FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

From simulation to development in MAS: Repast-JADE automatic code generation for interaction protocols

João Pedro Camacho Lopes

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Henrique Lopes Cardoso

February 11, 2014

From simulation to development in MAS: Repast-JADE automatic code generation for interaction protocols

João Pedro Camacho Lopes

Mestrado Integrado em Engenharia Informática e Computação

February 11, 2014

Abstract

Multi-agent systems (MAS) present an interesting approach to the efficient development of modular systems. MAS are composed by autonomous computational units called agents that are programmed to *compete* or *work together*, for instance in order to solve computational problems, engage in automatic negotiation or play computer games. Frameworks exist that aid the development of this class of systems and they range from mostly general-purpose frameworks to domain-specific in an array of different domains.

Multi-agent-based simulations (MABS) are sometimes used on the course of development of a full-featured MAS - for instance, for testing purposes. However, most platforms for MAS development are not well suited for MABS development [MTLD08].

JADE [BCPR03], a very popular MAS development framework allows the creation of seamless distributed agent systems and complies with FIPA standards for agent interaction. Unfortunately, its multi-threaded architecture falls short in delivering the necessary performance to run a local simulation with a large number of agents. Repast [Col03] is an agent-based simulation toolkit that allows creating simulations using rich GUI elements and real time agent statistics. Unlike JADE, though, Repast lacks much of the infrastructure for agent creation and interaction.

Some authors [GRM+11, GHM+11], as reviewed in this report, proposed an integration of JADE and Repast by means of a middleware, essentially representing the agent in both and taking advantage of the features provided by the frameworks.

In this thesis, a code generation tool is proposed, capable of not only generating a Repast simulation from an existing JADE MAS, but also of creating a full featured JADE application from a Repast-based simulation. An implementation of FIPA's interaction protocols will be proposed for Repast as a means to deliver the mentioned conversion tool. The most immediate advantage over the previous approaches is that existing systems can make use of this tool to create appropriate simulations where agent interaction can be more easily analyzed. Furthermore, proficient programmers in one framework can quickly get started in the development for the complementary framework.

Resumo

Os sistemas multi-agente (SMA) exprimem uma abordagem interessante no desenvolvimento de sistemas modulares e eficientes. Os MAS são compostos por elementos computacionais autónomos - chamados agentes - que são programados para *competir* ou *colaborar* de modo a, por exemplo, resolver problemas computacionais, iniciar negociação automática ou participar em jogos de computador. Existem ferramentas de software que facilitam o desenvolvimento desta classe de sistemas que podem variar entre ferramentas de âmbito geral até ferramentas focadas num domínio específico.

As simulações baseadas em agente (SBA) são frequentemente utilizadas durante o desenvolvimento de MAS completos - por exemplo, afim de testar o sistema. No entanto, a maior parte das plataformas para desenvolvimento de SMA não são apropriadas para a criação de SBA [MTLD08].

O JADE [BCPR03] é um exemplo de uma plataforma de desenvolvimento de SMA que permite a criação de sistemas distribuídos de agentes de forma simplificada cumprindo, ainda, os standards da FIPA (*Foundation for Intelligent Physical Agents*) sobre protocolos de interação entre agentes. No entanto, a sua arquitetura em *multi-thread* não garante a performance necessária para a execução de simulações com um elevado número de agentes. O Repast [Col03] é uma plataforma de criação de simulações baseadas em agentes que permite criar simulações ricas em interfaces gráficas para visualização de dados históricos e em tempo real sobre os agentes. Ao contrário do JADE, o Repast não dispõe de uma infraestrutura para criação de agentes e interação entre eles.

Alguns autores estudados neste relatório [GRM+11, GHM+11] propõem uma integração entre o JADE e o Repast por meio de uma camada de software intermédio, fazendo representar cada agente duplamente - uma vez em cada plataforma, essencialmente tirando partido das capacidades de ambas as plataformas em simultâneo.

Para esta dissertação é proposta uma ferramenta de geração automática de código capaz não só de gerar uma simulação baseada em Repast a partir de um SMA baseado em JADE, mas também de criar um SMA em JADE partindo de uma simulação baseada em Repast. Será também criada uma proposta de implementação, para a plataforma Repast, dos standards da de interação entre agentes da FIPA, tornando possível a conversão de código entre as duas plataformas. A vantagem mais imediata sobre a proposta anterior é que SMA já existentes podem tirar partido da ferramenta proposta para rapidamente criar uma simulação com funcionamento equivalente e ter acesso a ferramentas de visualização e estatísticas do sobre o comportamento dos agentes. Adicionalmente, programadores proficientes numa das plataformas pode iniciar rapidamente o desenvolvimento da plataforma complementar.

