SAJaS: Enabling JADE-based Simulations

Henrique Lopes Cardoso[1,2]

[1] Dep. Eng. Informática, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal
[2] LIACC – Laboratório de Inteligência Artificial e Ciência de Computadores, Porto, Portugal
hlc@fe.up.pt

Abstract. Multi-agent systems (MAS) are widely acknowledged as an appropriate modelling paradigm for distributed and decentralized systems, where a (potentially large) number of agents interact in non-trivial ways. Such interactions are often modelled defining high-level interaction protocols. Open MAS typically benefit from a number of infrastructural components that enable agents to discover their peers at run-time. On the other hand, multi-agent-based simulations (MABS) focus on applying MAS to model complex social systems, typically involving a large agent population. Several MAS development frameworks exist, but they are often not appropriate for MABS; and several MABS frameworks exist, albeit sharing little with the former. While open agent-based applications benefit from adopting development and interaction standards, such as those proposed by FIPA, MABS frameworks typically do not support them. In this paper, a proposal to bridge the gap between MAS simulation and development is presented, including two components. The Simple API for JADE-based Simulations (SAJaS) enhances MABS frameworks with JADE-based features. While empowering MABS modellers with modelling concepts offered by JADE, SAJaS also promotes a quicker development of simulation models for JADE programmers. In fact, the same implementation can, with minor changes, be used as a large scale simulation or as a distributed JADE system. In its current version, SAJaS is used in tandem with the Repast simulation framework. The second component of our proposal consists of a MAS Simulation to Development (MASSim2Dev) tool, which allows the automatic conversion of a SAJaS-based simulation into a JADE MAS, and vice-versa. SAJaS provides, for certain kinds of applications, increased simulation performance. Validation tests demonstrate significant performance gains in using SAJaS with Repast when compared with JADE, and show that the usage of MASSim2Dev preserves the original functionality of the system.

Keywords: Multi-agent systems, Multi-agent based simulation, Model conversion, Standards

1 Introduction

The field of multi-agent systems (MAS) studies how to model complex systems using agents – autonomous, intelligent entities exhibiting social abilities that

enable them to interact with each other [21]. Agent-based applications are in widespread use in multiple fields, both in research and industry. Such applications 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)[3] 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 [16].

Several frameworks exist [14, 2] that offer some level of abstraction for a proper development of agent-based applications, allowing programmers to focus on a more conceptual approach in MAS design. However, only a few of them support FIPA standards (the most notable being JADE [4]), making interoperation between agents developed using different frameworks more difficult (although some claim to be interoperable with JADE, e.g. Jason [5] and SeSAm [12]).

Multi-agent based simulations (MABS) focus on applying MAS to model complex social systems, involving a large agent population. Simulations are sometimes used in the course of development of a full-featured MAS, for the purpose of testing. However, most platforms for MAS development are not well suited for MABS due to scalability limitations [13, 17]. Popular agent-based simulation (ABS) frameworks, such as Repast [15] and NetLogo [19], lack support on advanced agent programming and multi-agent features, such as communication and infrastructural components. Given their social sciences background, it could be said that the kinds of agents such frameworks are best at modelling are not the same kind of agents considered in the multi-agent systems research community.

Still, agent-based simulation frameworks are widely used for MABS, given their support for large-scale simulations through the use of schedulers, environment spaces, data-collection and visualization facilities. In fact, there are potential gains in performance when running a MAS on top of a native simulation framework, which enables efficient large-scale testing of specific MAS properties.

Given this state of affairs, there is a growing interest in solutions that provide a richer set of programming tools for developing MABS, such as those typically available in MAS development frameworks. At the same time, an opportunity exists to partially automate the development of robust MAS from a previously tested simulation [12]. This would comprise a "write once, simulate and deploy" philosophy, where the same code could be used to run a large-scale simulation and to deploy the MAS in a distributed way.

These two points are exactly the research directions taken in this paper. We focus on two popular frameworks for MAS development and simulation, respectively: JADE and Repast. These choices are related with the fact that both of these frameworks are open-source, have a wide and lively user community and extensive online support.

