Bridging the Gap between MAS Simulation and Development using a JADE-based API for Repast

João Pedro C. Lopes, Henrique Lopes Cardoso, João S. V. Gonçalves, Rosaldo J. F. Rossetti

Department of Informatics Engineering
Artificial Intelligence and Computer Science Laboratory
Faculty of Engineering
University of Porto
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
{joao.pedro.lopes, hlc, joao.sa.vinhas, rossetti}@fe.up.pt

Abstract. Agent-based applications and simulations are in widespread use today both in research and industry. Multi-Agent System (MAS) and Multi-Agent-Based Simulation (MABS) are two approaches to make use of multi-agent systems technology, each with distinct characteristics and goals. While open agent-based applications benefit from adopting interaction standards, most MABS frameworks do not support them. In this paper we propose an architecture, based on FIPA and JADE, for agent-based simulations, focusing on supporting agent communication using FIPA interaction protocols. Based on this architecture, we present the FooRepast API, whose goal is to bring JADE and Repast, two popular MAS and MABS development frameworks, closer together, facilitating the creation of simulations by JADE programmers and enabling an automatic portability between both tools. For illustration, we show the creation of a simulation where agents interact in a contract net. Finally, we present an early overview of a code conversion tool under development, whose aim is to automatically generate a JADE MAS from a Repast simulation that makes use of FooRepast.

1 Introduction

A Multi-Agent System (MAS) are composed of autonomous computational elements, called agents, capable of interacting with each other [8]. Agent-based applications and simulations are in widespread use in multiple fields of research and industry and can be very heterogeneous, often requiring interoperation between agents from different systems. Therefore, for agent-based technologies to mature, standards needed to emerge to support the interaction between agents.

The specifications of The Foundation for Intelligent Physical Agents (FIPA)[1] include such standards and try to promote interoperability in heterogeneous

[1] http://fipa.org/

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 [7].

Several frameworks exist that offer some level of abstraction from low level development, allowing programmers to focus on a more conceptual approach in MAS design. They are usually focused in one or more domains. It turns out, though, that FIPA standards (or any standards) are not supported by all of them [5].

In this paper we focus on two frameworks, the Java Agent DEvelopment Framework (JADE) and the Recursive Porous Agent Simulation Toolkit (Repast). For this paper, version 4.3.2 of JADE and version 2.1 of Repast Simphony (henceforth designated simply as "Repast") were used.

JADE is a FIPA-compliant, general-purpose (i.e. not focused on a single domain) framework used in the development of distributed agent applications. JADE seamlessly hides all complexity concerning its distributed architecture [1]. However, experiments with JADE show that the platform's scalability is limited, meaning that JADE is not an appropriate tool to create MABS which usually deal with a large number of agents [4] [2].

Repast, on the other hand, is a toolkit focused in the creation and execution of Multi-Agent-Based Simulation (MABS). Repast [6] is a very rich simulation tool, but does not support any open standards. As such, one of the main motivations for the creation of the software described in this paper is to enable Repast-based FIPA-compliant simulations to be created, thus improving its interoperability with JADE. As an ultimate goal, we wish to enable the conversion of Repast models into full featured JADE MAS, as well as the conversion of JADE MAS into Repast simulations. Table 1 summarizes the main differences between the two frameworks.

Table 1. Summary of JADE and Repast features.

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

The first step towards our goal is to implement FIPA interaction protocols in Repast. With this goal in mind, we propose the use of an API when developing Repast-based simulations. Our API, FooRepast (JADE-based FIPA Specifications for Simulation tools), addresses not only agent programming, but also their interaction and related infrastructural components. 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 S is also suited).

Some of the challenges of developing this software lie in balancing its complexity. MABS's execution is usually very fast paced; maintaining a simple architecture that reduces its impact on performance is therefore important. However, in order to enable interoperability and the possibility of conversion, the architecture we propose for the API is based on JADE. A compromise has been met by porting the features from JADE that we find of value in a MAS and that are needed for a meaningful exploitation of both frameworks.

