From Simulation to Development in MAS
A JADE-based Approach

João P. C. Lopes[1] and Henrique Lopes Cardoso[1,2]
[1]*DEI/FEUP, Faculdade de Engenharia, Universidade do Porto, Portugal*
[2]*LIACC – Laboratório de Inteligência Artificial e Ciência de Computadores, Porto, Portugal*
{*lopes.joao.pedro, hlc*}*@fe.up.pt*

Keywords: Multi-Agent Systems, Multi-Agent-Based Simulation, Model Conversion, Standards.

Abstract: Multi-agent systems (MAS) present an effective approach to the efficient development of modular systems composed of interacting agents. Several frameworks exist that aid the development of MAS, but they are often not very appropriate for some kind of uses, such as for Multi-Agent-based Simulation (MABS). Other frameworks exist for running simulations, sharing little with the former. While open agent-based applications benefit from adopting development and interaction standards, such as those proposed by FIPA, most MABS frameworks do not support them. In this paper we propose an approach to bridge the gap between the development and simulation of MAS, by putting forward two complementary tools. The Simple API for JADE-based Simulations (SAJaS) enhances MABS frameworks with JADE-based features, and the MAS Simulation to Development (MASSim2Dev) tool allows the automatic conversion of a SAJaS-based simulation into a JADE MAS, and vice-versa. Repast Simphony was used as the base MABS framework. Our proposal provides increased simulation performance while enabling JADE programmers to quickly develop their simulation models using familiar concepts. Validation tests demonstrate the significant performance gain in using SAJaS with Repast Simphony when compared with JADE and show that using MASSim2Dev preserves the original functionality of the system.

1 INTRODUCTION

A multi-agent system (MAS) is composed of autonomous intelligent agents, capable of interacting with each other (Wooldridge, 2008). Agent-based applications are in widespread use in multiple fields of research and industry and can be heterogeneous, often requiring interoperation between agents from different systems. In order to make this possible, agent technologies have matured and standards have emerged to support the interaction between agents.

The specifications of the Foundation for Intelligent Physical Agents (FIPA)[1] promote interoperability in heterogeneous agent systems. These standards define not only a common Agent Communication Language (ACL), but also a group of interaction protocols, recommended facilities for agent management and directory services (O'Brien and Nicol, 1998).

Several frameworks exist (Nikolai and Madey, 2009; Allan, 2009) that offer some level of abstraction from low level development of agent-based applications, allowing programmers to focus on a more conceptual approach in MAS design. It turns out, though, that FIPA standards (or any standards) are not supported by all of them.

Multi-agent based simulations (MABS) are sometimes used on the course of the development of a full-featured MAS – for instance, for testing purposes – for the potential gains in performance when simulating MAS. However, most platforms for MAS development are not well suited for MABS due to scalability limitations (Mengistu et al., 2008). Interest exists in solutions that make MABS more easily created using MAS development frameworks. Furthermore, an opportunity exists to partially automate the development of robust MAS from a previously tested simulation.

In this paper we focus on two frameworks, JADE and Repast. For this paper, version 4.3.2 of JADE and version 2.1 of Repast Simphony (henceforth simply "Repast") were used.

JADE (Bellifemine et al., 2007) is a FIPA-compliant, general-purpose (i.e. not focused on a single domain) framework used in the development of distributed agent applications. It is a very popular

[1]http://www.fipa.org/

MAS development framework that allows the creation of seamless distributed agent systems and complies with FIPA standards. It uses an architecture based on agent containers which allows the abstraction from the network layer, meaning that there is no difference between interacting agents running in separate machines or in the same one. However, experiments with JADE show that the platform's scalability is limited (Mengistu et al., 2008). Its multi-threaded architecture falls short in delivering the necessary performance to run a local simulation with a large number of agents, meaning that JADE is not an appropriate tool to create MABS.

Repast (Collier, 2003) is an agent-based simulation toolkit that allows creating simulations using rich GUI elements and real time agent statistics. It can easily handle large numbers of agents in a single simulation. Unlike JADE, though, Repast lacks much of the infrastructure for agent creation and interaction.

The main motivation for this paper lies in the potential performance gains when using agent simulation frameworks to produce a simulation of a MAS more complex than those typically developed with such frameworks. Some works (García et al., 2011; Gormer et al., 2011) propose, as a solution for bridging the gap between MAS development and simulation, an integration of JADE and simulation features by extending this framework with a simulation layer created from scratch, or by integrating another framework, such as Repast.

This work is aimed at developing an integrated solution for bridging the domains of simulation and development of MAS. In order to do that, two main goals were pursued:

1. First, the creation of an adapter or API that allows developers to abstract from simulation framework features and use familiar ones present in MAS development frameworks, thus creating "MAS-like MABS". This approach allows the resulting code to be easily ported to a full-featured MAS framework. JADE was selected for its wide use and FIPA-compliance nature. The developed API includes JADE's FIPA specifications for agent interaction and management.
2. Second, the development of a code conversion tool. By abstracting from the simulation tools and creating a MAS-like MABS, it becomes possible and more straightforward to engineer a tool that performs the automatic conversion of these MABS into equivalent MAS.

This solution is specially aimed at JADE developers who need to create a simulation of their already-developed MAS. By converting their code, the developer can perform tests and simulation and later convert the simulation back to a MAS, preserving all changes. JADE developers can also create simulations from scratch using frameworks like Repast using familiar JADE-like features, which would then be converted to a full features JADE MAS. Finally, this system is also of interest to Repast developers who desire to expand their knowledge of MAS development using more complex frameworks. We are aware that this kind of facilities is only valuable for true multi-agent based simulation, and not in general for any agent-based modeling and simulation approach (for which Repast is also suited).

The rest of this paper is structured as follows. Section 2 presents related work, mainly devoted at bridging the gap between MAS simulation and development tools. Section 3 makes an overview of the whole solution proposed in this paper. Sections 4 and 5 present the developed tools, SAJaS and MASSim2Dev, respectively, with more detail including their architecture and use cases. Section 6 explains how both tools were validated. Section 7 presents some conclusions and Section 8 suggests some future work to further develop SAJaS and MASSim2Dev.

