ADCAIJ: Advances in Distributed Computing and Articial Intelligence Journal
Regular Issue, Vol. 6 N. 4 (2017), 41-54
eISSN: 2255-2863
DOI: http://dx.doi.org/10.14201/ADCAIJ2017644154

Development of a Graphical Tool to integrate the Prometheus AEOlus methodology and Jason Platform

Rafhael Cunha

Universidade Federal do Rio Grande. Brazil. rcrafhaelrc@gmail.com

KEYWORD

Agent Oriented Software Engineering; Multiagent System; Jason Platform; Prometheus AEOlus Methodology

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 (MAS). Therefore a new subarea has been studied, called Agent Oriented Software Engineering (AOSE). The AOSE area proposes solutions to specific issues related to the development of agent oriented systems. There are several methodologies to model MAS, however, until now, there is not a standard modelling language because they are very complex systems, and involve several different concepts. Another issue of this subarea is that there are very few tools that are able to automatically generate code, reducing its acceptance in the software development market. In this work, we propose a tool to support the Prometheus AEOlus Methodology, because it provides modelling artifacts to all MAS dimensions proposed by Demazeau: agents, environment, interactions and organization. The tool supports all Prometheus AEOlus artifacts and it can automatically generated code to the agent and interaction dimensions in the AgentSpeak(L) language, which is the language used in the Jason platform. We have done some validations with the proposed tool and a case study is presented. Our results indicate that our tool has full compatibility with the Jason platform, and it is able to automatic generate code in AgentSpeak(L). As future work, we intend to develop the integration of the artifacts with the JaCaMo framework, enabling a full integration between our tool and the Prometheus AEOlus methodology.

1. Introduction

In the last few years, the area of software engineering has proposed processes and modelling languages to support software development. The purpose of the area is to provide support to build robust, reliable, easy-to-maintain and reusable software in a quickly and organized way (Sommerville, 2006).

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, enhancing the Multiagent Systems (MAS) area (Wooldridge and Jennings, 1995).

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). These methodologies have been created for several reasons. However, anyone of them is able to address all dimensions of a MAS, as defined by (Demazeau, 1995).

Another reason to use AOSE is the possibility to automatically generate code based on MAS modelling. Currently, there are several tools to support MAS development, as Jade[1], Jason[2] or Jack[3]. However, any of these tools supports the development of work products to all different dimensions of a MAS, or even connections with other tools that support these dimensions.

The Prometheus AEOlus methodology (Uez, 2013) (Uez and Hubner, 2014) 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 modelling of organization and environment dimensions, unlike other ones. As a final goal, the Prometheus AEOlus methodology tries to be compatible to the JaCaMo framework (Boissier *et al.*, 2013).

The JaCaMo framework integrates three different development platforms, which are complementary to each other 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(L) 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 has diagrams and notations to model all 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.

In this way, 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. This task provided the confirmation of the adherence of the methodology with the Jason development platform. As future work we intend to develop the association with the other platforms that compose the JaCaMo framework (CartAgO and Moise+) .

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 2.4 we show the related methodologies, as well as a comparative study to demonstrate the reason of this work. In the section 3 we discuss the general architecture of the developed plug-in, as well as the steps taken to completion. In the section 4 is presented a case study that aims to demonstrate the use of the developed tool. Finally, in the section 5 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, interacting with it via sensors and actuators. It can be virtual or not. Additionally, this agent is able to make decisions and take actions autonomously in order to reach its 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.

1. http://jade.tilab.com/
2. http://jason.sourceforge.net/
3. http://aosgrp.com/products/jack/

According to (Wooldridge and Jennings, 1995), an agent is a computer system located in an environment, and is capable of autonomous actions on the environment in order to achieve its stated objectives.

(Briot *et al.*, 2001) define 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 the 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 (Bergenti *et al.*, 2004).

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 modelling 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 (Eclipse 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 (Eclipse Foundation, 2014), the Graphical modelling 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 graphical editors and Eclipse modelling 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.

