Graphical Tool to integrate the Prometheus AEOlus methodology and Jason Platform

Rafhael R. Cunha\textsuperscript{1}, Diana F. Adamatti\textsuperscript{2}, Cléo Z. Billa\textsuperscript{2}

\textsuperscript{1}Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS) – Campus Vacaria
Rua Eng. João Viterbo de Oliveira, 3061 – Vacaria – RS – Brasil

\textsuperscript{2}Centro de Ciências Computacionais (C3) – Universidade Federal do Rio Grande (FURG)
Avenida Itália km 8 Bairro Carreiros – Rio Grande – RS – Brazil

rafhael.cunha@vacaria.ifrs.edu.br, \{dianaada, cleo.billa\}.gmail.com

Abstract. Software Engineering (SE) is an area that intends to build high-quality software in a systematic way. However, traditional software engineering techniques and methods do not support the demand for developing Multiagent Systems. Therefore a new subarea has been studied, called Agent Oriented Software Engineering (AOSE). The AOSE area proposes solutions to issues related to the development of agent oriented systems. There is still no standardization in this subarea, resulting in several methodologies. Another predominant factor for the instability of this subarea is the limited supporting tools available. The main purpose of this paper is to present an Eclipse plug-in to support Prometheus AEOlus methodology. Additionally, we have created a mechanism that is able to automatically generate code to AgentSpeak language, which is the base language of the Jason platform.

1. Introduction

In artificial intelligence, the agents oriented paradigm has been researched and used to minimize complexity and to increase the efficiency of distributed software. This practice is widely used to software development with these characteristics, boosting the Multiagent Systems (MAS) area.

In this context, several methods have been proposed in order to attend the demand of agents oriented software [Padgham and Winikoff 2002] [Bresciani et al. 2004] [DeLoach 1999] [Henderson-Sellers and Giorgini 2005] [Caire et al. 2002] [Cossentino and Potts 2002] [Wooldridge et al. 2000]. Such methodologies have been created for several reasons. However, one of these is able to address all dimensions of a MAS, as stated by [Demazeau 1995].

The Prometheus AEOlus methodology [Uez 2013] was proposed as an alternative to model agents based on the BDI architecture (Beliefs, Desires and Intentions) [Dennett 1989][Bratman 1987]. Furthermore, this methodology supports the modeling of organization and environment dimensions, unlike other ones. As a final goal, the Prometheus AEOlus methodology promises to be compatible to JaCaMo framework [Boissier et al. 2013].

The JaCaMo framework integrates three different development platforms, but complementary to develop MAS. The “Ja”, comes from Jason, and it consists of a platform for developing and running software agents based on BDI architecture and the AgentSpeak language. The “Ca” represents the CartAgO platform [Ricci et al. 2011],
that is responsible for programming environment artifacts, and “Mo”, symbolizes Moise+ [Hübner et al. 2002], for programming multi-agent organisations.

The Prometheus AEOlus methodology adds diagrams and notations to model other dimensions presented in a MAS. And it aims to provide a meta-models to allow code generation for the three development platforms that compose the JaCaMo framework. However, until now there is not a tool to support this methodology. Therefore, the goal of this paper is to present a graphical tool that supports Prometheus AEOlus methodology, as well as a mechanism to automatically generate code to the Jason development platform. Firstly, we have developed the interconnection to the Jason platform. As future work we intend to develop the association with the other platforms that compose the JaCaMo framework.

This paper is structured as follows: In the section 2 is presented theoretical and technological concepts for understanding the content of this work. In the Section 3 we show the related methodologies, as well as a comparative study to demonstrate the reason of this work. In the section 4 we discuss the general architecture of the plug-in developed, as well as the steps taken to completion. In the section 5 is presented a case study that aims to demonstrate the use of the developed platform. Finally, in the section 6 is showed the conclusions of the paper.

2. Theoretical-technological foundation

This section presents the concepts on agents and multi-agent systems, points out the origin and objectives of AOSE, and reports about plug-in development for the Eclipse IDE programming.