2 RELATED WORK

Several frameworks exist that offer support to the development of MAS or MABS. Some are domain specific, meaning that their purpose was well defined in their conception. MASeRaTi (Ahlbrecht et al., 2014), MATSim (Balmer et al., 2008) and SUMO (Krajzewicz et al., 2012) are some examples of MABS frameworks for traffic and transports simulation.

Other works like Repast (Collier, 2003), NetLogo (Tisue and Wilensky, 2004), GALATEA (Dávila and Uzcátegui, 2000) and Plasma (Warden et al., 2010) are considered general-purpose. This list comprises only tools that are free and open source and is not meant to be exhaustive.

Some works propose approaches that are very similar to the solution proposed in this paper, namely the bridging of the domains of MAS and Simulation. MISIA, JRep and Plasma were built on top of JADE to create a simulation environment based on it. Some details of each of these three frameworks was presented next.

2.1 MISIA

MISIA is a middleware whose goal is to enhance the simulation of intelligent agents and to allow the visualization and analysis of agent's behaviour. It was de-

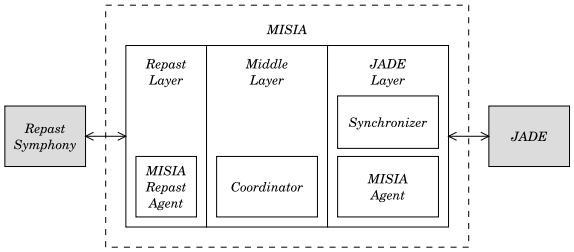

Figure 1: High-level representation of MISIA's architecture (adapted from (García et al., 2011)).

veloped by the Bioinformatic, Intelligent Systems and Educational Technology Research Group (BISITE) from Universidad de Salamanca[2]. It is no longer an active project; as a research experiment, work on this middleware evolved into other more specific tools.

MISIA's approach, as suggested by Figure 1, is to use a middle layer that acts as the bridge between two other layers that interact with JADE and Repast. By extending the agents in Repast and JADE, communicating through a coordinator and synchronizing their state, these agents work as a single one.

One of the challenges identified by the authors when re-implementing the FIPA interaction protocols was synchronizing them with the Repast tick-based simulation model. Given JADE's event-driven architecture, MISIA proposes the use of a coordinator agent that informs the JADE-Agent when a tick has passed. It also proposes its own implementation of the interaction protocols supported by JADE, making them tick-friendly.

2.2 JRep

JRep's approach is not as complex as MISIA's. By having the Repast Simphony agent encapsulate a JADE agent representation, synchronization is immediate and is assured without requiring an external coordinator. The two agent representations take care of synchronizing any state changes. Figure 2 represents the basic structure of JRep.

Each agent takes care of interfacing their respective frameworks. The interaction between agents in JRep is performed using FIPA ACL and the protocol implementations are those provided by the JADE platform. Similarly to MISIA, an Agent Representation Interface is used to introduce the concept of schedule in the JADE agent.

2.3 PlaSMA

Unlike the two previous frameworks, the PlaSMA system is based solely on the JADE platform. The distributed simulation is synchronized by entities called "Controllers" who communicate with the "Top Controller", keeping the pace of the simulation and handling agent lifecycle management as well. Figure 3 illustrates this architecture. PlaSMA, unlike MISIA and JRep, is still an active project.

[2]http://bisite.usal.es/

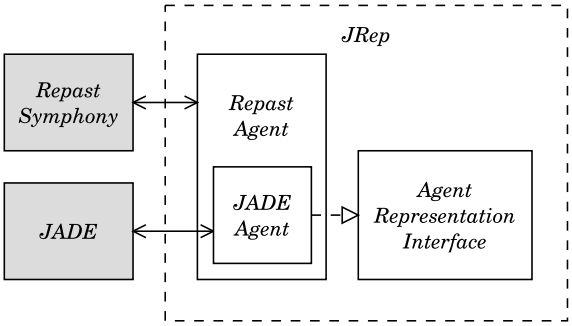

Figure 2: High-level representation of JRep's architecture (adapted from (Gormer et al., 2011)).

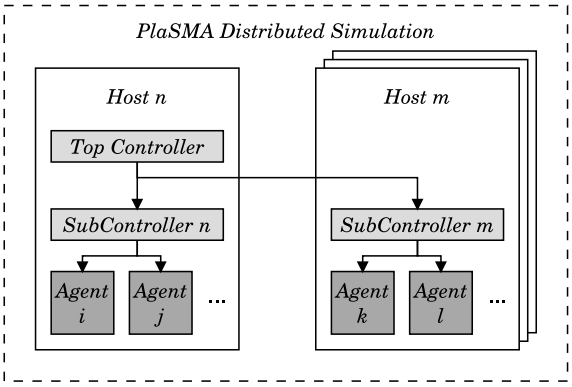

Figure 3: High-level representation of PlaSMA's architecture (adapted from (Warden et al., 2010)).

2.4 Comparison

JADE is a very rich platform but, for many simulation scenarios, the overhead it introduces has a significant impact on simulation performance (Mengistu et al., 2008).

Even though both MISIA and JRep attempt to integrate the features from both JADE and Repast, as far as Repast simulations are concerned JADE's multi-threaded infrastructure affects their performance very significantly. The main advantage of our approach is the possibility of using Repast with JADE features, namely FIPA specifications including interaction protocols, without the need to interface with JADE.

3 PROPOSAL

In order to benefit from different features, modeling the same system in multiple development environments is often necessary. The two main goals of this work have been fulfilled with two proposals:

1. The **Simple API for JADE-based Simulations (SAJaS)**, which provides a set of features present in JADE; those features were reimplemented from scratch in an attempt to simplify their internal complexity, preserving JADE-like external execution;
2. The **MAS Simulation to Development (MASSim2Dev)** code conversion tool in the form of an Eclipse plug-in that is capable of mapping JADE features to SAJaS' and vice versa, and convert applications developed using one into applications based on the other, automatically. The plugin has no Repast or JADE dependencies in the code, but makes use of a dictionary file that is specific to JADE and Repast.