Contents

CONTENTS

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Abbreviations

ACL	Agent Communication Language
AST	Abstract Syntax Tree
ATL	ATLAS Transformation Language
CeCILL-C	CEA CNRS INRIA Logiciel Libre (open source license)
EPL	Eclipse Public License (open source license)
FIPA	Foundation for Intelligent Physical Agents
FOSS	Free and Open Source Software
IDE	Integrated Development Environment
JADE	Java Agent DEvelopment Framework
JDT	[Eclipse] Java Development Tools
MABS	Multi-Agent-Based Simulation
MAS	Multi-Agent System

Chapter 1

Introduction

Multi-agent systems (MAS) are composed of autonomous computational elements capable of interacting with each other, called agents. The development of this class of systems comprises an interesting software paradigm but in terms of computer science history, MAS are a recent subject, having gained significant traction only after the mid 1990's [Woo08]. With multiple applications such as problem solving, simulation, trading, negotiation, computer games and logistics using an efficient and modular development approach, MAS enjoyed a rapid growth in popularity and are in widespread use nowadays [Fer99].

1.1 Context

Although their use is certainly widespread, there is no single general purpose standard for MAS development, since each system has different needs. Many times, such systems are created from scratch, meaning that the developers must define all features of the system - such as its agents, their behavior, communication and organization, using conventional programming languages and tools. However, several frameworks exist that offer some level of abstraction from the code, allowing for a more conceptual approach to MAS development [GHM+11].

Most uses of MAS, for instance in negotiation, games or logistics, demand a small number of agents, typically with larger resource demands but without any need for global control of execution, i.e. it is perfectly reasonable for these types of systems to be based on events and for its agents to work asynchronously. In contrast, multi-agent-based simulations (MABS) are usually implemented using a large number of lightweight agents with a small resource footprint. MAS development frameworks generally provide the programmer with a range of features such as execution control, communication protocols or agent awareness capabilities. In spite of that, most frameworks that focus on MAS development lack synchronization mechanisms and lightweight

agent infrastructure required by MABS. One of the main goals of simulations is to be able to visualize real-time, as well as historical data that allow to study emergent and evolutionary phenomena. [MTLD08]

1.2 Motivation

The motivation for this thesis pertains the study of the development of agent-based software, focusing on the notion that MAS and MABS can both integrate the same project. As said before, MAS and MABS developers expect different groups of features to be provided by the development frameworks. Furthermore, in the course of development, a need may arise to use a different set of features. For instance, while developing a large scale MAS, a need for creating a simulation based on the same model may arise. The reverse can occur as well - the development may start as a simulation that later must be implemented with another kind of framework.

This thesis will focus on using two frameworks that are very popular in their respective domain. First we have JADE, a framework that attempts to simplify the development of distributed agent applications by seamlessly hiding all complexity concerning its distributed architecture. JADE is also FIPA compliant, meaning that is much easier to create an open MAS. With JADE, the programmers "can focus their software development just on the logic of the application rather than on middleware issues, such as discovering and contacting the entities of the system" [BCPR03]. The second framework is Repast, a toolkit that provides a group of libraries for the creation and execution of MABS, while allowing the programmer to collect data from the agents. With Repast it is also fairly simple to create a wide range of different charts to represent the collected data as well as real-time agent performance.

Table 1.1: Summary of JADE and Repast features.

	JADE	**Repast**
Communication	FIPA ACL	Method calls Shared resources
Distribution	Yes	No
Simulation Tools	No	Yes
Scalability	Limited	High
Ontologies	Yes	No
Open Source	Yes	Yes
Agent Execution	Behavior-based Multi-threaded Event-driven Assync	Schedule-based Single-threaded Tick-driven Sync

While JADE excels in creating seamless, distributed, peer-to-peer systems, Repast's strength is in creating simulations as well as collecting and displaying statistical data in different ways. In JADE, as table 1.1 illustrates, agents execute in separate threads and while this architecture

facilitates the platform's distribution, JADE's agent are heavy in terms of resources. Experiments with JADE show that the platform's scalability is limited in number of agents and that the global system performance drops quickly for large number of agents [MTLD08] [GRM+11].