2.1. Agents and Multiagent Systems

According to [Weiss 1999], an agent is commonly described as a computer system that is located in an environment and can be virtual or not, interacting with this via sensors and actuators. Additionally, this agent is able to make decisions and take action autonomously in order to reach their goals. [Russell and Norvig 2009] explained that an agent is any object that can perceive the environment through sensors and acting on it through actuators. According to [Wooldridge and Jennings 1995], an agent is a computer system located in an environment, and is capable of autonomous action on the environment in order to achieve their stated objectives.

[Briot et al. 2001] define a Multi-Agent System (MAS) as an organized set of agents. This organization is responsible for setting rules for agents can coexist in a common environment, sharing resources and working collectively. [Demazeau 1995] proposes the division of a MAS into four main components: the Agent, the Environment, Interaction, and Organization. The used vowels are A, E, I, O, respectively to represent these dimensions, making this division known as a paradigm of vowels.

2.2. Agent Oriented Software Engineering

Agent Oriented Software Engineering (AOSE) has been developed to help the development of complex systems. This subarea merges Artificial Intelligence and Software Engineering areas to support the development of agents oriented systems [Guedes 2012].

According to [Gleizes and Gomez-Sanz 2011], the AOSE area has been raising for two main reasons: (i) the conceptual structures have reached a maturity level where its viable to spend efforts to find a consensus between modeling languages; (ii) the influence of model-oriented engineering emphasizes the potential value of having models in the center of the development process.
2.3. Eclipse Plug-In Development

Eclipse platform is based on plug-ins that are used to extend the functionality of Eclipse IDE [Foundation 2012]. These plug-ins are coded in Java programming language and they can offer several service modalities, as code library, documentation guides, and an extension of the platform itself [desRivieres and Wiegand 2004].

According to [Foundation 2014], the Graphical Modeling Framework (GMF) is a framework for developing graphical editors for domain models within the Eclipse platform. It was based on two frameworks called Graphical Editing Framework (GEF), used for creating generic graphic editors and Eclipse Modeling Framework (EMF), which enables developers to build metamodels and generate Java code referred to it. The graphics of this work have been developed using such framework.

3. Related Work

According to [Padgham and Winikoff 2002], Prometheus is a methodology for development of intelligent agents that differs from the other to be detailed and comprehensive, covering all necessary activities for the development of intelligent agent systems. The available tool for Prometheus is currently called Prometheus Design Tools (PDT) [Thangarajah et al. 2005]. PDT enables users to create and modify projects developed with Prometheus methodology. This tool ensures that there are no certain inconsistencies, using wrong diagrams notations, in addition to exporting individual diagrams and generate a report with the complete project. The methodology does not take into account the development platform to the phase of detailed design. However, it allows code to be generated for JACK language.

According to [Bresciani et al. 2004], the Tropos methodology [Mylopoulos et al. 2001] [Bergenti et al. 2004] [Cossentino et al. 2005] aims to support all the activities of analysis and project development to agents oriented software. It was developed based on the i* framework, which allows to model the multiagent system based on the concepts of actor, object, and dependency among actors. According to Cossentino et al. [Cossentino et al. 2005], the Tropos methodology allows direct mapping of modeling for JACK language. However, using the TAOM4E tool also can automatically generate code for JADEX language [Morandini et al. 2008].

The Multiagent Systems Engineering (MASE) methodology was proposed by DeLoach [DeLoach 1999]. It is a methodology developed for analysis and multiagent systems design. MASE uses the abstraction provided by MAS to help designers to develop distributed intelligence systems software [Henderson-Sellers and Giorgini 2005]. According to Henderson and Giorgini [Henderson-Sellers and Giorgini 2005], MASE is built on oriented techniques to existing objects. However, it is specialized for the MAS project. To help designers to use the methodology was built a tool called agentTool. This serves as a validation platform for MASE. The tool is based on graphics and fully interactive [DeLoach et al. 2001]. However, the tool does not have support for code generation to any development platform.