SAJaS is an API meant to be used with simulation frameworks to enable JADE-based features in them, including agent interaction protocols and agent management services. It also uses JADE's concept of behaviours which encapsulate most of agents' actions. SAJaS was initially created to be used with Repast Simphony but was developed to allow its integration with other simulation frameworks.

When developing SAJaS a close resemblance to JADE's own API was kept, not only to make the code conversion more straightforward, but to allow proficient JADE developers to create SAJaS-based simulations using a familiar JADE-based API.

SAJaS does not yet implement all JADE features. A set of the most common features were selected which allows for the simulation of scenarios with some complexity, including communication between agents and the creation of custom behaviours. Features in JADE applications which are not available in SAJaS will typically be shown as errors when converted – MASSim2Dev simply ignores them and they should be re-implemented manually as necessary.

3.1 JADE and Repast

A study of JADE was performed to identify which features to implement. As Table 1 illustrates, JADE agents execute in separate threads and while this architecture facilitates the platform's distribution, JADE agents 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 numbers of agents (Mengistu et al., 2008) (García et al., 2011). This further strengthens the idea that using JADE or a JADE-Repast hybrid, as described in Section 2, is not the best course of action if performance is an important issue.

In JADE, agents are distributed in containers. Each host machine can contain multiple containers, and JADE allows agents in different containers and hosts to interact with each other. In each JADE instance, a main container exists where some special agents reside, which help in the management and address resolution of the agents. JADE agents can even hop onto another container.

While there is no equivalent infrastructure for these agents in Repast, Repast contains a "Context" object that indexes all scheduled agents. Because Repast does not support distributed applications, there is no need for address resolution services.

Table 1: Summary of JADE and Repast features.

	JADE	**Repast**
Interaction	FIPA ACL	Method calls Shared resources
Distributed	Yes	No
Simul. Tools	No	Yes
Scalability	Limited	High
Ontologies	Yes	No
Open Source	Yes	Yes
Agent Execution	Behaviours Multi thread Event-driven Assync	Scheduler Single thread Tick-driven Sync

In Repast, agent execution is scheduled manually. An agent class can contain annotations that indicate which methods should be called and when (for instance, in every simulation tick or from time to time). This approach is very flexible, allowing to schedule any method; however, more complex structures are non-existent in Repast.

JADE agent actions can be executed in their setup and takedown, but most are encapsulated in objects called Behaviours. JADE has many different kinds of behaviours that function in different ways, such as running one single task once or running them cyclically. Other behaviours implement FIPA interaction protocols, which agents can use to interact with other agents.

3.2 FIPA Specifications

SAJaS follows JADE architecture very closely, including how FIPA standards are implemented. The implemented features can be divided in three cate-

gories: Agent Management, Messaging and Interaction Protocols.

Agent Management includes the directory facilitator (DF) service and the Agent Management Service (AMS). The DF is a component that provides a yellow page service. It allows one agent to perform searches about agents rendering specific services. The AMS' purpose is to manage the agent platform, namely creating and terminating agents. Registration of each agent in the AMS is required for agents to interact, since it is from the AMS that agents obtain their own AID, needed identify the agent in communication.

Messaging includes the ACL Message, the Message Template and the Message Transport Service (MTS). The MTS is a service for transportation of ACL messages between agents. It is responsible for resolving agent addresses and it may request information from the AMS to perform this task. The ACL Message is the envelope that contains the details for communication. The Agent Communication Language (ACL) stipulates what fields a message should contain.

FIPA Interaction Protocols typify communication interactions among agents by specifying two roles: initiator (the agent starting the interaction) and responder (a participant in the interaction). Each protocol defines precisely which messages are sent by each role and in which sequence.

In JADE, agent activity is programmed through the notion of behaviours. For interaction protocols, two complementing behaviours are used for each side of the interaction, and JADEs API supports the most important protocols with built-in initiator and responder behaviours. Table 2 presents a mapping between some FIPA interaction protocols and the classes that implement them in JADE. The ones represented in this table are also available in SAJaS.

3.3 Usage Scenarios

Figure 4 illustrates the scenarios where this system is expected to be useful.

One possible scenario is when a JADE developer creates a JADE MAS and desires to perform some tests and simulations in a local and controlled environment. The developer can use this tool to convert the MAS into a MABS. Eventually, the application can be converted back if changes were made while in the simulation format.

A second possible scenario could be that of a developer that intends to create a MABS with the goal of later creating a MAS out of it. This could be a Repast developer who desires to create agent simulations that take advantage of communication and agent

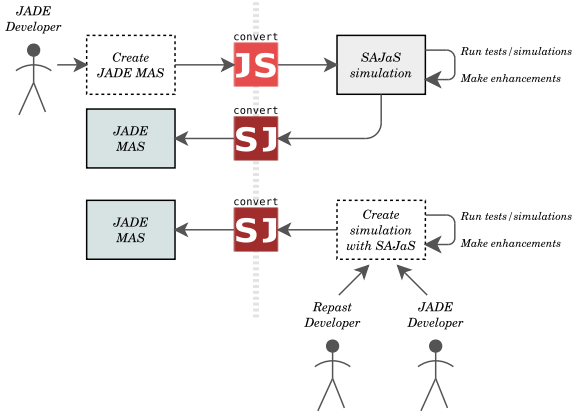

Figure 4: Possible work flows for SAJaS users. "SJ" and "JS" represent conversion from SAJaS to JADE and the reverse, respectively.

management tools from MAS, or a JADE developer that wants to create Repast simulations using familiar JADE-like tools.

A third scenario is when a developer simply wants to create a complex agent-based, FIPA-compliant simulation. In this case, there is no need for a code conversion tool, but SAJaS can be used as a standalone library.

The next sections detail both the SAJaS API and the MASSim2Dev conversion tool.