2.4. Related Work

According to (Padgham and Winikoff, 2002), Prometheus is a methodology for the development of intelligent agents that differs from others because it is detailed and comprehensive, covering all necessary activities for the development of intelligent agent systems. The available tool for Prometheus is currently called Prometheus Design Tool (PDT) (Thangarajah *et al.*, 2005). PDT allows users to create and modify projects developed with Prometheus methodology. This tool ensures that there are no certain inconsistencies, such as using wrong diagrams notations. The tool allows to export 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. It generates code to the 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 multiagent systems based on the concepts of actor, object, and dependency among actors. According to (Cossentino *et al.*, 2005), the Tropos methodology allows direct mapping of modelling for JACK language. However, the TAOM4E tool can also automatically generate code for the 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-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[4].

According to (Henderson-Sellers and Giorgini, 2005), the Ingenias methodology provides a notation for MAS modelling through a well-defined collection of activities to guide the development process. Specifically, the following tasks: 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) (Uez and Hubner, 2014), 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, 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 modelling agents. To support the model code, there is the PTK tool (PASSI Toolkit). The tool 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.

The GAIA methodology (Wooldridge *et al.*, 2000) is designed to address aspects of system's analysis and design based on agents. This methodology deals with macro-level (company) and micro-level (agents) aspects, and it appears to be neutral in relation to the agent architecture domain. According to (Uez, 2013) (Uez and Hubner, 2014), GAIA has been developed to allow the MAS to be analyzed and designed with sufficient detail to enable its implementation. According to (Zambonelli *et al.*, 2003), the GAIA methodology produces textual artefacts, and it does not have a particular graphical notation, although it is recommended the use of AUML.

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 to create an extension of Prometheus methodology to include Environment and Organization dimensions (Uez, 2013) (Uez and Hubner, 2014).

The development process defined in this methodology is divided in four phases: system specification, architectural design, detailed design, and implementation (Uez, 2013) (Uez and Hubner, 2014). 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) (Uez and Hubner, 2014). 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) (Uez and Hubner, 2014) was based on the MAS division proposed by (Demazeau, 1995), that divides MAS in four dimensions: agents, interactions, environment, and organization.

According to (Uez, 2013) (Uez and Hubner, 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.

4. http://agenttool.cis.ksu.edu/

2.4.1. *Analysis of related work*

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

Methodology	Support Tool	Development Platform	MAS Dimension
Prometheus	PDTools	Jack	Agents and Interaction
Tropos	TAOM4E	Jadex e Jack	Agents and Interaction
MaSE	AgentTool	-	-
Ingenias	IDK	Jade	Agents and Interaction
MESSAGE	-	-	-
PASSI	PTK	Jade	Agents and Interaction
GAIA	-	-	-
Prometheus AEOlus	-	JaCaMo	Agents, Environment, Interaction and Oraganization

In Table 1, Methodology and Support Tool columns show the methodologies studied in this work and the graphical tools that support their use, respectively. In this table, there are many graphical tools for modelling MAS and although each technique has its own peculiarities, it is known that they are able to model MAS beyond the agents dimension. Therefore, from this comparison, it follows that the main problem regarding the specification/implementation of a MAS consists in processing the work products to code developed automatically. Regarding to the code generation, it aims to transform the specifications set out in the work products in to source code. Table 1 also shows the development platforms that allow to automatically generate code, besides the dimensions of the MAS that this generation supports, through the Development Platform and MAS Dimensions columns, respectively. Thus, it is shown through the table, the preponderance for code generation for Jade and Jack frameworks, with the exception of Tropos that also generates code for the 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.

Based on Table 1, only one methodology has in its core the concern with agent, environment, interaction and organization dimensions, and that is the Prometheus AEOlus, through the framework JaCaMo (Boissier *et al.*, 2013). For this reason, we decided to develop a graphical tool to the Prometheus AEOlus methodology.