The Multiagent Systems Engineering (MASE) methodology was proposed by DeLoach [DeLoach 1999]. It is a methodology developed for analysis and multiagent systems design. MASE uses the abstraction provided by MAS to help designers to develop distributed intelligence systems software [Henderson-Sellers and Giorgini 2005]. According to [Henderson-Sellers and Giorgini 2005], MASE is built on oriented techniques to existing objects. However, it is specialized for the MAS project. To help designers to use the methodology was built a tool called agentTool. This serves as a validation platform for any development platform.  

1http://agenttool.cis.ksu.edu/
MASE. The tool is based on graphics and fully interactive [DeLoach et al. 2001]. However, the tool does not have support for code generation to any development platform2.

According to [Henderson-Sellers and Giorgini 2005], the Ingenias methodology provides a notation for MAS modeling through a well-defined collection of activities to guide the development process. Specifically, the task analysis, design, verification and implementation. These steps are supported by an integrated set of tools called Ingenias Development Kit (IDK).

According to [Caire et al. 2002], MESSAGE takes UML as a starting point and add agent, role, and task concepts to attend the needs of multiagent systems. According to [Uez 2013], this methodology proposes the analysis and design of a MAS based on five points of view: organization, objectives/tasks, agents/roles, interaction, and domain. However, such methodology does not have any tool.

According to Cossentino and Potts [Cossentino and Potts 2002], Process for Agent Societies Specification and Implementation (PASSI) is a methodology to design and develop multiagent societies integrating design models and concepts of software engineering and artificial intelligence, using the UML notation. Furthermore, PASSI is based on the FIPA architecture for modeling agents. To support the model code, there is the PTK tool (PASSI Toolkit). The tool also provides standard libraries that can be used for code generation and allows system testing before deploying software. The method allows the generation of code for the JADE framework.

According to Uez [Uez 2013], the Prometheus AEOlus methodology was developed based on two technologies: the Prometheus methodology [Padgham and Winikoff 2005] and JaCaMo framework [Boissier et al. 2013]. The main purpose of this methodology is create an extension of Prometheus methodology to include Environment and Organization dimensions.

The development process defined in this methodology is divided in four

---

2http://agenttool.cis.ksu.edu/
phases: system specification, architectural design, detailed design, and implementation [Uez 2013]. The first phase aims to specify the settings and system objectives. In the architectural design phase, the elements of the system are defined. After, the detailed design phase aims to define the internal structure of the agents through their beliefs, plans, and capabilities [Uez 2013]. The last phase is the implementation phase, and it has the goal to generate code for the JaCaMo framework. The Prometheus AEOlus methodology [Uez 2013] was based on the MAS division proposed by Demazeau [Demazeau 1995], he divides MAS in four dimensions: agents, interactions, environment, and organization.

According to [Uez 2014], Prometheus AEOlus presents the following specification activities: 1) scenarios; 2) objectives; 3) shares; 4) perceptions; 5) roles; 6) organizational structure; 7) missions; 8) norms; 9) environment; 10) interactions; 11) agents; 12) plans. In each of these activities, which can be developed over an interaction during the four phases of development, at least one diagram or descriptor is created. However, the methodology does not have a graphical tool to support its use.

3.1. Analysis of related work

Table 1. Methodologies x development platforms that automatically generate code from a methodology tool.

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Support Tool</th>
<th>Development Platform</th>
<th>MAS Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prometheus</td>
<td>PDTools</td>
<td>Jack</td>
<td>Agents</td>
</tr>
<tr>
<td>Tropos</td>
<td>TAOM4E</td>
<td>Jadex e Jack</td>
<td>Agents</td>
</tr>
<tr>
<td>MaSE</td>
<td>AgentTool</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ingenias</td>
<td>IDK</td>
<td>Jade</td>
<td>Agents</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PASSI</td>
<td>PTK</td>
<td>Jade</td>
<td>Agents</td>
</tr>
<tr>
<td>GAIA</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Prometheus AEOlus</td>
<td>-</td>
<td>JaCaMo</td>
<td>Agents, Environment, Organization</td>
</tr>
</tbody>
</table>

In the Table 1, Methodology and Support Tool columns show the methodologies studied in this work and the graphical tools that support their use, respectively. According this table, it shows that there are many graphical tools for modeling MAS and although each technique has its own peculiarities, it is known that same pattern beyond an MAS agents dimensions. Therefore, from this comparison, it follows that the main problem regarding the specification/implementation of a MAS consists in processing the work products³ of code developed automatically.