4 SAJaS

From the point of view of the MAS programmer, working with the SAJaS API feels the same as working with JADE, although limited to the presently available features. However, SAJaS has a much simpler internal architecture, which attempts to implement only the fundamental components needed for everything else to work. The most evident feature that was not ported from JADE was the network layer that enables the creation of distributed MAS. The decision of not including this JADE feature in SAJaS is based on its negative effects on the performance of applications where agent interaction is very frequent.

The diagram in Figure 5 represents a simplified version of SAJaS. Classes with a dotted border are internal and specific to SAJaS. All other classes are part of the API. The BehaviourAction and the RepastAgent offer support to Repast. More specifically, the BehaviourAction is responsible for interfacing Repast Symphony and scheduling the execution of all behaviours.

Protocol initiators are behaviours that initiate the communication, sending the first message. Responders start by waiting for this message to arrive. Be-

Table 2: Interaction protocols supported in JADE (adapted from (Bellifemine et al., 2007)).

Protocol(s)	Initiator class	Responder class
FIPA Request FIPA Query	AchieveREInitiator	AchieveREResponder
FIPA Contract Net	ContractNetInitiator	ContractNetResponder SSContractNetResponder
FIPA Propose	Propose Initiator	Propose Responder
FIPA Subscribe	SubscriptionInitiator	SubscriptionResponder

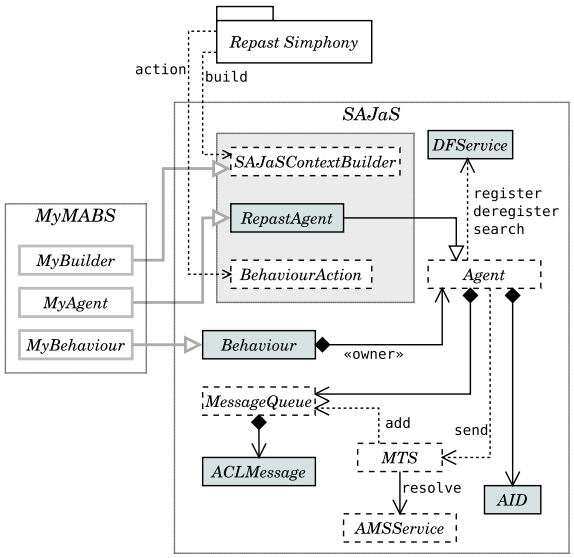

Figure 5: A simplified UML diagram of the architecture of SAJaS.

1. `onStart` is called immediately before the first call to `action`
2. `action` is called once per tick
3. `done` is called every tick to determine if the behaviour has ended and if true is returned, it triggers `onEnd`
4. `onEnd` is called right after the behaviour ended

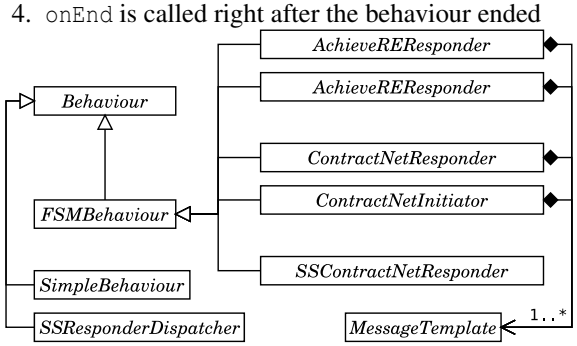

Figure 6: Simplified UML class diagram for the behaviours and protocols in SAJaS.

cause protocols have states, each responder is able to handle one conversation at a time. In JADE, a responder restarts after the conversation has finished, thus becoming ready to handle a new conversation. For the ability to handle concurrent conversations, JADE provides responder dispatchers, which may be used to handle the first message of a protocol and create on the fly an appropriate single session (SS) responder, that is, a responder whose purpose is to handle a single conversation. For this, JADEs API includes single session versions of responders. Using this mechanism, a variable number of responder behaviours may be active at any point in time, one for each of ongoing conversations.

SAJaS follows this same architecture of behaviours and protocols. The next section presents the general structure of behaviours in SAJaS and JADE.

4.1 Behaviours

Figure 6 shows the implementation of behaviours in SAJaS with more detail. The Behaviour superclass contains methods that are triggered by certain events.

The methods `action` and `done` are abstract in Behaviour, so other behaviours have to implement it. Methods `onStart` and `onEnd`, though, are implemented but do nothing in Behaviour.

The five "responder" and "initiator" classes implement the two protocols currently available in SAJaS. The FSMBehaviour, which they extend, contains a dynamic list of states and transitions. These can be registered and unregistered at runtime, typically before initiating the behaviour or in its setup. Each state is itself a behaviour.

Each protocol can be represented as a different state machine. For instance, in a Contract Net, the initiator's state starts as "sending CFP", then "waiting for replies", "waiting for result notifications" and finally "finished".

Internally, protocols in SAJaS use Java `enums` instead of the FSM Behaviour from JADE to represent their internal state machine. An `enum` is an immutable variable type while the FSM is a dynamic structure. The performance of enum-based and FSM-based protocol behaviours was compared and enum-based behaviours have shown to be significantly faster than

a dynamic FSM. The FSMBehaviour is available in SAJaS to allow the creation of custom complex behaviours by programmers.

4.2 Agent Execution

JADE execution can be concurrent and parallel, since JADE supports distributed and multi-threaded agent systems. Execution in Repast, on the other hand, is not concurrent. Repast uses a time-share type of execution, granting each agent the right to perform its tasks until they finish them, in sequence, but in no particular order.

Except when executed during their setup or takedown, agents' actions in JADE are encapsulated in Behaviours. In JADE, multiple agents can be executing their behaviours simultaneously. In Repast, however, all scheduled objects run consecutively with variable execution sequence. This schedule is one of the components in SAJaS that is specific to its Repast interface.

Even though a local application can take advantage of direct method invocation, when the simulation platform is single-threaded - as Repast is - there is a risk of the simulation stagnating if, for instance, two agents engage in a very long "conversation".