JADE and Repast were chosen over other platforms mainly for their popularity and widespread use - not dismissing their quality, of course. As an example of an alternative framework, Cougaar (Cognitive Agent Architecture) solves the problem explain above by proposing a fully featured agent architecture, while maintaining high performance and scalability required for simulation. It doesn't, however, implement any interaction standards; messages are exchanged by means of serialized Java objects [HTW04]. Table 1.1 presents a rundown of the comparison of these frameworks.

The analysis made in this report was based on to JADE 4.3.1 (6 December 2013) and Repast Symphony 2.1 (12 August 2013), which are also the versions that will be used throughout the production of this thesis.

1.3 Goals

The first goal of this thesis is to create a code conversion tool. This tool enables the programmer to generate a Repast simulation automatically from the source code of a JADE application (development to simulation), as well as obtaining a proper multi agent system in JADE starting from a Repast simulation (simulation to development). Such a conversion tool not only allows the developer to quickly deploy a MAS or create a simulation, as explained before, but also enables a proficient programmer in one framework to quickly get started in developing with the other framework.

It's important to note that the main focus of this thesis is to convert agent interaction between the two frameworks. Agent behavior can be far more complex and go beyond simply agent interaction and, while the initial work will be laid down to allow for future work on perfectly mimicking agent behavior, this is not a main goal.

The second goal is to bring communication standards to Repast. While JADE complies with the FIPA standards for agent interaction, Repast implements no standards at all. Creating a library for implementation of FIPA interaction protocols in Repast will not only enrich that framework but will make the code generation more straightforward.

One of the soft goals for the final product is that the programmer should not be forced to introduce significant changes to the original code in order to be able to use the tool. The generated code must also preserve the functionality of the original code - meaning that the re-conversion must generate code that is equivalent to the original one.

1.4 Report contents

To summarize, this report proposes the development of a tool that converts a MABS created in Repast into MAS that uses JADE. Conversely, it should allow the conversion of a JADE MAS

into a Repast MABS as well. This tool is useful in the context of development of a MAS whose development started as a MABS or when the need to create a simulation arises during development. JADE and Repast were chosen for this thesis not only for their quality but also for their widespread use.

After this introduction, chapter 2 starts by discussing some related work about JADE, Repast and about integrating both frameworks. While the motivation for this thesis pertains the improvement of MAS development methodologies, its goal is to convert the code from one framework to the other. Therefore, a large part of this thesis will be about Java code transformations and part of the state-of-the-art review is related to code transformation tools.

Chapter **??** describes the details of the development environment as well as the definition of how the final product will be validated and evaluated. Chapter 3 includes a detailed work plan for the following months and finally, chapter 4 includes final notes about this report and about the work done so far.

Chapter 2

Bibilography Review

This chapter presents a summary and analysis of some works related to the use of Repast and JADE frameworks in the development of MAS. The related work is sorted in four categories.

The first category includes a discussion about existing works on integrating both JADE and Repast simultaneously by developing a middleware framework. While the goal of this thesis is not to use both frameworks simultaneously, these works give a valuable insight into the shortcomings of both frameworks as well as providing some interesting comparisons between their features. The second category includes discussion about implementations of these protocols in Repast. Then, the third category presents some existing Java code generation and transformation tools. These tools will give support to the automation of code refactoring, detection of code patterns and library calls. This chapter ends with a summary of the bibliography review and an overview of the most interesting topics approached in this chapter.

2.1 JADE and Repast integration

JADE is a very popular framework for developing MAS, providing an infrastructure that allows the programmer to create a distributed system in a very transparent way. As discussed before, in the motivation for this thesis, JADE is not the best tool to develop a MABS, not only because it lacks the visualization tools that other frameworks provide for displaying data about agent behavior, but also because due to its multi threaded architecture JADE has performance issues when the system comprises a large number of agents. Following are two frameworks created to bridge these two worlds of MAS development and simulation by integrating Repast and JADE in the same platform by means of a middleware.

2.1.1 MISIA: Middleware Infrastructure to Simulate Intelligent Agents

MISIA is a middleware whose goal is to enhance the simulation of intelligent agents and to allow the visualization and analysis of agent's behavior. Not only does MISIA allow "simulation,

visualization and analysis of the agent's behavior", but also "makes use of technologies for the development of multi-agent systems known and widely used, and combines them so that it is possible to use their capabilities to build highly complex and dynamic systems." In Garcia *et al*, the authors chose to develop this framework on top of JADE and Repast Symphony. Their reasoning lies on the widespread use of JADE as an agent-based software development middleware, on Repast being simple and well documented and finally on both frameworks being open source. [GRM+11]

Most platforms like JADE lack organizational features and support for simulation constraints - this infrastructure must be developed by the programmer. The main concept introduced to JADE by MISIA is the concept of time - synchronization is fundamental for simulation. MISIA also brings the possibility to develop open MAS in Repast by taking advantage of JADE implementation of FIPA protocols.