Regarding to the code generation, that aims to transform the specifications set out in the work products in source code. Table 1 also shows the development platforms that allow to generate automatic code, besides the dimensions of the MAS that this generation supports, through the Development Platform Dimensions and MAS columns, respectively. Thus, it is shown through the table, the preponderance for code generation for Jade and Jack frameworks, with the exception of Tropos also generates code for JADEX platform. However, analyzing them, it is clear that all focus their implementation within agents, causing loss of other information present in the work products designed previously.

³Work Product is the result of the completion of a work that serves as a resource to achieve some goal.
Based on above, only one methodology has in its core the concern with the agent, environment and organization dimensions is the Prometheus AEOlus, through the framework JaCaMo [Boissier et al. 2013]. Because of the peculiarity found this methodology, we decided to develop a graphical tool to support it.

### 4. A Platform to Support Prometheus AEOlus Methodology

This section presents the general architecture to develop the plug-in as well as the steps to implement it.

#### 4.1. General Architecture

The general architecture of Prometheus AEOlus tool is illustrated in Figure 1. Each package is a generated plug-in function to incorporate the tool developed. The package called generation is responsible for generating tool code. Within this package are grouped classes responsible for carrying out this task.

![Figure 1. General Architecture of Prometheus AEOlus Tool.]

The diagram package holds the classes responsible for the graphic editor. It contains all the configuration files and the codes generated by all from the GMF Eclipse development framework. Unlike other plug-ins that make up this tool, graphic editor does not have an initializer, resulting in dependence on the plug-in wizard to boot.

The wizard package is responsible for making the connection among the other two plug-ins that make up the tool developers. It combines the features described for other plug-ins and hence have dependency among them. In this plug-in, there is no source code, but only extension points in the eXtensible Markup Language (XML) configuration, where the dependencies to other plug-ins are reported.

#### 4.2. Development of Agent Overview Diagram

In the Section 3, the Prometheus AEOlus methodology specifies at least twelve diagrams to the development process. However, was decided to explain the development only the overview diagram agents. According to Uez [Uez 2014], the agent overview diagram describes the agent internally, i.e., this diagram details plans, skills, and beliefs of an agent, as well as the capabilities. The plan indicates the actions that the agent must perform to achieve a goal. Each plan must have an event trigger that will start it. Events triggers could be messages or environment perceptions. An event is linked to a plan by a dotted arrow. As mentioned by Uez [Uez 2014], the plans may include messages, actions, beliefs, or perceptions. All these elements are connected to a plan through simple and solid arrows, and they are connected by dotted arrows to indicate the order in which these actions should occur.

Figure 2 shows the six steps necessary to design the graphical editor referring to the overview diagram of the agent. This editor is designed using the GMF Eclipse described in Section 2.3. The development process shown in Figure 2 follows the specified
stream Eclipse Foundation [Foundation 2014]. In the first step, the meta-model is developed diagram, i.e., classes and attributes are defined that represent the rating that contains the diagram. Examples of class are: Action Message Plan, among others. The second step is to generate the skeleton code relating to metamodel developed previously. The third step is to create a derivative metamodel file, which is the graphic editor of the design field diagram in question, and all the figures used in this field are specified by geometric points and represent their classes specified in the metamodel earlier development. These steps are from the part of the EMF GMF Eclipse plug-in.

The fourth step of the development process is the graphic palette specification that will be used to chart component choice that should be plotted in the drawing field previously produced. The palette is also described based on the metamodel, and images for each metamodel class are developed externally. The fifth step is the combination of all previously created files. Finally, the sixth stage is the generation of a file that will contain all the information specified above, plus some configuration parameters needed to add extra functionality to the editor. This file can then be used to generate the code needed for the implementation of graphic editor referring to the diagram developed. These steps are related to GEF, part of GMF Eclipse plug-in.