One of the aims while developing this API was to make the software somewhat generic, in the sense that it was not heavily depended on a framework. We accomplished this by using a pure Java implementation that does not rely on constructs provided by Repast or JADE, as opposed to some of the similar works we discuss later in section 2. These works try to integrate both frameworks in a single environment, enabling them to use the best of JADE and Repast. They are still relevant to our proposal since they give important insight on how to implement the interaction protocols from JADE (inherently event-driven) in Repast's tick-based scheduler.

Section 3 contains a brief discussion about FIPA specifications, especially focused on the concepts that are relevant to this project and their presence in JADE. The details of the architecture, protocols and usage guidelines are defined in section 4. We give a clear overview of the details of the current implementation, while justifying the design decisions that were made and showing how it compares to JADE's own implementation.

For exemplification purposes, in section 5 we present an experiment performed in both frameworks. The experiment consisted of a contract net where a large number of agents communicate in order to buy and sell goods.

In section 6 we present our proposal for the code conversion tool that will allow programmers to convert between Repast simulations created with FooRepast and JADE applications. While still under development, we try to give an overview of its main features.

Finally, we end this paper with a summary of this paper, some conclusions and some suggestions of future work.

2 Related Work

Some works have approached the problem of complementing Repast and JADE's features by bringing them together in a single framework by means of a middleware. Following are two relevant examples we selected, MISIA [2] and JRep [3].

They propose combining Repast and JADE, allowing the creation of Repast simulations that also take advantage of JADE's networking capabilities and use of FIPA standards.

2.1 MISIA

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

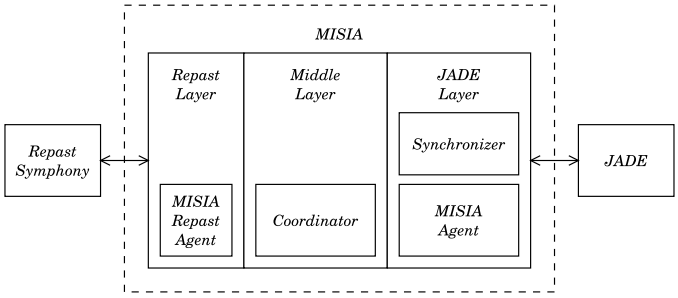

Fig. 1. High-level representation of MISIA's architecture (adapted from [2])

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

JReps's approach is not as complex as MISIA's. By having the Repast Simphony agent encapsulate the JADE agent representation, the synchronization is immediate and does not require an external coordinator. The two agent representations take care of synchronizing any state changes. Figure 2 represents the basic structure of this platform.

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 provided by the JADE platform.

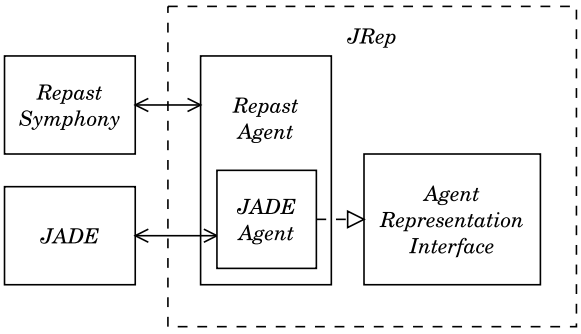

Fig. 2. High-level representation of JRep's architecture

2.3 Comparison with FooRepast

JADE is a very rich platform but, for many simulation scenarios, the overhead introduced by it has a significant impact on simulation performance [4]. FooRepast, as we describe with more detail in the section 4, uses an architecture that is conceptually very close to JADE's but tailored for Repast with no extra dependencies. In fact, the API could be used with other simulation frameworks with little adaptation.

As suggested by Figure 3, FooRepast's general structure is simpler than that of JRep and MISIA. Our API does not intend to maintain an active connection to a JADE platform, eliminating the need for synchronization. Instead, our goal is to replicate in our API the main features of JADE, allowing for a straightforward and dependency free feature mapping between our API and JADE.

