is a coordinator protocol between two agents and it documents a communication class diagram for the initiator and responder of each conversation by finite state machines. Figure 5 shows the "Scheduling Report state diagram" for the scheduler manager agent as an initiator. Figure 6 shows the "Scheduling Report state diagram" for the report generator agent as a responder.

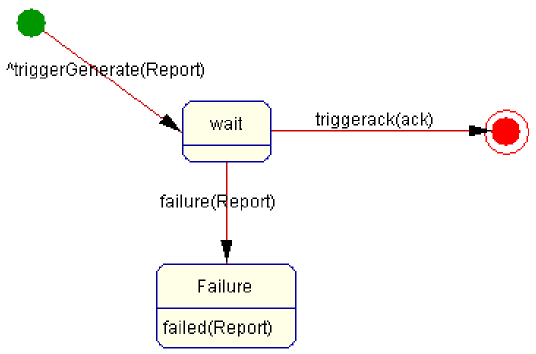

Figure 5 Scheduling Report Conversation (Initiator)

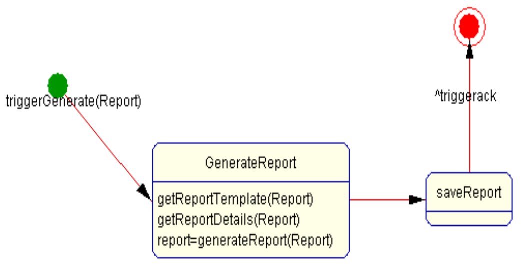

Figure 6 Scheduling Report Conversation (Responder)

5. Developing JADEX Agents from a MaSE Model

MaSE does not use BDI concepts throughout the whole development cycle. Therefore, when moving from the MaSE model to an implementation using the Jadex framework, some assumptions need to be made. In our work, the task of each role identified in MaSE analysis is considered as its plan to achieve its goals with correspondence with the Jadex terminology.

5.1 Detailed Design

Many important design issues have yet to be considered and covered while trying to implement a MaSE model with the Jadex framework. Some of them are: a) Agent Internal Architecture, b) Social Architecture and c) Communication.

Jadex is focused on the use of BDI-concepts for its agent internal architecture. Hence, these have to be supported by the MaSE. MaSE needs to be able to describe how goals (by which plan) can be achieved and which beliefs these plans need to access. For this, we propose a slight modification to the agent architecture model compared to the original agent architecture model presented in MaSE. We believe that this modification able to properly described BDI-concepts elements to be implemented using Jadex. Figure 7 shows an example of internal architecture of a scheduler manager agent (SchedManager).

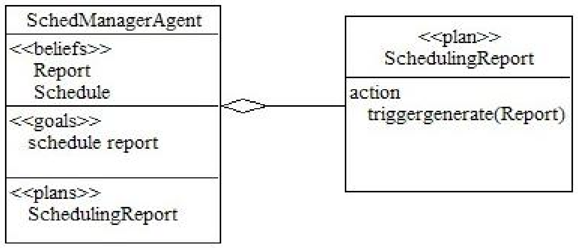

Figure 7 Scheduler Manager Agent Internal Architecture

MaSE uses roles as a concept to structure a MAS and to identify single agent classes. Roles are not explicitly supported by Jadex, they can be implemented using services. MaSE describes protocols between agents as the exchange of messages in relation to the processing inside the agents. We found that this representation is of the same suitability for Jadex.

5.2 The Jadex Implementation

The procedure is quite straightforward. All MaSE role tasks are translated to Jadex plan for the agent. Jadex uses the plan-library approach to represent the plans of an agent. For each plan, a plan head defines the circumstances under which the plan may be selected and a plan body specifies the actions to be executed. The most important parts of the head are the goals and/or events which the plan may handle and a reference to the plan body.