3. A Tool to Support Prometheus AEOlus Methodology

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

3.1. General Architecture

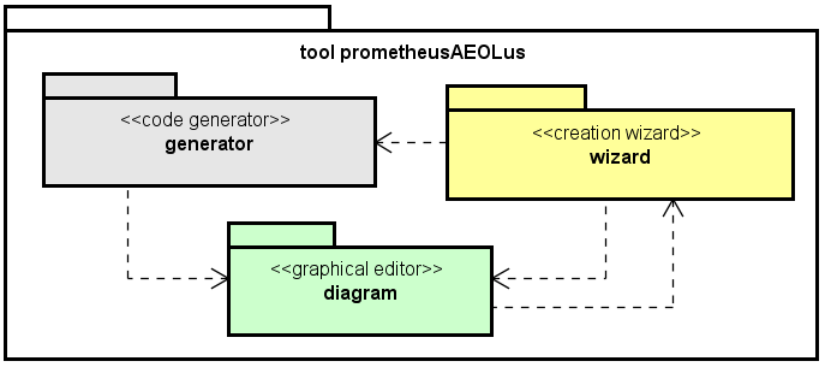

Figure 1: General Architecture of Prometheus AEOlus Tool

The general architecture of the Prometheus AEOlus tools is presented in Figure 1. Each package is a different plug-in to incorporate functions to the tool. The package called *generator* represents the plug-in responsible for code generation. This package has a dependency relation with the package *diagram* because it needs information from the diagrams.

The *diagram* package is the plug-in to support the graphical editor. This package contains all configuration and code files generated by the GMF eclipse plug-in, which was used to develop the graphical part of our tool. Unlike others plug-ins that are part of the tool, the *diagram* is not a Plug-in Development Environment (PDE), so it does not have the capacity to start independently. So the *diagram* package has a dependency to the *wizard* package. All meta-models and details of diagrams were obtained in the work of (Uez, 2013) (Uez and Hubner, 2014). Therefore, we do not show them. Our focus is to explain the modelling and the code generation.

The *wizard* package has the plug-in responsible for connect the other two plug-ins, so it has dependency to them. This plug-in is a PDE, so it can start the other ones, because it has *runtime* menu on Eclipse IDE. An interesting characteristic of this plug-in is that it does not have source code, just extension points in a XML configuration file to determine the dependencies.

3.2. Diagram Development on the Graphical Editor Plug-in

The Prometheus AEOlus methodology specifies at least twelve diagrams to the development process. However, to a concept proof, in this paper, we explain just the development of the overview diagram agents. According to (Uez, 2013) (Uez and Hubner, 2014), the agent overview diagram describes the agent internally, i.e., this diagram details plans, skills, and beliefs of an agent, as well as its 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, 2013) (Uez and Hubner, 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 necessary steps to design the graphical editor referring to the overview diagram agents. This editor is designed using the GMF Eclipse described in Section 2.3. The development process shown in Figure 2 follows the specified flow by the Eclipse Foundation (Eclipse Foundation, 2014). In the first step, the meta-model of a diagram is developed, i.e., classes and attributes are defined to represent the digram's information, as Action Message Plan class. The second step is to generate the skeleton code related to the

metamodel previously developed. The third step is to create a metamodel derivative file to the graphical editor, and all the figures used in the editor are specified by geometric points, they represent the classes specified in the metamodel designed earlier. These steps are from the part of the EMF Eclipse plug-in.

The fourth step of the development process is the graphic palette specification that will be used to draw the component. The palette is described based on the metamodel, and images for each metamodel classes are developed externally. The fifth step is to combine 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 the graphical editor referring to the diagram developed. These steps are related to GEF Eclipse plug-in.