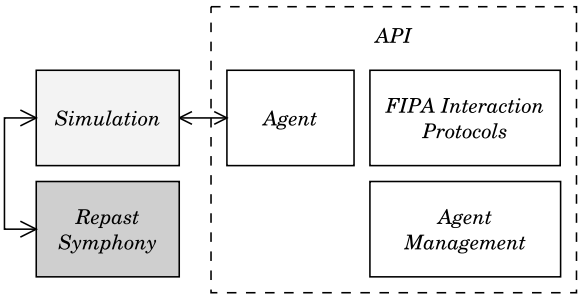

Fig. 3. Basic structure of FooRepast API

One problem that is evident in both these frameworks is the fact that they are never truly free from JADE. Even though they attempt to integrate the features from both JADE and Repast, as far as Repast simulations are concerned, JADE's multi-threaded infrastructure affect their performance very significantly.

The main advantage of our approach is, therefore, the possibility of using Repast with JADE features, namely interaction protocols, without the need to interface with JADE.

3 FIPA Specifications

FooRepast closely follows JADE's architecture regarding the use of protocols and services specified by FIPA, represented in Figure 4. The architecture of the API described in this paper includes four main concepts proposed by FIPA: the Directory Facilitator (DF), the Message Transport Service (MTS), the ACL Message and the Interaction protocols. The following is a brief description of these concepts and of how JADE uses and implements them.

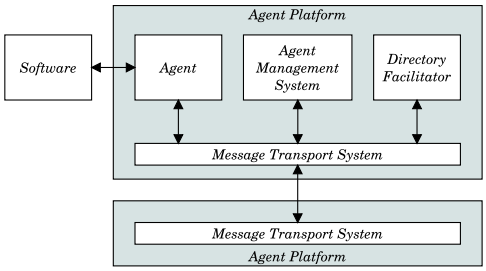

Fig. 4. Agent Management Reference Model, as specified by FIPA and implemented by JADE. FooRepast only supports a single Agent Platform.

The DF is a component that provides a yellow page service and is part of the FIPA Agent Management Specification. It allows one agent to perform searches in order to request information about other agents. Only agents that are registered in the DF will be indexed and agents can register and deregister themselves at any time.

When using the DF, agents can specify a series of restrictions that filter the search results. FIPA specifies the DF Agent Description that represents this filter and contains the fields on table 2.

The MTS is a service, also specified by FIPA, for transportation of ACL messages between agents. It is also responsible for resolving agent addresses, in order to be able to deliver those messages.

The ACL Message is the envelope that contains the details for communication. ACL is specified by FIPA, which stipulates what fields the message should contain. Table 3 was adapted from the FIPA ACL Message structure specification and contains the list of fields in a message. Not all of them are mandatory. FIPA specifies the `performative` as the only mandatory field, although the `sender`, `receiver` and `content` are usually expected to be present.

JADE uses the concept of behaviours which are adopted by agents that want to play a certain role or perform a task. Some of these behaviours are meant to be

Table 2. Fields of the DF Agent Description used to filter the results of the DF service.

Parameter	Description
`name`	The identifier of the agent
`services`	A list of services supported by this agent
`protocol`	A list of interaction protocols supported by the agent.
`ontology`	A list of ontologies known by the agent.
`language`	A list of content languages known by the agent.

Table 3. FIPA ACL Message Parameters. The highlighted fields are the ones featured in FooRepast.

Parameter	Category of Parameters
`performative`	Type of communicative acts
`sender`	Participant in communication
`receiver`	
`reply-to`	
`content`	Content of message
`language`	Description of Content
`encoding`	
`ontology`	
`protocol`	Control of conversation
`conversation-id`	
`reply-with`	
`in-reply-to`	
`reply-by`	

used in agent interaction and include two roles, as specified in FIPA interaction protocols: the initiator and the responder.