4.3. Development of The Code Generator Plug-In

The Eclipse EMF plug-in uses a derivative of XML file, called XML Metadata Interchange (XMI) to make data persistence, as shown in Eclipse Foundation [Foundation 2014]. This file contains several tags to make the process of information extraction more accessible. In the process of code generation for Jason platform, we used these XMI files generated by GMF. These files contain information regarding about the General Model of the Prometheus AEOlus methodology, i.e., the diagram that combines together all entities and relationships of the methodology. Because of implementation issues, we developed a general diagram to enable the modeling of all entities of the methodology, to the code generation process to be more accessible for this first prototype.

The Code Generator plug-in is divided into four layers, show is Figure 3. The layers are named in: Model, Generator, View, and Service. The first layer is responsible for representing objects through the entities used by the diagrams of Prometheus AEOlus methodology. In addition, this layer also has classes that help the plug-in to find
Figure 3. Architecture of code generator plug-in

information presented in the XMI file. The Generator layer includes the classes that centralize the plug-in tasks in the Eclipse environment, and the classes that write the string with the code that must be stored in the file the plug-in will generate later. The Service layer is responsible for reading the XMI file generated by the graphical editor plug-in, turning it into objects, and return them as a hashmap. The View layer aims to manage the use of services and code generation. Through methods from the classes of this layer, the developer can enable the functionality of the plug-in. Activation occurs through events that are triggered by users that use it.

5. Case Study - Build a House

This section aims to demonstrate the use of the developed solution. The validation tool occurred on two different ways. The first aimed to perform unit tests to evaluate the efficiency of the code generator plug-in, validating its architecture and evaluating its efficiency. However, this first form of validation is not addressed in this paper, because it is not the focus of this work. The second validation method presents the use the plug-in graphic editor, in order to demonstrate its effectiveness in enabling users to transcribe integrally the diagrams described in [Uez 2013]. For this reason, we used the work products produced in this same work, but transcripts in our computational tool.

In [Boissier et al. 2013], the case study called “Build a House” was presented. In the case study, a character called Giacomo wants to build this house. He needs to hire companies that can perform tasks related to construction and coordinate the work of these companies. Hiring will be done through a bidding process in which the firm submitting the lowest price for the job will be hired. This case study aimed to demonstrate the use of JaCaMo framework, and therefore, this work has also been underutilized.

To demonstrate the use of the graphic editor, we chose to model computationally Giacomo Agent, which is the main character of the case study in question. Figure 4 shows the Overview Diagram of the Giacomo Agent. This is one of the diagrams of the scenario. We model other diagrams, but they are not in this paper because there are no space to it. In Figure, the Giacomo entity is modeled as an agent. The entities buildHouse, notifyWinner, hire and, auctionStart represent plans that make up the agent. The entities buildHouse, build, hire, auctionWinner, notifiesWinners and, startAuction represent messages sent or received by the plan or the perception of them. The entities wait, lookArt, createArt and, nameArt represent the actions to be performed during the plan implementation. Finally, the entities task and winner represent the beliefs related to their plans.

The Figure 5 exemplifies the generated code to the agent Giacomo. This code regards the diagram shown in Figure 4. The syntax of Jason recommends that the agent’s goals should be informed at first. Then, beliefs should be declared, and finally, the plans of the agent. The properties of each element was based on the case study presented on Uez [Uez 2013].
In Figure 5, line 1 is a message to the plug-in user, showing where he can inform the individual goals of each agents. After, the plans of the agent are described, as we mentioned previously. It is worth remembering that in the AgentSpeak language, the syntax of a plan is as follows:

\[
\text{Trigger : context } \leftarrow \text{ body}
\]

Therefore, lines 3, 8, 14, and 20 of Figure 5 describes the beginning of the plans. In line 4, is the declaration of the `buildHouse` plan, with the trigger `buildHouse`. The trigger watches an event to start the plan. As Uez [Uez 2013] describes, only two things can be a trigger to a plan: entities of the message type, or perception linked to a plan by dotted arrows. The `True` statement after `:` indicates the context of the plan, in this case stipulated by parameters by the designer. Lines 5, and 6, respectively, represent the inclusion of new objectives.