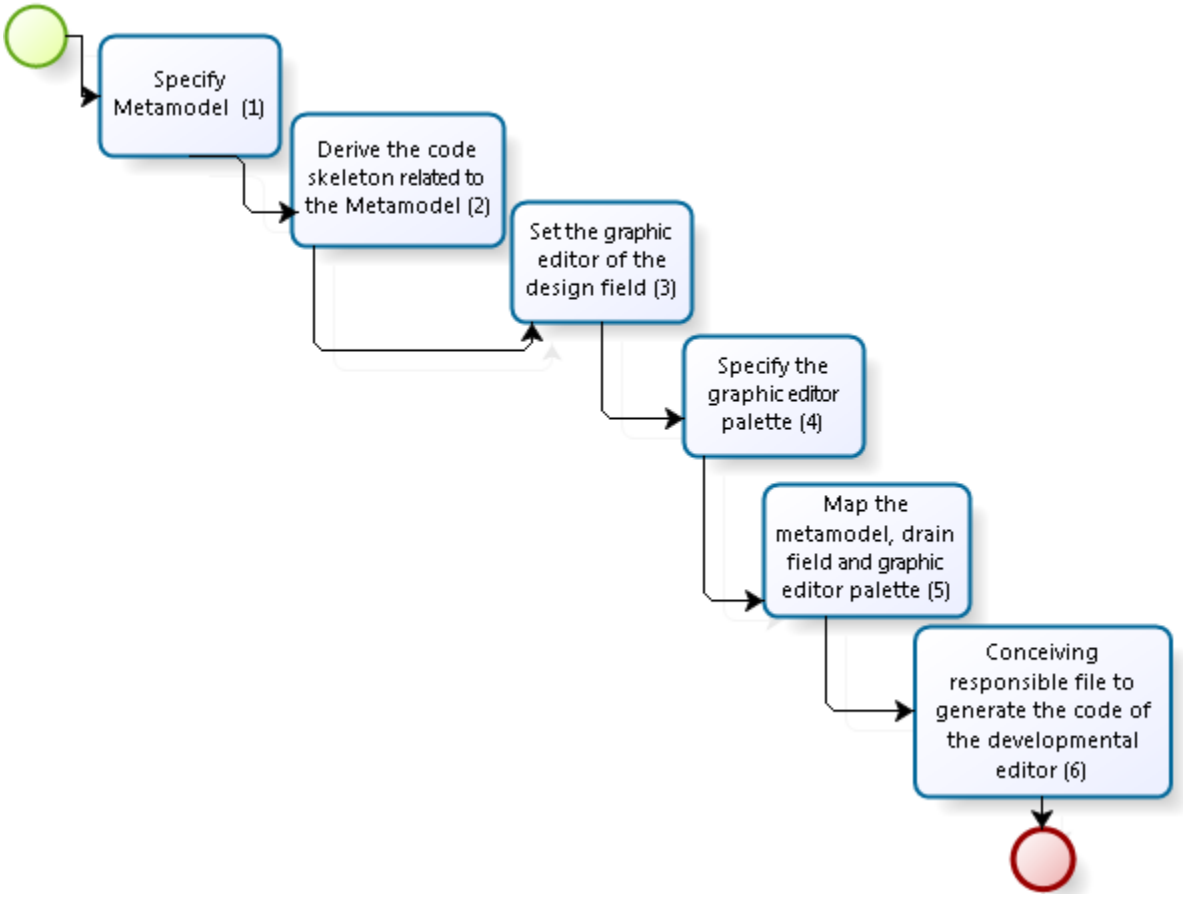

Figure 2: Process development graphic editor

3.3. Data Persistence in the GMF Eclipse Plug-in

The Eclipse GMF uses a XMI file to make data persistence. According to (Rouse, 2005), XMI is variation of XML to provide a standard way to exchange information about metadata (essentially, information about a data set and its organization). An example of an XMI file generated by the GMF is showed in Figure 3.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <PrometheusAEOLus_generalModel:Domain xmi:version="2.0"
3  xmlns:xmi="http://www.omg.org/XMI"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xmlns:PrometheusAEOLus_generalModel="PrometheusAEOLus_generalModel">
6    <nodes xsi:type="PrometheusAEOLus_generalModel:Goal"/>
7    <nodes xsi:type="PrometheusAEOLus_generalModel:Role"/>
8    <nodes xsi:type="PrometheusAEOLus_generalModel:Action"/>
9    <nodes xsi:type="PrometheusAEOLus_generalModel:Artifact"/>
10   <nodes xsi:type="PrometheusAEOLus_generalModel:Perception"/>
11   <nodes xsi:type="PrometheusAEOLus_generalModel:Protocol"/>
12   <nodes xsi:type="PrometheusAEOLus_generalModel:Message"/>
13   <nodes xsi:type="PrometheusAEOLus_generalModel:Belief"/>
14   <nodes xsi:type="PrometheusAEOLus_generalModel:Plain"/>
15   <nodes xsi:type="PrometheusAEOLus_generalModel:Scenario"/>
16   <nodes xsi:type="PrometheusAEOLus_generalModel:Capability"/>
17   <nodes xsi:type="PrometheusAEOLus_generalModel:Agent"/>
18   <missions/>
19   <groups/>
20   <workspaces/>
21   <edgesSimple source="//@nodes.11" target="//@nodes.1"/>
22 </PrometheusAEOLus_generalModel:Domain>
```

Figure 3: Example of the General Diagram Model

The Figure 3 shows a XMI file with different tags that makes the information extraction process easier. For example, line 6 to line 17 describe the entities that were modeled in the canvas (graphical diagram). Line 21 describes a relationship between two entities, Role - node 1 and Agent - node 11. For these tags, it is also possible to identify the type of the relationship. In this case it is a edgesSimple between Role - node 1 and Agent - node 11.

In the process of code generation to the Jason Platform, we used the XMI files. These files have information about the modeled General Diagram. This diagram contains all entities and relationships. We read the XMI files to extract information about the MAS, and after, we are able to automatically generate code.