In order to create an application using these protocols, programmers only need to extend these behaviours and implement the message handlers. All the complexity regarding the interaction and networking infrastructure is hidden and taken care of by JADE, allowing the programmer to focus on the implementation of agent behaviour.

4 The FooRepast API

The modelling of agent-based systems can be accomplished using on of the wide number of available platforms, such as those surveyed in Nikolai *et al.* [5]. In order to benefit from different features, modelling the same MAS in different tools may be necessary. The API we propose is the direct result of an undergoing project to solve such necessity. More specifically, we propose a way to easily create JADE-like simulations based solely on the Repast Symphony platform.

The main feature of FooRepast is the fact that it enables the development of FIPA-compliant MABS in Repast. An objective that has also been taken in consideration in the framework's architecture is ultimately being able to perform a seamless conversion between the two frameworks. In this section we describe FooRepast's architecture thoroughly, while comparing our design decisions with those of JADE.

4.1 FIPA Specifications

As previously mentioned, FooRepast closely follows JADE's architecture and includes the four main concepts proposed by FIPA: the DF, the MTS, the ACL Message and the Interaction protocols. Our API implements the DF Agent Description used to filter DF searches. The focus of our tool is the development of local simulations (as opposed to distributed). As such, a single DF exists in each simulation. FooRepast's implementation of the MTS is simplified as well due to the absence of a distributed infrastructure; all our agents are local, therefore agent address resolution is unnecessary.

Our implementation uses a version of the ACL Message that is slightly simpler than in the one found in JADE, focusing on the most common elements (the ones highlighted in Table 3).

Finally, the main focus of the API we're proposing is the incorporation of FIPA Interaction Protocols in Repast. At the present time, we chose to include a few of the most common protocols in FooRepast, following JADE's implementation.

The FIPA protocols from JADE we chose to include in FooRepast at this point were the "request-like" Achieve Rational Effect (AchieveRE) protocol, the Propose protocol and the Contract Net protocol. The AchieveRE is a single protocol that encompasses multiple FIPA protocols, namely FIPA-Request, FIPA-query, FIPA-Request-When, FIPA-recruiting and FIPA-brokering protocols, as defined in JADE's documentation [2].

4.2 Agent Execution

In the process of integrating JADE's protocols in FooRepast, it was necessary to make adaptations to its behaviours' implementation in order to take Repast's concept of time into account. Even though a local application can take advantage of direct method invocation, for the sake of compatibility with JADE communication in FooRepast is made asynchronously.

JADE agent execution is concurrent and possibly parallel, since JADE supports distributed and multi-threaded agent systems. Agent execution in Repast, on the other hand, in 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.

[2] http://jade.tilab.com/doc/api/jade/proto/AchieveREInitiator.html

Figures 5 and 6 represent a scenario where two agents send a message to a third one who then replies. In Repast (Fig. 6), messages are delivered to the agent C's mailbox, and processed only in C's turn. In JADE (Fig. 5), messages can arrive concurrently. Their arrival triggers an event and they are processed right away. In this case, agent C precessed the messages as they arrive and issues the respective reply.

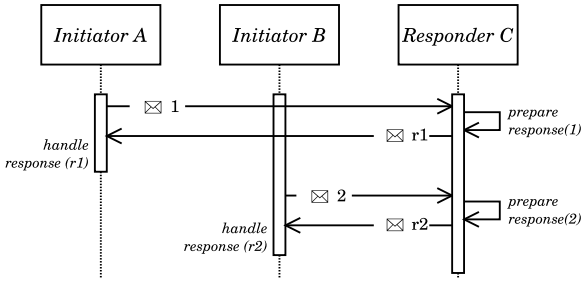

Fig. 5. Communication example in JADE. Agents are executed concurrently or in parallel.