Figure 7 represents a scenario where two agents engage in a conversation that involves multiple multiple replies from both sides. With direct method invocation, the response is instantaneous but other agents do not get any time to execute in-between.

In Figure 8 on the other hand, each agent is allowed to execute one task - send one message, in this case. Messages stay waiting until the agent reads and processes it. This way, Agent C did not have to wait for the other two to finish.

In (Mengistu et al., 2008), authors concluded that increasing the granularity of the system, it is possible to improve its overall performance, even if individual tasks are delayed. The granularity of an agent task is explained as the communication-to-computation, or "a measure of the amount of computation an agent executes before entering the communication phase of one simulation time step".

In SAJaS, as in JADE, agent interaction occurs using the messaging service. Therefore, asynchronous execution is appropriate for the kind of applications developed for JADE and SAJaS. Simulations in Repast, though, usually depend on the synchrony of the environment. The use of ACLMessages, which wait in the message queue until processed, allows to maintain a synchronous execution, while simulating an asynchronous one.

To better demonstrate the differences between

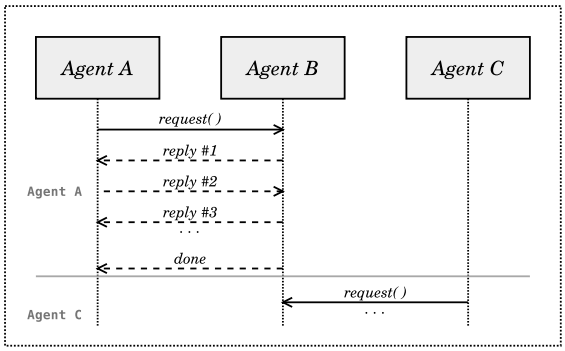

Figure 7: Communication with direct method invocation.

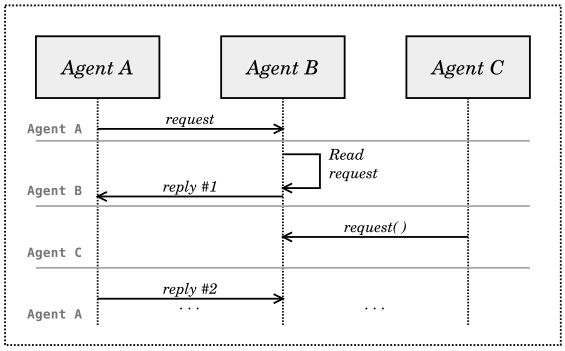

Figure 8: Asynchronous communication.

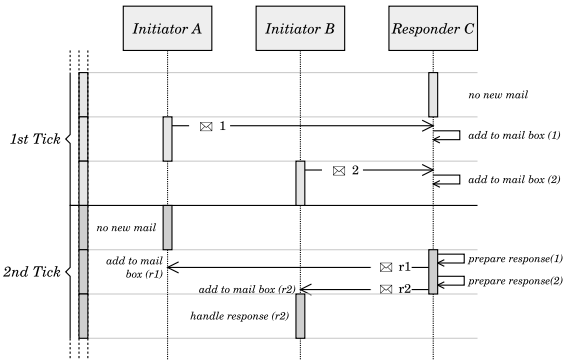

Figure 9: Communication in SAJaS.

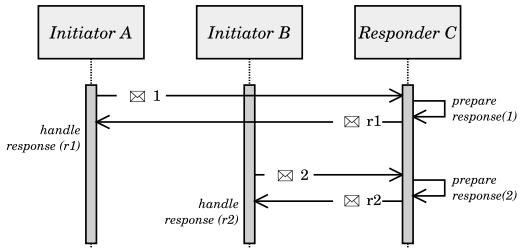

Figure 10: Communication in JADE, running in parallel.

agent execution in both frameworks, Figures 9 and 10 represents a scenario where two agents send a message to a third one who then replies. In SAJaS

(Fig. 9), messages are delivered to agent C's message queue, and processed only in C's turn. In JADE (Fig. 10), messages can arrive concurrently. Their arrival triggers an event and they are processed right away in the receiving agent's thread. In this case, agent C handles the messages as they arrive and issues the respective replies.

It should be noted that agent behaviours are the objects actually being scheduled, as each agent typically initiates multiple behaviours. The order by which Repast executes each scheduled behaviour should not be predictable: to remove the influence that a fixed execution order can have in the outcome of a simulation, the schedule is randomized every tick. As a result, it is not guaranteed that all the behaviours of a single agent are executed consecutively. This is the expected execution when working with Repast as well as with JADE (given its multi-threaded nature) and it is up to the programmer to ensure that the application does not rely on the order of execution.

5 MASSim2Dev

MASSim2Dev is an Eclipse plugin that makes use of SAJaS. It acts as a translator that changes the MABS dependencies from one platform to the equivalent classes in the other platform (see Figure 11). After conversion, no dependency to the previous platform exists in the generated project, as long as the original SAJaS simulation is not using internal components from the SAJaS library, or the original JADE application is not using elements from the JADE API that are not yet implemented in SAJaS.

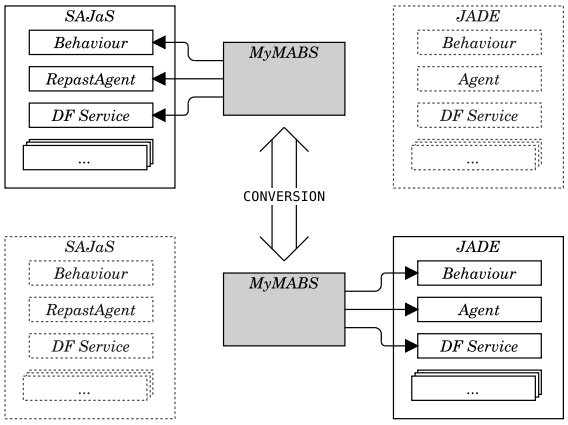

Figure 11: Representation of the conversion of code.

5.1 Plugin Execution

When the plugin is activated, it triggers a series of actions.