3.4. Code Generator Plug-in

The Figure 4 shows the architecture of the Code Generator plug-in. This plug-in works with the graphical editor plug-in. Both were developed as different projects and they are put together in the *wizard* plug-in.

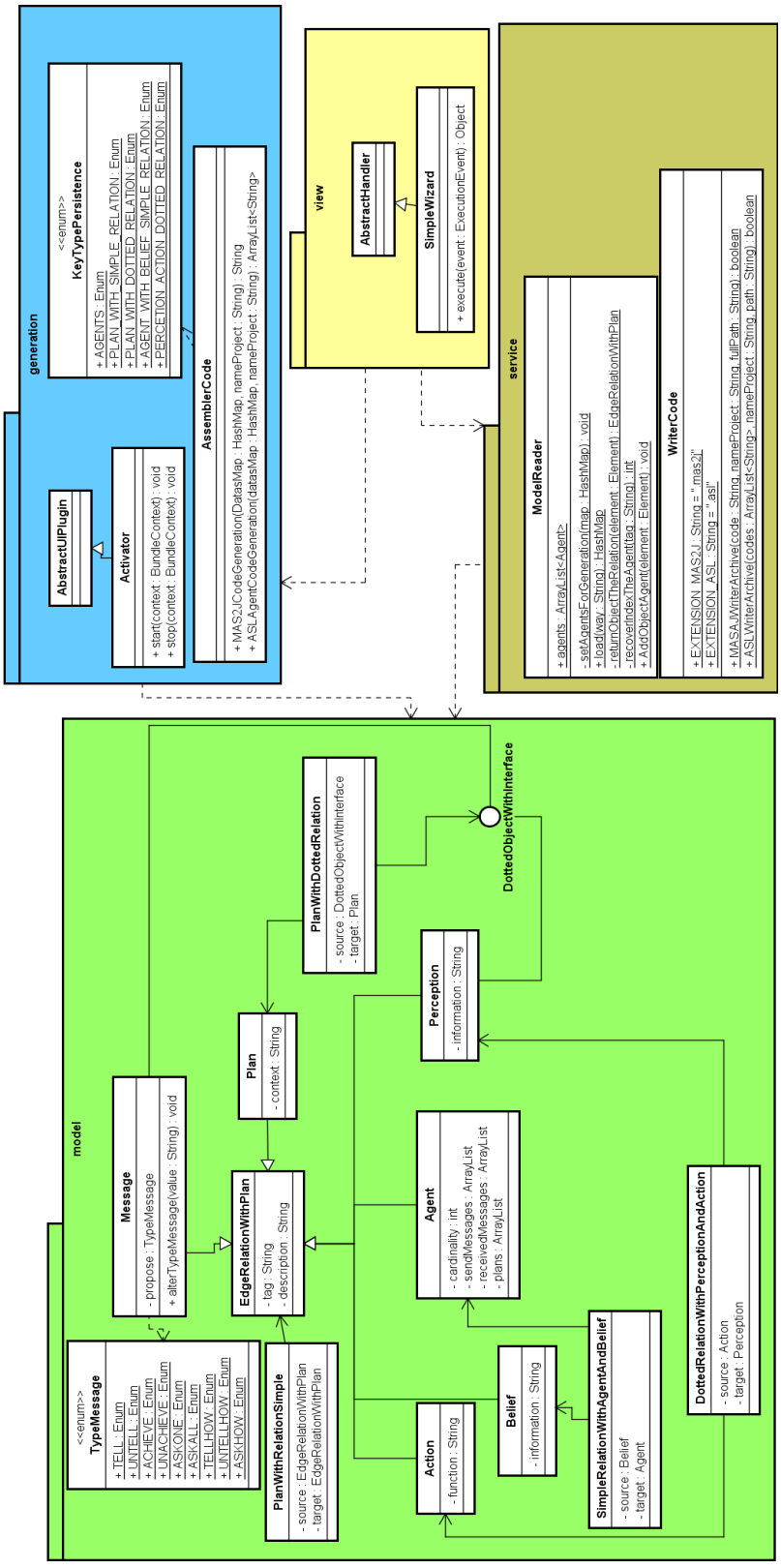

Figure 4: Architecture of Code Generator plug-in

The code generator plug-in has 4 layers: model, generator, view and service. The first layer is responsible for represent the entities through objects. Moreover, this layer also has some classes to help the plug- in to find information in the XMI file. For example, the *PlanWithRelationSimple*, *EdgeRelationWithPlan*, *PlanWithDottedRelation*, *DottedRelatonWithPerceptionAndAction* and *SimpleRelationWithAgentAndBelief* classes do not represent entities. They describe possible relations used on the code generation process.

The generator layer has the plug-in tasks within the Eclipse environment. These classes are called *AbstractUIPlugin*

and *Activator*. In this layer, it is also included the *AssemblerCode* class, whose function is to write the *string* that will contain the codes that must be persisted in the file that the *plug-in* will generate later, therefore it is an Application classes.

The service layer is responsible for reading the XMI file explained in Section 3.3, which is generated by the use of the graphic modelling plug-in. This plug-in does not read all the XMI files generated by the graphical editor plug-in, being interpreted only the file that corresponds to the information modelled in the Prometheus AEOlus General Model diagram. The class that performs this reading is called *ModelReader*, and it is also responsible for transforming all the XMI tags file into objects, returning a data structure of type *hashmap*, which contains the transformed objects.

The view layer manages service utilization and code generation developed by the plug-in. From methods of this layer, it is possible to activate the plug-in functionality. Activation occurs through events that are triggered by users.

4. Case Study - Build a House