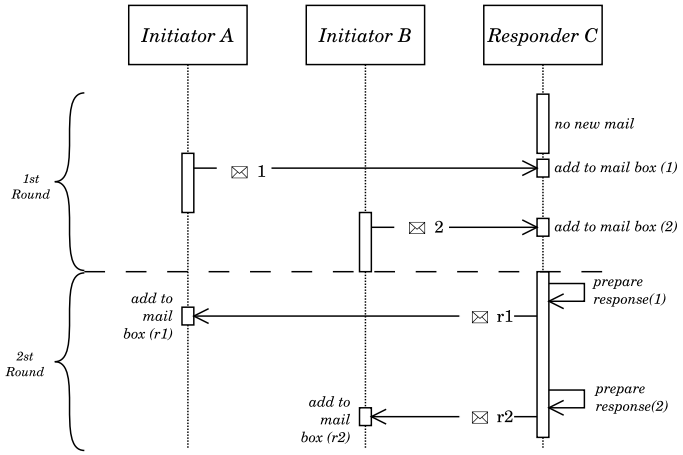

Fig. 6. Communication example in Repast using FooRepast.

It is worth noting that the order by which Repast executes each scheduled behaviour is unpredictable. In fact, it is not guaranteed that all the behaviours of a single agent are executed sequentially. This is the expected execution when working with Repast as well as with JADE and it is up to the programmer to ensure that the application does not depend on the order or execution.

4.3 Architecture

Figure 7 illustrates the details of FooRepast's architecture. Most concepts represented in this diagram are present in JADE, namely the Agent, ACL Message, Behaviour, MTS and DF service.

An agent in FooRepast contains a Mail Box and a set of Behaviours. The Behaviours have access to the agent who owns them and to its Mail Box. A behaviour that implements an interaction protocol contains a Message Template. It is used in the Mail Box to retrieve relevant messages.

The DF Service, as described before, provides the yellow page service. Agents can register or unregister themselves in the DF as well as perform searches. While tasks can be performed during agent setup, in runtime they are typically executed inside Behaviours.

To follow JADE-like communication, ACL Messages carry agent identifiers (AID) for senders and receivers. These AIDs are returned by the DF as search results and are resolved to an agent by the MTS when sending a message. In FooRepast the MTS keeps a mapping of AIDs to agents for easier access.

The "plug-in points" of the API are the Agent, the Behaviour and its derived classes. In Figure 7 all protocol definitions are implied by the generic sub-classes "ProtocolInitiator" and "ProtocolResponder".