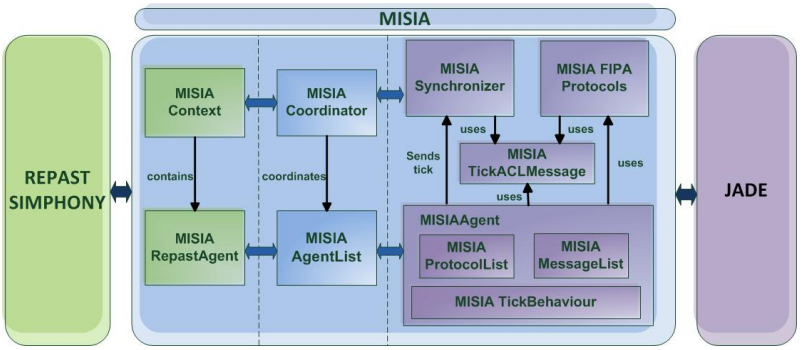


Figure 2.1: Functional structure of MISIA [GRM+11]

As illustrated in the diagram on figure 2.1, MISIA consists of three layers of components: a layer of contact with Repast; an intermediate layer between the two frameworks; a layer that interfaces with JADE and enables the concept of time in that framework.

The purpose of the 'JADE layer' is for JADE's agents to perform their actions in each tick - the smallest unit of time in a Repast-based simulation. The synchronization of JADE's agents is made by informing them about the current tick. MISIA uses an agent for this - dubbed 'synchronizer' - that acts as a notifier. Furthermore, FIPA protocols are implemented with a wrapper that adapts them to Repast ticks.

Besides creating a wrapper on JADE's agents and FIPA protocols, an extra agent was included in the Repast layer. This means that for each agent created with MISIA, there are actually two agents being created. This layer also lets the synchronizer know that a new tick has passed.

2.1.2 JREP - Extending Repast Symphony for JADE Agent Behavior Components

In Gömer *et al* [GHM+11], the authors propose JREP, another platform for integrating JADE and Repast Symphony in the same framework, by means of a middleware. To demonstrate its use, the authors present an example of a smart airport and how it can be simulated using JREP.

> "JREP [is] a novel integration of JADE and Repast Symphony that efficiently combines the macro and micro perspective with an interaction layer. It allows to see not only the overall system behavior, but also the individual together with its interests, goals and the communication to others for local coordination and cooperation. Scheduling of the agents, (time) synchronization and the registration of new agents with the environment has been solved." [GHM+11]

The rationale for their choosing of JADE and Repast Symphony as working platforms is well defined: Repast provides statistical functions, real-time monitoring, graphs and history recording; it also supports external statistical programs and has good execution speed. JADE, on the other hand, is pointed as having a highly efficient message transport layer.

JREP's architecture is similar to that of MISIA. There are three layers where the first one, called *macro level* interacts with Repast, providing the scheduling mechanism; the second one, which they called *micro level*, interacts with JADE and contains the behavior objects and the third one, called *interaction level* which handles agent interaction using FIPA ACL. Agents in JREP are represented as both a Repast Agent and a JADE Agent. The main difference to MISIA is that in JREP the Repast Agent contains the reference to the respective JADE agent while is MISIA this was handled by an intermediate layer.

2.1.3 Similar work

Other works related to integration of features from Repast and JADE are available. This, however, is not the main focus of this thesis since its goal is not to use both frameworks simultaneously. That said, there is some interest in these works, as I explained in the beginning of this chapter.

2.2 FIPA standards in Repast

MISIA and JREP both attempted to complement Repast's lack of communication protocols by creating an interface with JADE's implementation of FIPA interaction protocols.

Open source projects exist that contemplate the use of FIPA ACL as a library, but none was found to be actively maintained or properly documented. Therefore, this thesis will contemplate the creation of a Java library that brings FIPA standards to Repast.

2.3 Code generation

There are multiple ways to tackle the problem of code transformations, as described in this thesis. The brute force approach would be to parse the source code, create an abstract syntax tree (AST), which represents all code constructions in a program, perform certain transformations in the tree, and then generate back the code from the new AST. Fortunately, there are free and open source projects that developers can use to do exactly this with significantly reduced effort. From the

available tools, I selected the ones that are the most relevant to this thesis, i.e. tools for Java code transformation that are open source, well documented and still supported.