This section aims to demonstrate the use of the developed tool. The validation tool occurred on two different ways: firstly, we aimed to perform unit tests to evaluate the efficiency of the code generator plug-in, validating its architecture and evaluating its efficiency (computational level). However, this first validation is not addressed in this paper. We focused our validation in this paper on usage of the plug-in graphical editor, in order to demonstrate its effectiveness in enabling users to transcribe integrally the diagrams described in (Uez, 2013) (Uez and Hubner, 2014).

In (Boissier *et al.*, 2013), the case study called «Build a House» was presented. In this case study, a character called *Giacomo* wants to build its 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.

To demonstrate the use of the graphical editor tool, we chose to model computationally Giacomo Agent, which is the main character of the case study in question. Figure 5 shows the Overview Diagram of the Giacomo Agent. This is one of the diagrams of the scenario. In Figure, the Giacomo entity is modeled as an agent. The entities buildHouse, notifyWinner, hire and, auctionStart represent plans that the agent has. The entities buildHouse, build, hire, auctionWinner, notificesWinners 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 plans.

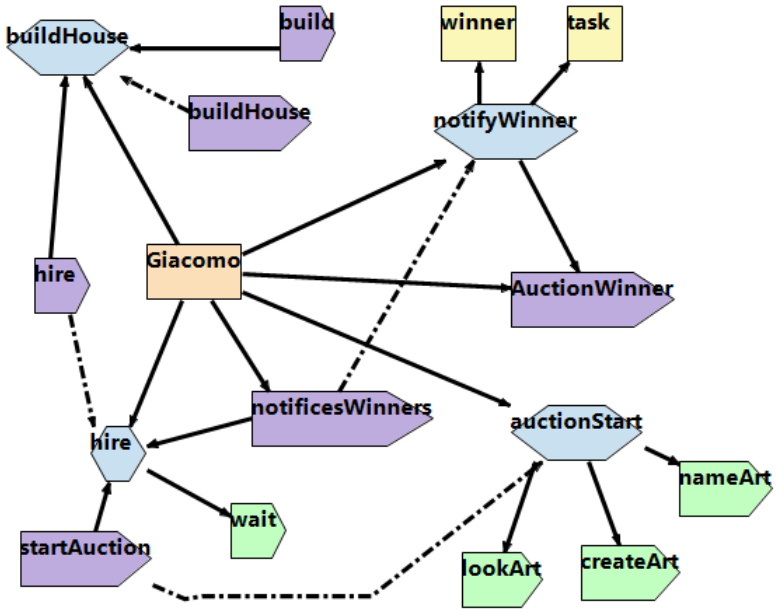

Figure 5: Overview Diagram of the Giacomo Agent in the «Build a House» Case Study

The Figure 6 exemplifies the generated code to the agent Giacomo. This source code represents the diagram shown in Figure 5. The syntax of Jason recommends that the agent's goals should be informed at first. After, beliefs should be declared, and finally, the plans of the agents. The properties of each element were based on the case study presented on Uez (Uez, 2013) (Uez and Hubner, 2014).

In Figure 6, line 1 shows a message to the plug-in user, showing where he/she can inform the individual goals of each agent. After, the plans of the agent are described, as we mentioned previously. The sintax of the AgentSpeak language is:

Trigger : context <- body

Therefore, lines 3, 8, 14 and 20 of Figure 6 describe the plans beginning. In line 4, buildHouse plan, with the trigger buildHouse. A trigger watches an event to start the plan. As Uez (Uez, 2013) (Uez and Hubner, 2014) describes, only two things could be a trigger into 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 developer. Lines 5 and 6, respectively, represent the inclusion of new goals.

Also in Figure 6, lines 9, 10, 11 and 12 represent the hire plan. After the < - 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.