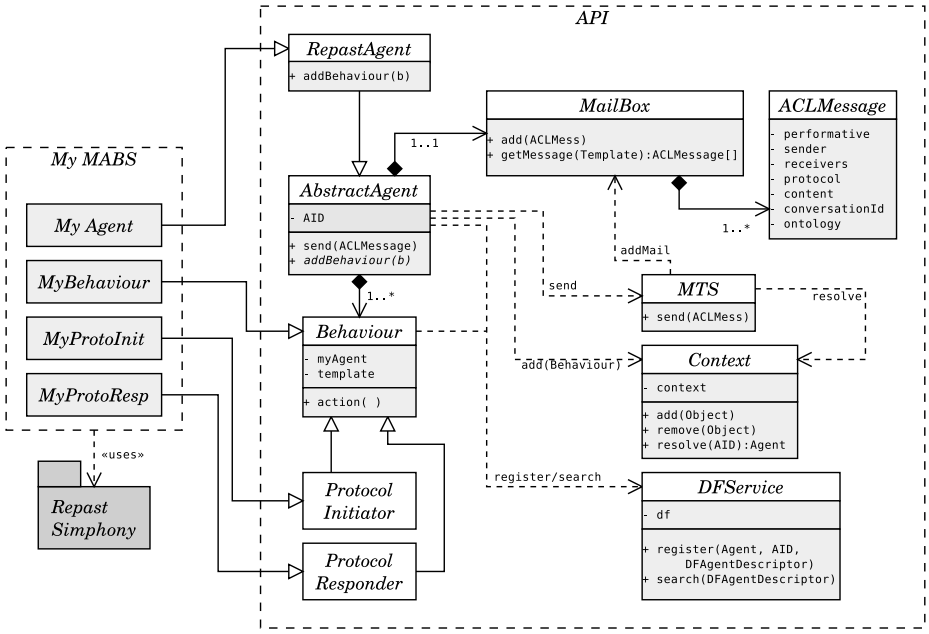

Fig. 7. Detailed architecture or FooRepast

4.4 Using Repast

Even though the API is fairly generic in its architecture, allowing its integration with different frameworks, the main candidate for integration is Repast Simphony. As such, we include an implementation of the AbstractAgent called RepastAgent. It handles the logic of scheduling the behaviour and managing of the Context. The Context is a Repast structure that contains the set of objects to be scheduled. In Figure 7, the Context in the API is a wrapper for the Repast Context.

5 Verification

To perform the verification of the API, we developed a contract net scenario. This was a deterministic test which used a fixed data set, thus having always having the same outcome. The example was implemented in Repast and then manually ported to JADE, after some adaptations.

5.1 Experiment Setup

The diagram in Figure 8 tries to illustrate the contract net created for this test. An agent - the buyer - intends to purchase a certain quantity of three kinds of goods - rice, flour and oats. Besides the quantities of each product it needs, the buyer also stipulates a maximum price for the whole deal. The buyer will issue a call for proposals (CFP) containing a request for supplies to all agents that announce themselves as suppliers.

The supplier agents have a maximum supply capacity and a price for each product. After receiving a CFP, the supplier will send a price proposal to the buyer if the supply is within the seller's capacity. Otherwise, a REFUSE message will be sent to the buyer.

Finally, the buyer agent will compare all valid proposals, choose the cheapest offer and send an ACCEPT PROPOSAL to the owner of the best offer, and REJECT PROPOSAL to every other agent.

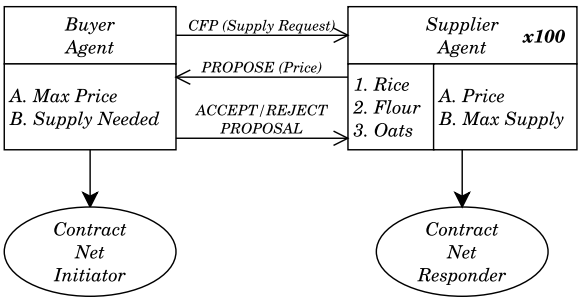

Fig. 8. Representation of the example contract net.

Using a large data set with values for prices and supply capacity in both frameworks, we ensured the proper comparison of results. For demonstration purposes, we focused on two simple metrics to evaluate our work: time and outcome.

The time was measured from be begining of the protocol, until all suppliers were notified. The JADE application was also tested in two different setups: first, with all agents running in a single container; second, with the supplier agents in one container and the buyer in a separate one (but in the same machine). All tests were performed using an Intel i7 CPU (8 logical cores). The second validation metric we used was the actual result of the protocol, i.e who was the supplier agent chosen by the buyer and its price proposal.

5.2 Results

The average performance on both experiments is represented in 9. As excepted, Repast excels when the number of agents is high. JADE was able to perform better with a lower number of agents by taking advantage of its multi-threaded architecture - more so when using two distinct containres. As studied by Mengistu et. al [4], JADE's performance drops very significantly when there is a high communication-to-computation ratio in the application.

Regarding the outcome of the protocol, an equal value was obtained in the three experiments with equal number of agents. This allowed us to verify that the behaviour of the protocol was similar identical in both implementations.