Figure 8 depicts an example of a plan head for the scheduler manager agent. The plan head is very simple in this case and consists only of the obligatory body expression that describes how a plan body is created at runtime and how it is triggered. As the trigger refers to the "*schedule_report*" goal type it is applicable for each goal instance of that type. The plan body, as depicted by Figure 9 is a Java class named "*SchedulingReportPlan*" that extends the Jadex framework class "*Plan*". Inside the mandatory *body()* method the plan create a message event to send a request to a report generator agent.

```
<plans>
<plan name="schedulewrite">
<body>new SchedulingReportPlan()</body>
<trigger>
   <goal ref="schedule_report"/>
</trigger>
</plan>
</plans>
```

Figure 8 Plan head example

```
public class SchedulingReportPlan extends Plan
{
   public void body()
   {
     try
     {
       :
       :
       while(true)
       {
         TriggerGenerate rc = new TriggerGenerate();
         IMessageEvent mevent =
             createMessageEvent("trigger_generate");
         mevent.getParameterSet(SFipa.RECEIVERS).addValue(report);
         mevent.setContent(rc);
         sendMessage(mevent);
                     :
       }
       :
       :}}
```

Figure 9 Plan body example

Each task identified in the role model and detailed in the concurrent task diagrams, has been used as basis for defining the details of each conversation. From the communication class diagrams of both the agent that initiates the conversation and the responder the interaction protocols that must be used were identified. For instance, when a report is scheduled to be generated, the scheduler manager agent makes a request for writing a report to its report generator agent using a FIPA-Request. This is implemented through a message event declared in the Scheduler Manager Agent as depicted in Figure 10.

In the ontology development, the facilities of Jadex relating Protégé integration using Beanynizer Generator tool have been exploited. The basic elements of the defined ontology have been:

- Concepts: User, Report, Schedule.
- Actions performed by agents: TriggerGenerate, RetrieveReportTemplate, RetrieveReportDetails, RetrieveReport, Notify.
- Predicates: SystemUsers.

To create and start an agent in Jadex, the system needs to know the properties of the agent to be instantiated. A complete definition of an agent is defined in a XML based Agent Definition File (ADF). The initial state of an agent is determined among other things by the beliefs, goals, and the library of known plans. Figure 11 depicts part of the scheduler manager agent definition file. Based on the example, the initial belief for the agent consists of three objects namely the reference to the report generator agent, the report and the timer to trigger a message to be sent to the report generator agent.

```
<messageevent name =
"trigger_generate" type="fipa"
direction="send">
<parameter name="performative"
class="String" direction="fixed">
       <value>SFipa.REQUEST</value>
</parameter>

<parameter name="content-class"
class="Class" direction="fixed">
<value>TriggerGenerate.class
</value>
</parameter>

<parameter name="ontology"
class="String" direction="fixed">
<value>wtprms.ONTOLOGY_NAME
</value>
</parameter>
</messageevent>
```

Figure 10 Declaration of a message event in the scheduler manager agent

```
<agent xmlns="http://jadex.sourceforge.net/jadex"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://jadex.sourceforge.net/jadex
                          http://jadex.sourceforge.net/jadex-0.94.xsd"
       name="SchedManager"
       package="wtprms.scheduler">

       <imports>
              :
       </imports>
       <capabilities>
              :
       </capabilities>

       <beliefs>
              <belief name="report" class="AgentIdentifier"/>
              <belief name="rep" class="Report"/>
              <belief name="timer" class="long">
                     <fact>System.currentTimeMillis()</fact>
              </belief>
       </beliefs>

       <goals>
              <achievegoal name="schedule_report">
              <creationcondition> $beliefbase.timer%86400000==0
              </creationcondition>
              </achievegoal>
       </goals>

       <plans>
              <plan name="schedulewrite">
                     <body>new SchedulePlan()</body>
                     <trigger>
                            <goal ref="schedule_report"/>
                     </trigger>
              </plan>
       </plans>
```

Figure 11 Extracted Scheduler Manager Agent Definition File example

Summarizing, the following steps should be followed in order to easily translate a MaSE model to a Jadex implementation:

1. Define all plans for an individual agent using the tasks identified in the MaSE role model. Based on

the role model, we can know which goals and sub-goals that the plan needs to achieve.