2.3.1 Eclipse Java Development Tools (JDT)

The Java Development Tools are are a group of plug-ins that provide Eclipse the necessary means to become a full-featured Java IDE. These tools make up what we perceive as the "Eclipse IDE" and programmers can use these libraries in their Java projects and perform tasks that require a certain introspection of the code itself. Some of the possibilities presented by Eclipse JDT that could be interesting for this thesis are the creation of projects (in runtime and programmatically) which would allow the generated code to be shown immediately in the IDE; accessing projects in the workspace, allowing the programmer to access the project's code; finding calls to methods, very useful for mapping library calls from one framework to the other (for instance) and AST parsing, for a lower level parsing of code, allowing to directly manipulate methods and data structures [Fou].

JDT can be broken down into five main components:

APT - Annotation Processing Tool

This component can be used to parse annotations in the code. Annotations are a Java construction introduced in Java 5 and can be used to add meta information about classes, objects and methods. These annotations can then be parsed by frameworks that use the POJO (plain old Java objects) that enclose them. As an example in the context of the code generation tool, annotations could be used to select entities like agents or data structures in a JADE project that would then be visualized in a Repast chart.

Core - Java IDE headless infrastructure

The core for the Java IDE, it allows to transverse the Java element tree and find package fragments, compilation units, binary classes, types, methods and fields. This tool could be useful for mapping methods and library calls from one framework to another.

Debug - Debug support for Java

This tool makes the debug facilities of Eclipse possible. It provides functionalities such as execution control and contextual expression evaluation. It will probably not be very useful in the context of this thesis

Text and UI - Java editing support and Java IDE user interface

These components make development in Eclipse possible by providing a GUI for code editing, enriched with syntactic errors highlighting, among other interesting aids. This component is not immediately necessary for the development of the code generation tool.

2.3.2 Spoon - Program Analysis and Transformation in Java

Spoon is a tool created to take advantage of the new features introduced on the release of Java 5, namely annotations and generics. It is a Java transformation tool that uses annotation processing,

compile-time reflection and templates in a pure-Java environment to enable programmers to create their own platforms for code transformations. As illustrated by figure 2.2, these transformations occur on compile time and annotations are used as parameters for compilation. The programmer uses plain Java code to define the transformations that should occur, for instance, adding code snippets to the beginning of a method in a class. Finally, Spoon can be seamlessly integrated in an IDE such as Eclipse [PNP06, PN+06].

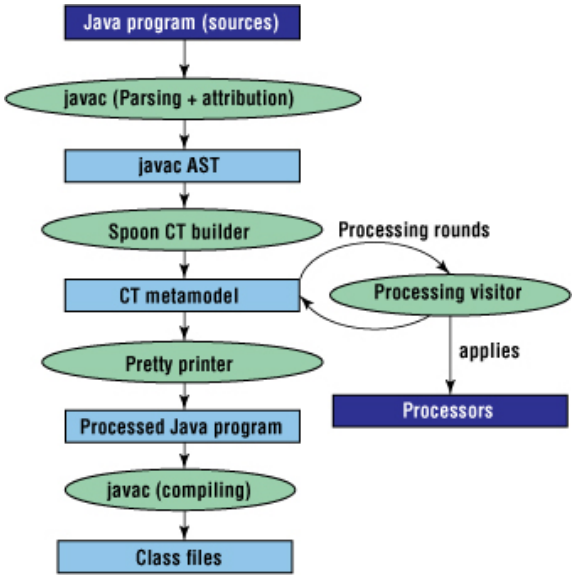


Figure 2.2: Spoon compile-time processing [PN+06]

2.3.3 ATL - ATL Transformation Language

ATL - *ATLAS Transformation Language*, is both the name of a language and its enclosing plug-in and allows the creation of model transformations. Unlike JDT and Spoon analyzed before, ATL does not focus on applying transformations to the source code.

> "ATL is applied in a transformational pattern shown in Fig. 1 . In this pattern a source model Ma is transformed into a target model Mb according to a transformation definition mma2mmb.atl written in the ATL language. The transformation definition is a model. The source and target models and the transformation definition conform to their metamodels MMa, MMb, and ATL respectively. The metamodels conform to the MOF metametamodel."

As explained in Jouault *et al* and as illustrated in figure 2.3 by the same authors, "a source model Ma is transformed into a target model Mb according to a transformation definition mma2mmb.atl written in the ATL language", which is also a model itself. The three must conform to their respective metamodels MMa, MMb and ATL which then conform to the metametamodel MOF.