```
1  //insert here the goals to boot your agent
2
3  @buildHouse
4  +!buildHouse :true
5  <-!hire;
6  !build.
7
8  @hire
9  +!hire :true
10 <- !startAuction(x);
11 !notificesWinners;
12 .wait(500).
13
14 @notifyWinner
15 +!notificesWinners :true
16 <-.broadcast(tell, winner(Task, Winner));
17 ?task;
18 ?winner.
19
20 @auctionStart
21 +!startAuction(x) :true
22 <-.concat("auction_",Task,ArtifactName);
23 focus(id);
24 makeArtifact(ArtefactName, "auction", [Task,MaxValue],Id).
```

Figure 6: Generated code for Giacomo Agent in the «Build a House» Case Study

The word broadcast refers to a message that will be sent to all the agents of the system. In addition, through the Figure 5, 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 this 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. After, the XMI produced by work diagrammed product is analysed, and the entire skeleton codes required for reproduction of the agent in the Jason development platform is generated. Furthermore, the generation code regards the BDI architecture principles.

5. Conclusions

This paper proposes an Eclipse plug-in that allows the graphical design to the Prometheus AEOlus methodology, in order to develop MAS applications. This methodology was developed by Uez (Uez, 2013) (Uez and Hubner, 2014) and it aims to enable the integrated modelling of three dimensions involving MAS design. This integration tries to simplify the MAS implementation to the JaCaMo framework. To aid this process, our plug-in is able to take information about the MAS that is available in graphical diagrams and automatically generates code. This code is generated in AgentSpeak(L) language, which is interpreted by the Jason platform, part of the JaCaMo framework. The environment (CartAgO) and organization (Moise+) dimensions, which integrates the JaCaMo framework, were not addressed in this work.

The separation of the Prometheus AEOlus methodology in several plug-ins is one of the positive points in our work, as it is the simplification of the tool maintenance process, and decouple the responsibilities in several small solutions. This separation provides the adhesion of collaborators, making it easier to maintain different parts of the source code developed, and consequently, facilitating the version control.

The choice of a consolidated graphical modelling plug-in to make the graphical editor, entitled GMF Eclipse (Eclipse Foundation, 2014), was also an important aspect, since it is one of the graphic development editors most used in market, that presents a good documentation and several forums with answers by expert users. 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 where the specification of the functionality and logic business of implementation are defined in separated ways. The development of an automatic plug-in to generate code is an aspect 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) (Uez and Hubner, 2014) is compatible 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 tool will be a complete solution in the MAS modelling area, because there is not a development platform to generate code for all dimensions proposed by (Demazeau, 1995). In this way, this tool is an important contribution to the MAS modeling area.

6. 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.

7. References

Bergenti, F., Gleizes, M.-P., and Zambonelli, F., 2004. *Methodologies and software engineering for agent systems: the agent-oriented software engineering handbook*, volume 11. Springer. ISBN 978-1-4020-8057-9.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A., 2013. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6): 747-761.

Bratman, M., 1987. Intention, plans, and practical reason. Harvard University Press, Cambridge, MA.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J., 2004. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3): 203-236.

Briot, J.-P., Demazeau, Y. *et al.*, 2001. *Principes et architecture des systèmes multi-agents*, volume 217. Hermès Science Publications.

Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavón, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R. *et al.*, 2002. Agent oriented analysis using MESSAGE/UML. In *Agent-oriented software engineering II*, pages 119-135. Springer.

Cossentino, M. and Potts, C., 2002. PASSI: A process for specifying and implementing multi-agent systems using UML. *Retrieved October*, 8:2007. Http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.3182. Cossentino, M., Seidita, V., Miciletto, N., and Rubino, R., 2005. Tropos: Processo e frammenti. *Rapporto Tecnico N.: RT-ICAR-PA-05-06, Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni*.

DeLoach, S. A., 1999. Multiagent systems engineering: a methodology and language for designing agent systems. Technical report, DTIC Document.

DeLoach, S. A., Wood, M. F., and Sparkman, C. H., 2001. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(03): 231-258.

Demazeau, Y., 1995. From interactions to collective behaviour in agent-based systems. In *In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*. Http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.7968&rep=rep1&type=pdf. Dennett, D. C., 1989. *The intentional stance*. MIT press.

desRivieres, J. and Wiegand, J., 2004. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, 43(2): 371-383.

Eclipse Foundation, 2012. Eclipse documentation - Current Release. Http://help.eclipse.org/luna/index.jsp. Eclipse Foundation, 2014. Graphical Modeling Framework. Http:wiki.eclipse.org/Graphical_Modeling_Framework. Gleizes, M.-P. and Gomez-Sanz, J. J., 2011. *Agent-Oriented Software Engineering X: 10th International Workshop, AOSE 2009, Budapest, Hungary, May 11-12, 2009, Revised Selected Papers*, volume 6038. Springer.

Henderson-Sellers, B. and Giorgini, P., 2005. *Agent-oriented methodologies*. IGI Global.

Hübner, J. F., Sichman, J. S., and Boissier, O., 2002. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *Advances in artificial intelligence*, pages 118-128. Springer.

Morandini, M., Penserini, L., and Perini, A., 2008. Automated mapping from goal models to self-adaptive systems. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 485-486. IEEE Computer Society.

Mylopoulos, J., Kolp, M., and Castro, J., 2001. UML for agent-oriented software development: The Tropos proposal. In *UML 2001-The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, pages 422-441. Springer.

Padgham, L. and Winikoff, M., 2002. Prometheus: A Methodology for Developing Intelligent Agents. *John Wiley & Sons*.

Padgham, L. and Winikoff, M., 2005. *Developing intelligent agent systems: A practical guide*, volume 13. John Wiley & Sons.

Ricci, A., Piunti, M., and Viroli, M., 2011. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2): 158-192.

Rouse, M., 2005. XMI (XML Metadata Interchange) definition. Http://searchsoa.techtarget.com/definition/XMI. Russell, S. and Norvig, P., 2009. *Artificial Intelligence: A Modern Approach*, volume 3. Prentice Hall Press.

Sommerville, I., 2006. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison- Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0321313798.

Thangarajah, J., Padgham, L., and Winikoff, M., 2005. Prometheus design tool. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 127-128. ACM.

Uez, D. M., 2013. *Método para o desenvolvimento de software orientado a agentes considerando o ambiente e a organização*. Master's thesis, Universidade Federal de Santa Catarina.

Uez, D. M. and Hubner, J. F., 2014. *Engineering Multi-Agent Systems*, chapter Environments and Organizations in Multi-Agent Systems: From Modelling to Code. Springer.

Uez, D. M. and Hübner, J. F., 2014. Environments and organizations in multi-agent systems: From modelling to code. In *Engineering Multi-Agent Systems*, pages 181-203. Springer.

Weiss, G., 1999. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

Wooldridge, M. and Jennings, N. R., 1995. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02): 115-152.

Wooldridge, M., Jennings, N. R., and Kinny, D., 2000. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, 3(3): 285-312.

Zambonelli, F., Jennings, N. R., and Wooldridge, M., 2003. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3): 317-370.