1. Clone the selected project;
2. Change all references to SAJaS classes in class imports;
3. Inject needed libraries in the new project and add them to the build path;
4. Fix hierarchy (e.g classes that extended `RepastAgent` must now extend `Agent`).

When the hierarchy needs to be fixed, it means that the Java type name of the superclass is not the same for JADE and SAJaS. Currently this occurs solely for the Agent class. Throughout SAJaS' code, all references to the agent use the abstract type `core.Agent`. However, agent types in simulations based on SAJaS+Repast always extend the `RepastAgent` type.

To perform the mapping of the class imports between the frameworks, a dictionary file exists within the plugin. It allows for quick upgrades to the tool in the future without having to edit the actual code. The dictionary contains annotations that inform how to deal with the hierarchy fixing problem. It also contains information about extra dependencies, such as JADE and Repast libraries.

5.2 Code Conversion

There are multiple ways to tackle the problem of code transformations. 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.

The Eclipse Java Development Tools (JDT)[3] used to develop MASSim2Dev are a group of tools integrated in the Eclipse IDE. Some of its most interesting features are the automatic cloning of projects, the handling of classes, imports, methods and fields as objects and the possibility of doing complex manipulation tasks without parsing the code. It does, however, allow the use of a high level AST for a more direct manipulation of the source code. JDT is accessible to plugin developers from within Eclipse.

6 VALIDATION

To validate both SAJaS and MASSim2Dev, a set of experiments was designed, with the aim of covering

[3]https://www.eclipse.org/jdt/

and testing all the available features. Table 3 summarizes the aims of each experiment.

The first example consists of a simple contract net between one buyer and multiple sellers. In the second example, multiple contract nets run concurrently and some of the buyers make use of available computational trust about sellers. The third example is a board game called Risk developed prior to the start of this project. The goal of this last example was to test SAJaS with an application that had not been developed specifically for this test, in order do demonstrate that performance results obtained in JADE applications generated using MASSim2Dev were not caused by faulty code generation.

6.1 Simple Contract Net

In this scenario, an agent (the buyer) intends to purchase a certain quantity of goods. The diagram in Figure 12 illustrates the interactions between the agents in this scenario.

The buyer issues a call for proposals (CFP) containing a request for supplies to all agents that announce themselves as suppliers in the DF. After receiving a CFP, the supplier replies with a PROPOSAL containing a price for each product if the demanded supply is within the seller's capacity. Otherwise, a REFUSE message will be sent to the buyer. Finally, the buyer agent compares all valid proposals, chooses the cheapest offer for each of the three products and replies with an ACCEPT PROPOSAL to the best offers, and REJECT PROPOSAL to all others.

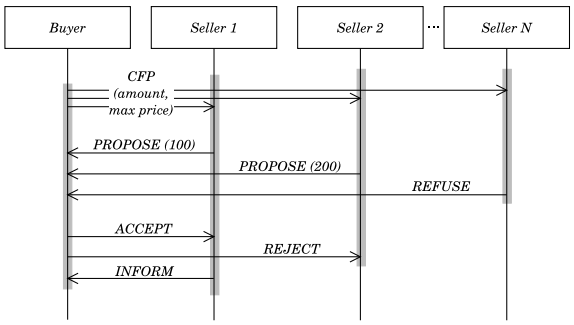

Figure 12: Sequence of agents interaction in the simple contract net scenario.

One of the difficulties of testing MABS is the potential stochastic behaviour of agents in the simulation, which makes it more difficult to obtain a clearly defined result. This first validation scenario circumvents this problem, aiming instead at comparing simulation performance. To ensure the proper comparison of results between multiple runs of the experiment and between runs in SAJaS and JADE, a predefined data set was used in all executions. This experiment focused on two simple metrics to evaluate the result: execution time and outcome. After 10 executions, the average performance of the experiment was calculated for different numbers of agents, as shown in Figure 13. The performance of the simulation based on SAJaS was significantly better, excelling when the number of agents is high. JADE was able to perform better when using two distinct containers. The outcome of this experiment consisted in identifying the agent with the lowest bid and was known a priori. As expected, in all executions with the same number of agents, this result was identical in SAJaS and in JADE.

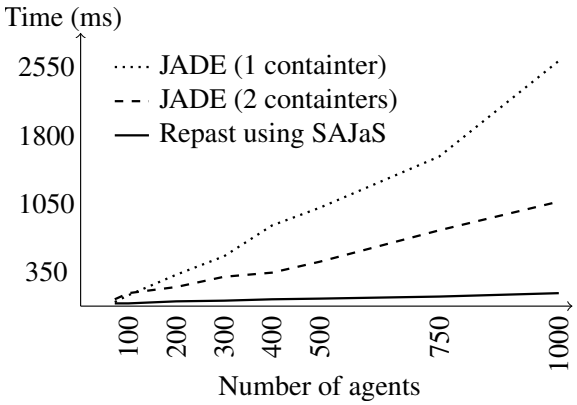

Figure 13: Average execution time on each framework in each setup.

6.2 Multiple Contract Net

This scenario is composed of multiple buyers and sellers running simultaneously. The diagram in Figure 14 illustrates one round of negotiation in this scenario involving a single buyer. After searching the DF for sellers of a given product, each buyer sends the list of sellers to the CTAgent (CT standing for computational trust). The CTAgent keeps track of the sellers' past contracts with buyers and provides information on the top sellers in terms of successful contracts.

With this information, the buyer sends a CFP only to the most trusted agents and accepts the best proposal from them. Some sellers will occasionally violate the contract after acceptance. The buyer will then inform the CTAgent if the contract was fulfilled or violated.

Some buyers are programmed to ignore trust and rely solely on the proposal. The premise is that informed buyers eventually avoid contacting sellers programmed to violate contracts more often. The goal of this experiment is to model this scenario in SAJaS, convert it to JADE and verify that the obtained results are similar.

Table 3: Features covered by each experiment.