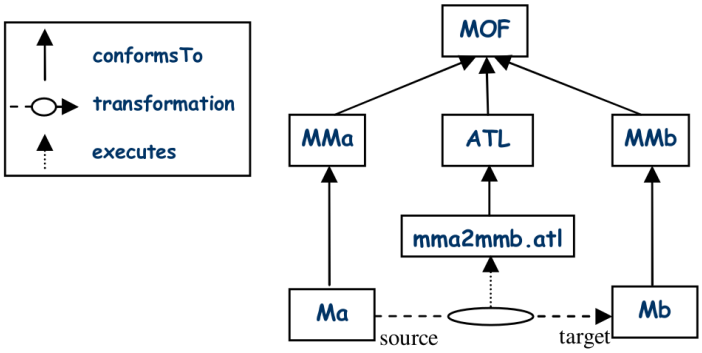


Figure 2.3: Overview of ATL transformational approach [JK06]

2.4 Summary

Although the specific problem of converting code between JADE and Repast has not been approach before in the available literature, it is clear that some related work is useful in the definition of a solution proposed in this thesis. JREP and MISIA show is that interoperability between the two frameworks is possible and helped defining the limitations of each framework. The usefulness of these works is limited, though, since the source code is not readily available and it is unknown if either project is still being developed and supported. Also, one of the goals proposed in the Introduction refers to the ability to generate code with introducing extensive changes to the original code. What JREP and MISIA propose is a platform to build new systems from the beggining using JADE and Repast.

Further testing is necessary to determine which of the code generation approaches proposed by the tools reviewed in this report should be used. As shown above, these tools provide interesting features and different methodologies to developers who wish to apply transformations on the source code of their projects in an automated way. An integration of these tools could prove to be the best approach, allowing the creation of an Eclipse plug-in with JDT and simplifying code transformations using simple Java constructions with Spoon or the specific transformation language of ATL.

Table 2.1: Comparison of code generation tools

	ATL	**Spoon**	**JDT**
Language	ATL	Java	Java
License	EPL (FOSS)	CeCILL-C (FOSS)	EPL (FOSS)
Annotation Processing	No	Yes	Yes
AST parsing	No	Yes	Yes

Table 2.1 presents a summary of the comparison of these three tools in an attempts to point out each one's strengths and limitations. From what was possible to gather about there tools without further practical testing, each tool could be useful in this thesis: ATL should be chosen if the model-based approach proves to be useful; JDT is the preferred choice for using runtime reflection and Spoon presents the simplest of the approaches to JAVA transformations in terms of programming difficulty.

Chapter 3

Future Work

This chapters presents a planification of the future work for this thesis. On the course of the next months, the planned tasks can be separated in three phases: Planning and Specification; Development and Testing; Documentation. Following the work plan is a brief description of the expected development environment, including the tools that will be used, followed by an overview of the perspectives of validation for the results of this thesis.

3.1 Work Plan

The diagram in figure 3.1 makes an attempt to summarize all the tasks planned for the next months. The work plan can be grouped in three phases: specification phase, when the details of implementation will be defined with as much detail as possible, and the foundation for the documentation of the tool will be created; the development phase, when the actual programming work load will take place, including creation and execution of tests and finally, the writing of the thesis, which will continue beyond this five month period.