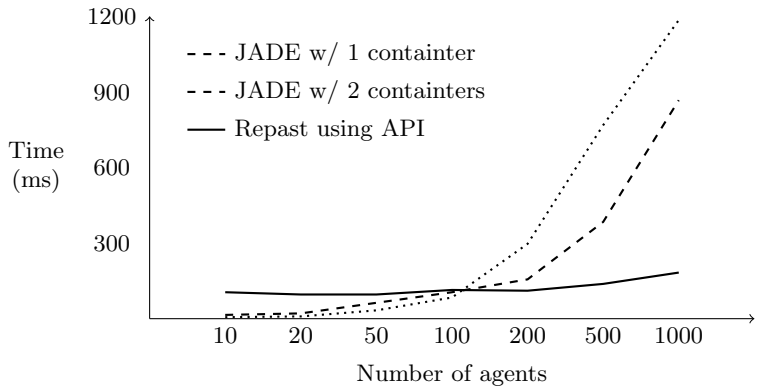

Fig. 9. Average execution time of each framework in the different experiments.

6 Future Work

In this section we specify some of the future developments that are planned for FooRepast. We also present the tool under development that initially created the need for the development of FooRepast.

6.1 Conversion Tool Prototype

The goal of the tool we propose is to allow the automatic conversion of Repast simulations into JADE applications. This conversion tool will not only allow the developer to quickly generate a MAS or create a simulation but also enables a proficient programmer in one framework to quickly get started in developing with the other framework. Figure 10 illustrates these two plausible workflows.

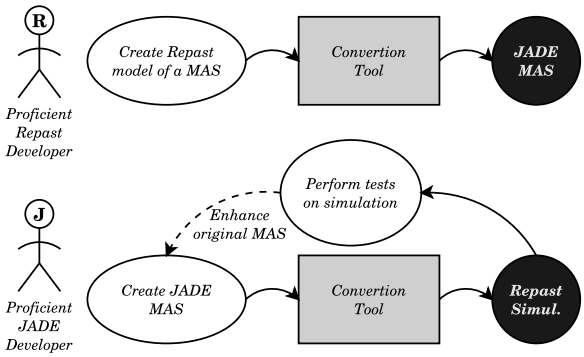

Fig. 10. Two possible workflows for FooRepast users

The code conversion tool uses the Java Development Tools (JDT) from Eclipse which enables Java programmers to perform introspection and reflection tasks with ease. This approach allows programmers to perform code transformations without having to create parsers for Java code and abstract syntax trees (AST).

The conversion tool will be developed in the form of an Eclipse plug-in, in order to be able to make use of all JDT features.

6.2 FooRepast development

Although FooRepast is already fit for development of Repast simulations, it's still an ongoing project. One important trait of FooRepast is that, although tailored for repast, it's very generic. This means that MABS built using other Java simulation tools can integrate our API successfully in the future and, therefore, use Jade-based features.

7 Conclusion

The development of MAS may require the use of tools that are not readily available in the agent-based framework originally chosen. The API architecture we propose in this paper emerges as a need to solve a concrete instance of this problem. FooRepast is our solution to integrating FIPA specifications in MABS frameworks, more specifically Repast, as well as allowing the creation of JADE-based simulations.

The goal of our API is not only to enrich Repast and other simulation frameworks, but also to allow proficient JADE developers in quickly getting started in developing simulations using familiar JADE-based tools. FooRepast also opens the door to MAS to MABS conversion.

The short experiment we described in the previous section showed that our API is capable of handling simulations with a large number of agents, which is common in many simulation scenarios. Aside from the inherent advantages of using open standards in agent systems, FooRepast's JADE-based architecture brings Repast closer to JADE. The immediate result of this architecture is that converting models built with Repast using FooRepast into JADE MAS is very straightforward.

References

1. F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. JADE: A White Paper. *EXP in search of innovation*, 3(3):6–19, 2003.
2. 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.
3. 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.
4. 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.
5. Cynthia Nikolai and Gregory Madey. Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies & Social Simulation*, 12(2):2, 2009.
6. M.J. North, T.R Howe, Collier N.T., and J.R. VOS. Fipathe repast simphony runtime system. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms.*
7. Patrick D O'Brien and Richard C Nicol. Fipa—towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.
8. Michael Wooldridge. *An introduction to multiagent systems.* Wiley. com, 2008.