	Simple CNet	Multiple CNet	Risk Game
ACL, DF, MTS, AMS, Behaviours	✓	✓	✓
Contract Net Protocol	✓	✓	
Achieve RE Protocol		✓	✓
SS Contract Net Protocol		✓	
FSM Behaviour			✓
Simple Behaviour			✓
Responder Dispatcher		✓	
SAJaS to JADE Conversion	✓	✓	
JADE to SAJaS Conversion			✓

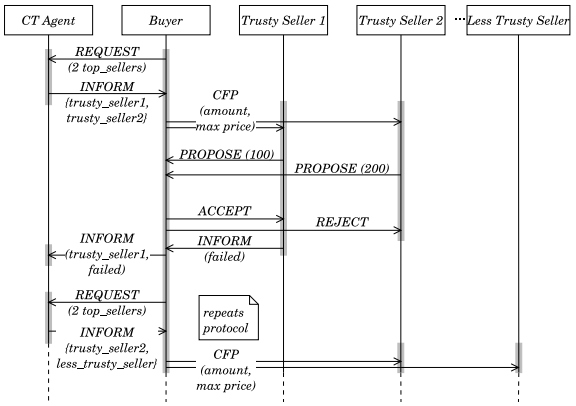

Figure 14: Sequence of agents interaction in the simple contract net scenario.

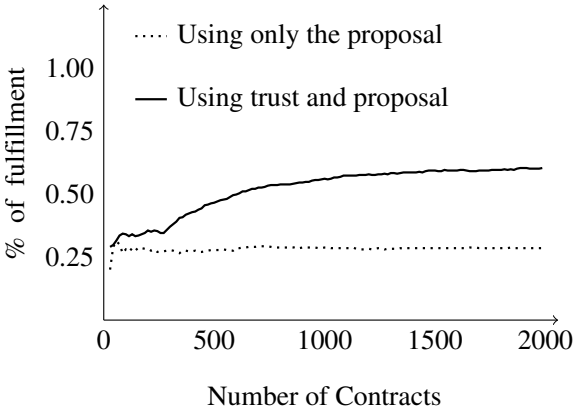

Figure 15: Average result of 5 executions of the Multiple Contract Net scenario during 2000 contracts in JADE.

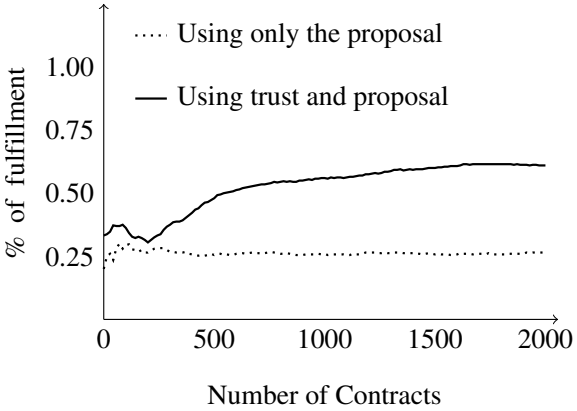

Figure 16: Average result of 5 executions of the Multiple Contract Net scenario during 2000 contracts in SAJaS.

The experiment was executed 5 times in JADE and 5 times in SAJaS. The scenario is composed of 20 buyers using computational trust, 20 not using trust and 80 sellers. A total of 2000 contracts were recorded to create the charts in Figures 15 and 16. Unlike in the first example, this one included non deterministic elements (the sellers' bids). The comparison of the results was made by analysing the charts. As shown, buyers who made use of computational trust obtained increasingly more successful contracts. The fluctuations early in the simulation are due to the initial lack of trust information. As expected and as shown in the charts, a similar outcome was observed both in JADE and SAJaS.

6.3 RISK Board Game

RISK is a multi-player strategy board game played in turns. It is a game currently owned by Hasbro and the full rules manual is available online[4]. The game used for this experiment was developed with JADE before the conception of the project described in this paper. The game is played automatically by agents, competing against each other for the conquest of a map that loosely resembles a world map and its regions.

Player agents have different behavioural architectures and are classified as aggressive, defensive, opportunistic or random. Communication occurs between the players and the game agent using the FIPA REQUEST protocol. The game also heavily relies on custom Finite State Machine Behaviours (FSMBehaviour) to control game progress. To evaluate the

[4]http://www.hasbro.com/risk/

performance of the game, logging features were introduced to the original source code of the application. Other than that, no other changes were made to the original code.

For this experiment, a match with 5 "random agents" was setup. Random agents do not follow any particular strategy of attack, defense or soldier distribution and a game with only random agents is always never-ending. To analyze the overall performance of the agent system, the game was converted from JADE to SAJaS using MASSim2Dev. The experiment was then repeated 5 times during 8 seconds and the number of game rounds was registered. The chart in Figure 17 shows that SAJaS was capable of executing many more rounds in the same period. The GUI of the game made it possible to assure that the game was executing correctly.

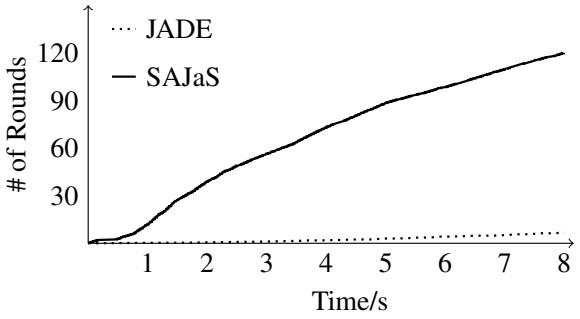

Figure 17: Performance of a Risk match with 5 random agents.

7 CONCLUSIONS

To bridge the gap between MAS development and simulation, SAJaS was created in order to allow JADE developers to create Repast-based simulations using familiar JADE features. MASSim2Dev was also developed to enable the conversion of simulations created with SAJaS into JADE MAS.

The main advantage of using SAJaS over the solutions studied in the related work is the ability to run simulations that do not need JADE at runtime, relying on Repast only. JADE is a very rich platform for the development of agent systems distributed in a network, but its architecture quickly becomes a performance hog as the number of agents in a system rises. Therefore, Repast was presented as a viable alternative to serve as the base for complex but scalable agent-based simulations. A conscious effort was made to keep a clear separation within SAJaS between Repast-specific elements and the core of this API.