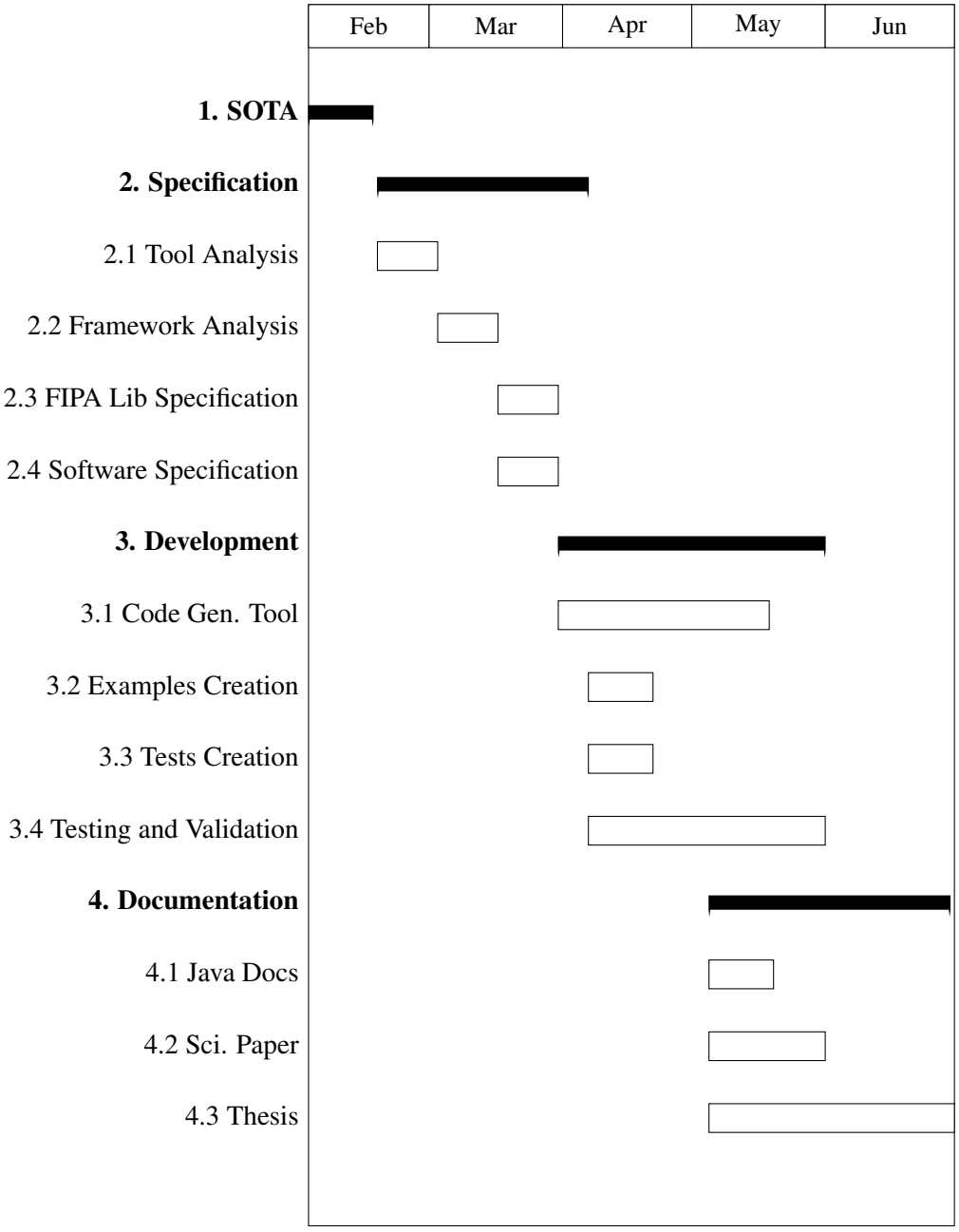
The *specification phase* starts with an analysis of the tools to use (namely Eclipse, Spoon and ATL). The first two weeks of this phase will be dedicated to the study of both tools' API and documentation (2.1), as well as actual testing of its features, followed by two weeks dedicated to a thourough study of JADE and Repast and the mapping of equivalent features between them (2.2). Next comes the definition of the requisites for the FIPA-ACL library for Repast (2.3), as well as the specification for the rest of the code generation tool. (2.4).

The *development phase* starts with a week exclusively dedicated to the creation of the general structure of the code generation tool, followed by the progressive development of all intended features (3.1). This task also includes the creation of a FIPA-ACL protocols library for Repast. Starting in the second week, sample projects will be created. These will be the test examples that the code conversion tool should be capable of converting (3.2). During this period, it is also expected that the creation of most unit tests should be completed, to ensure proper development

of the tools (3.3). Validation should start as the tests creation is completed and as the development phase progresses (3.4).

Finally, in the *documentation phase* three kinds of documents will be created. First, and essencially based on the in-code documentation already included in the development phase, Java Docs will be generated. This task will also contemplate the creation of a user manual for the code generation tool. The second document will be a scientific paper which is expectd to be submitted to an international conference or journal. Finally, this phase's main task is the writing of the thesis that will sum up all the details of the actual implementation and documentation of all results.

Figure 3.1: Work plan: Specification Phase



3.2 Development Environment

This thesis will make use of JADE 4.3.1 (6 December 2013) and Repast Symphony 2.1 (12 August 2013), which are the latest stable versions of these frameworks as of the writing of this report. Repast has implementations in languages other than Java but, since JADE is written in this language, it is only natural to use the Java implementation of Repast for the sake of simplifying the conversion process. The latest stable version of the Java platform, Java 7, is compatible with both frameworks and will be used in the development phase of this thesis.