Also in Figure 5, lines 9, 10, 11, and 12 represent the `hire` plan. After the `\leftarrow` symbol is the concatenation of all the entities that make up the body of a plan. The lines 15, 16, 17, and 18 represent the `notifyWinner` plan. The word broadcast refers to a message sent to all the agents of the system. In addition, through the Figure 4, we can see that the links between entities messages and plans represent the messages that will be sent by the system. However, it is necessary that the message has a connection with the sender and the receiver. Lines 17, and 18 are queries of the agent’s beliefs. Finally, lines 21, 22, 23, and 24 represent the `auctionStart` plan. Line 22 describes the actions that should be executed by the plan.

In addition to Giacomo agent, this case study is composed of at least three more agents to represent companies that can participate in the bidding process. We only demonstrated the work product produced for Giacomo agent and the result of the code generation for the same. This process is enabled through the user clicking a button available on the Eclipse programming IDE for this purpose. Subsequently, the XMI produced by work diagrammed product is analyzed, thereby generating the entire skeleton codes required for reproduction of the agent in the Jason development platform. Furthermore, regarding the generated BDI architecture code, which is the basis which the platform in question. The plug-in developed still is in the testing phase, therefore, is not available for download.

**Figure 4. General Model Diagram for Giacomo Agent in the “Build a House” Case Study**
This paper proposes an Eclipse plug-in that allows the graphical design of MAS using the Prometheus AEOlus methodology. This methodology was developed by Uez [Uez 2013] and it aims to enable the integrated modelling of three dimensions involving MAS design. This integration intends to simplify the implementation of a MAS in the framework called JaCaMo. To aid this process, our plug-in is able to scan information about the MAS on the available diagrams and automatically generates code. This code is generated in AgentSpeak programming language, which is interpreted by the Jason platform, part of the JaCaMo framework. The dimensions environment (CartAgO) and organization (Moise+), which make up the rest of the JaCaMo framework, were not addressed in this work.

The separation of the Prometheus AEOlus methodology support platform in several plug-ins is one of the positive points in the preparation of this work, as it is the simplification of the tool maintenance process, and decouple the responsibilities in various minimized solutions. This separation provides the adhesion of collaborators, making it easier to spot maintenance in different parts of the source code developed by facilitating their control version.

The choice of a plug-in consolidated graphical modeling to make the graphical editor, entitled GMF Eclipse [Foundation 2014], was also an important factor, since it is cemented in the graphics editors market development, presenting a good documentation available and various forums able to answer questions by experienced users in using the plug-in. The GMF Eclipse separates the process developed in several steps, using the concept of Model Driven Engineering (MDE) which corresponds to an architecture related to models, consisting of an approach for the construction of systems by separating the specification of the functionality or logic business of implementation.

The development of an automatic plug-in to generate code is a factor to be improved in future work. Although the solution presents an architecture with well-divided responsibilities, it can later be replaced by an alternative that explore the MDE concepts present in Eclipse GMF, becoming a fully independent solution and easy support.

Finally, the platform is an alternative to model MAS, where work products related
to environmental and organizational dimensions can be specified, an innovation in the AOSE area. In addition, this solution demonstrates that Prometheus AEOlus [Uez 2013] compatibles to Jason development platform, but it is necessary to build new solutions that enable its connection with CartAgO and Moise+ platforms. After all connections are completed, the solution will be unprecedented in the MAS modeling area, as there is not a development platform to generate code for all dimensions proposed by [Demazeau 1995] to a MAS.

Acknowledgment

The authors thank the Universidade Federal do Rio Grande (FURG) and Fundacao de Amparo a Pesquisa do Rio Grande do Sul (FAPERGS) to support this research.

References