Three tests were designed to validate the results of this paper. Technically, the scenarios covered all features currently available in SAJaS. The Achieve RE protocol was used in the Multiple Contract Net scenario to communicate with the CTAgent and it was widely used in Risk for all communications. The Contract Net protocol was covered in the first two experiments. The Multiple Contract Net scenario also allowed multiple concurrent contracts to take place by using the Responder Dispatcher. The creation of custom Simple Behaviours and Finite State Machine (FSM) Behaviours was covered by the Risk example scenario. This feature is also important for the creation of more custom protocols or any kind of agent behaviour.

All scenarios made use of the DF, AMS and MTS services, as well as structures like the ACL Message, Message Template, DF Agent Description and Service Description. All these JADE-based facilities and structures are widely used in JADE-based applications.

It was possible to demonstrate that bringing JADE and Repast together is not only feasible using the developed tools, but also provides increased performance when compared with a JADE MAS. Furthermore, our results show that, even though only a subset of the protocols and features from JADE are implemented in SAJaS, it is already possible to create simulations with a reasonable degree of complexity. SAJaS and MASSim2Dev are meant to be developed in the future and released to the academic community for further development, discussion and use.

8 FUTURE WORK

SAJaS and MASSim2Dev already provide programmers with the means to develop MABS with some complexity. There is, however, still room for development and future enhancements, both in SAJaS and MASSim2Dev.

The most immediate extension to this project is to enlarge the range of supported JADE features present in SAJaS. While it is important to keep the simplicity of SAJaS's internals in order to maintain good simulation performance, more elements from the JADE API can be implemented in SAJaS. The core of SAJaS allows to easily extend the API with more interaction protocols and other types of behaviours. Furthermore, SAJaS' modularity allows future extensions without changing the API and opens doors to future integration with other simulation tools.

One interesting feature in Repast is the ability to create real time visualizations of simulation data. This is possible in part because agents in Repast are

executed locally, so access to this data is facilitated. It could be interesting to include data collection and display tools that could be ported between frameworks, taking advantage of MASSim2Dev.

Possible enhancements to the plugin could include providing support for user configurations like the selection of the name and location of the newly generated project, conversion of individual classes and the automatic creation of "stub launchers" that would allow to quickly test if the generated project executes correctly.

Finally, future projects could enrich the SAJaS platform by creating new frameworks for different simulation tools. While SAJaS core takes care of the agent model and communication and other functionalities as described in this paper, a simple independent module takes care of interfacing with Repast. It is possible to create new modules to interface with other simulation tools. Such modules should implement the following features:

- Contain a structure that organizes all the behaviours from all agents;
- Allow the adition of new behaviours to the pool and initiate them by calling the appropriate method ("onStart");
- Allow the removal of behaviours from the pool and terminate them by calling the appropriate method ("onEnd" and "done");
- Schedule each behaviours' execution by calling the "action" method on each behaviour – SAJaS' Repast interface does this sequentially, but other frameworks could do it concurrently, for instance.

The development of manuals for the implementation of such interface modules is already underway, as is the organization of all project documentation and source code.

REFERENCES

Ahlbrecht, T., Dix, J., Köster, M., Kraus, P., and Müller, J. P. (2014). A scalable runtime platform for multiagent-based simulation. Technical report, Technical Report IfI-14-02, TU Clausthal.

Allan, R. (2009). Survey of agent based modelling and simulation tools. *BT Technology Journal*.

Balmer, M., Meister, K., Rieser, M., Nagel, K., Axhausen, K. W., Axhausen, K. W., and Axhausen, K. W. (2008). *Agent-based simulation of travel demand: Structure and computational performance of MATSim-T*. ETH, Eidgenössische Technische Hochschule Zürich, IVT Institut für Verkehrsplanung und Transportsysteme.

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons.

Collier, N. (2003). Repast: An extensible framework for agent simulation. *The University of Chicagos Social Science Research*, 36.

Dávila, J. and Uzcátegui, M. (2000). Galatea: A multi-agent simulation platform. In *Proceedings of the International Conference on Modeling, Simulation and Neural Networks*.

García, E., Rodríguez, S., Martín, B., Zato, C., and Pérez, B. (2011). Misia: Middleware infrastructure to simulate intelligent agents. In *International Symposium on Distributed Computing and Artificial Intelligence*, pages 107–116. Springer Berlin Heidelberg.

Gormer, J., Homoceanu, G., Mumme, C., Huhn, M., and Muller, J. (2011). Jrep: Extending repast simphony for jade agent behavior components. In *Proc. 2011 IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology, Vol. 02*, pages 149–154. IEEE Computer Society.

Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. (2012). Recent development and applications of SUMO - Simulation of Urban MObility. *Int. J. on Advances in Systems and Measurements*, 5:128–138.

Mengistu, D., Troger, P., Lundberg, L., and Davidsson, P. (2008). Scalability in distributed multi-agent based simulations: The jade case. In *2nd Int. Conf. on Future Generation Communication and Networking Symposia (FGCNS'08)*, volume 5, pages 93–99. IEEE.

Nikolai, C. and Madey, G. (2009). Tools of the trade: A survey of various agent based modeling platforms. *J. of Artificial Societies & Social Simulation*, 12(2).

O'Brien, P. and Nicol, R. (1998). Fipatowards a standard for software agents. *BT Technology Journal*, 16(3):51–59.

Tisue, S. and Wilensky, U. (2004). Netlogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, pages 16–21.

Warden, T., Porzel, R., Gehrke, J. D., Herzog, O., Langer, H., and Malaka, R. (2010). Towards ontology-based multiagent simulations: The plasma approach. In *24th European Conf. on Modelling and Simulation (ECMS 2010). European Council for Modelling and Simulation*, pages 50–56.

Wooldridge, M. (2008). *An introduction to multiagent systems*. Wiley. com.