3.3 Validation Perspectives

Because the validation of this tool's ability to correctly generate code depends entirely on the functionality of said code, unity testing can be used in order to verify conformity of a series of examples. For testing purposes four sets of unit testing will be developed. The first set of tests will cover the code of the code generation tool itself; the second one will cover the code of the Repast/FIPA-ACL implementation library; the third will include a group of example Repast code to be converted to JADE - the unit tests will be applied on the generated code; the fourth will perform the complementary tests to the third set, testing the conversion of JADE code to Repast.

While unit testing allows to verify the generated code functionality, it's also important to actually evaluate the quality of the generated MAS or MABS. The agents generated by the code conversion tool should display similar behavior to the original agent's. The work plan reserves a period for the creation of sample projects, during which simple JADE and Repast example projects will be created for testing purposes.

Chapter 4

Conclusions

Existing works have proposed ways of simplifying the use of simulations during the development of MAS. However, for this thesis, a new methodology is proposed, distinct from those proposed in the available literature.

4.1 Context Summary

Two related works were selected for analysis in this report: JREP and MISIA. To approach the need for creating MABS in JADE, these authors proposed the integration of both JADE and Repast in the same platform by means of a middleware.This approach effectively brings to the development with JADE such simulation tools as execution control GUI and a panoply of charts that display both simulation records as well as real-time information about the agents. The problem with this approach is the need of using this integrated framework from the beginning of development and even after deployment of the MAS, i.e. the MAS is dependent on both JADE and Repast.

4.2 Proposal Overview

The tool proposed in this report will allow developers of MAS in JADE to quickly generate an equivalent simulation in Repast. It will also allow a MAS to be initially designed as a MABS in Repast and converted later on.

On the course of this thesis, and as defined in the work plan (chapter 3), the details of the methodology that will be used to generate the code will be specified. This methodology will comprise ways of detecting patterns in the code, their use of certain objects, classes, structures and library resources and convert them to their equivalent in the complementary framework. Three software solutions were selected as potential tools to use in the code conversion tool to be developed: JDT, ATL and Spoon. These tools will be studied to determine which provide the most

appropriate group of features. Some of the main possibilities presented by these tools are the introspection of code, detection of code patterns, code reflection and refactoring, model transformations and AST transformations.

References

[BCPR03] F Bellifemine, G Caire, A Poggi, and G Rimassa. Jade-a white paper, sept. 2003. *Online document accessible under http://jade. tilab. com/papers/2003/WhitePaperJADEEXP. pdf (last access: March 22, 2008)*, 2003.

[Col03] Nick Collier. Repast: An extensible framework for agent simulation. *The University of Chicago's Social Science Research*, 36, 2003.

[Fer99] Jacques Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.

[Fou] The Eclipse Foundation. Eclipse java development tools (jdt). Available at http://www.eclipse.org/jdt/, last access on February 2014.

[GHM+11] Jana Gormer, Gianina Homoceanu, Christopher Mumme, Michaela Huhn, and Jorg P Muller. Jrep: Extending repast simphony for jade agent behavior components. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 149–154. IEEE Computer Society, 2011.

[GRM+11] Elena García, Sara Rodríguez, Beatriz Martín, Carolina Zato, and Belén Pérez. Misia: Middleware infrastructure to simulate intelligent agents. In *International Symposium on Distributed Computing and Artificial Intelligence*, pages 107–116. Springer Berlin Heidelberg, 2011.

[HTW04] Aaron Helsinger, Michael Thome, and Todd Wright. Cougaar: a scalable, distributed multi-agent architecture. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 2, pages 1910–1917. IEEE, 2004.

[JK06] Frédéric Jouault and Ivan Kurtev. Transforming models with atl. In *Satellite Events at the MoDELS 2005 Conference*, pages 128–138. Springer, 2006.

[MTLD08] Dawit Mengistu, P Troger, Lars Lundberg, and Paul Davidsson. Scalability in distributed multi-agent based simulations: The jade case. In *Future Generation Communication and Networking Symposia, 2008. FGCNS'08. Second International Conference on*, volume 5, pages 93–99. IEEE, 2008.

[PN+06] Renaud Pawlak, Carlos Noguera, et al. Spoon: Program analysis and transformation in java. 2006.

[PNP06] R. Pawlak, C. Noguera, and N. Petitprez. Spoon: Program analysis and transformation in java. Technical Report 5901, INRIA, 2006.

[Woo08] Michael Wooldridge. *An introduction to multiagent systems*. Wiley. com, 2008.