2. Define the details of each agent conversation by using the MaSE role model and its detailed concurrent task diagrams.
3. Identify both the agent that initiates the conversation and the responder and the interaction protocols to be used by using the communication class diagram of both the initiator and responder agent.
4. Define the XML based Agent Definition File by using the improved MaSE internal architecture diagram.

6. Conclusion

This paper has presented the analysis, design and implementation phases of a Water Treatment Plant Information Management System. As mentioned before, the only pretension we have with this paper is to share our experience on how one can combine the MaSE methodology and the JADEX development environment in order to implement a real multi-agent system. MaSE methodology is an easy to use agent-orient software development methodology that however presently, covers only the phases of analysis and design. On the other hand JADEX is a FIPA specifications compliant agent development environment that gives several facilities for an easy and fast implementation. Our aim was to reveal the mapping that may exists between the basic concepts proposed by MaSE for agents specification and agents interactions and those provided by JADEX for agents implementation, and therefore to propose a kind of roadmap for agents developers.

7. References

[1] Agent UML: http://www.auml.org/

[2] Bellifemine, F., Caire, G., Trucco, T., Rimassa, G., "JADE: A FIPA-compliant agent framework", in *Proceedings of the Practical Applications of Intelligent Agents and Multi-Agents*, April 1999, pp. 97--108.

[3] Braubach, L., Pokahr, A., Lamersdorf, W.: Jadex: A Short Overview, in: Main Conference Net.ObjectDays 2004, AgentExpo.

[4] Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J and Perini, A.: "TROPOS: An Agent-Oriented Software Development Methodology" in Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers Volume 8, Issue 3, Pages 203 - 236, May 2004.

[5] Collis, J. and Ndumu, D., Zeus Technical Manual. Intelligent Systems Research Group, BT Labs. British Telecommunications. (1999)

[6] Collis, J. and Ndumu, D., *Zeus Methodology Documentation Part I: The Role Modelling Guide*. Intelligent Systems Research Group, BT labs. British Telecommunications (1999).

[7] DeLoach S. and Wood, M., "Developing Multiagent Systems with agentTool", in: Castelfranchi, C., Lesperance Y. (Eds.): Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop (ATAL 2000, Boston, MA, USA, July 7-9, 2000),. *Lecture Notes in Computer Science*. Vol. 1986, Springer Verlag, Berlin (2001)

[8] DeLoach, S. A., Wood, M. F. and Sparkman, C. H.: "Multiagent Systems Engineering", *The International Journal of Software Engineering and Knowledge Engineering*, Volume 11 no. 3, pp. 231-258, June 2001.

[9] FIPA specification XC00061E, FIPA ACL Message Structure Specification (2000) http://www.fipa.org.

[10] FIPA-OS: A component-based toolkit enabling rapid development of FIPA compliant agents: http://fipa-os.sourceforge.net/

[11] Kendall, E.A., "Role Model Designs and Implementations with Aspect Oriented Programming", in the Proceedings of the Conference on Object- Oriented Programming Systems, Languages, and Applications (OOPSLA'99), 1999.

[12] Reticular Systems Inc: AgentBuilder An Integrated Toolkit for Constructing Intelligent Software Agents. Revision 1.3. (1999) http://www.agentbuilder.com

[13] Sycara, K., Paolucci, M., van Velsen, M. and Giampapa, J., "The RETSINA MAS Infrastructure." Technical report CMU-RI-TR-01-05, Robotics Institute, Carnegie Mellon University, March, 2001.

[14] Wooldridge, M., Jennings, N.R., Kinny, D., "The Gaia Methodology for Agent-Oriented Analysis and Design." *Journal of Autonomous Agents and Multi-Agent Systems* Vol. 3. No. 3 (2000) 285-312