[3] `http://www.fipa.org/`

JADE [4] 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 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, from the programmers perspective, between interactions among agents running in the same or separate machines. In terms of agent programming, JADE proposes the concept of *behaviour* as the building block for defining the tasks agents are able to execute. However, experiments with JADE show that the platform's scalability is limited [13]. 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 [6, 15] is an agent-based modelling and simulation system 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. The former "flavour" of Repast – Repast 3 [6] – is still, for its higher simplicity, widely used, in particular by skilled Java programmers. The current Repast suite includes Repast Simphony [15], which comprises a significantly different approach to develop agent-based simulations, including visual tools for non-programmers, and ReLogo and Java APIs.Unlike JADE, though, Repast lacks much of the infrastructure for multi-agent management and interaction. Furthermore, programming agents in Repast is a task that starts from basic Java objects, without any conceptual support for agent development.

The main motivation for this work is thus to facilitate the development of rich multi-agent based simulations taking advantage of agent-based simulation frameworks. In the end, it should be straightforward to produce a simulation of a MAS more complex than those typically developed with such frameworks. Furthermore, code written for the simulation should be portable to the full-featured version of the underlying MAS.

In order to develop an integrated solution for bridging the domains of simulation and development of MAS, two main goals were pursued:

1. First, the creation of an adapter or API that allows MAS developers to abstract from simulation framework features and use familiar ones present in MAS development frameworks, thus creating "MAS-like MABS".
2. Second, the development of a MABS-MAS conversion tool. Having a MABS that is close to its underlying MAS makes it feasible and straightforward to engineer a tool that performs automatic conversion of MABS into equivalent MAS and vice-versa.

Given our choice for JADE, this solution is particularly useful for JADE developers who need to create a simulation of their already-developed MAS. By converting their code, the developer can run simulations and perform tests and fixes, later converting the simulation back to a MAS, preserving all changes. JADE developers can also create multi-agent simulations from scratch, using

frameworks such as Repast, but taking advantage of familiar JADE-like features. Such simulations would then be converted to full-featured JADE MAS. Finally, the approach 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 modelling and simulation approach (for which Repast is primarily 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 provides an overview of the whole solution proposed in this paper, presenting SAJaS and MASSim2Dev. Sections 4 and 5 describe the developed contributions – SAJaS and MASSim2Dev – in more detail, including their design choices and use cases. Section 6 explains how both tools have been validated. Section 7 presents some conclusions and Section 8 points lines of future work.

2 Related Work

Closing the gap between simulation and development of multi-agent systems has been identified as an important research direction. In a more comprehensive approach (as compared with the work reported in this paper), the fields of agent-oriented software engineering, on one hand, and agent-based modelling and simulation, on the other, can fruitfully be integrated, as suggested in the survey by Fortino and North [9].

The opportunity for applying multi-agent simulation in domains other than pure social simulation has been identified long ago. Davidsson [7] points out this trend by highlighting some properties that make MABS appealing to other domains, namely those requiring more sophisticated agent behaviours.

The idea of enriching agent-based simulation frameworks with multi-agent features is not new. In their work on SeSAm [12], Klügl *et al.* propose using agent-based modelling and simulation for software development. They focus on designing agent systems whose validity and robustness is ensured by prior verification through simulation.

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 [1] and MATSim [3] are some examples of MABS frameworks for traffic and transport simulations; PlaSMA [20] was designed for the logistics domain. Other frameworks like Repast [6, 15], NetLogo [19] and GALATEA [8] are considered general-purpose. This enumeration is not meant to be exhaustive, including only a few examples of open-source tools.

A few attempts have been made in the direction of enriching simulation platforms with (multi-)agent features. Sakellariou *et al.* [18] proposed extending NetLogo with BDI agent programming, as well as FIPA-ACL-like communication. Their choice of NetLogo is based on its potential use as an educational platform for simulation and multi-agent systems.

In the line of our own work, in the literature we can find a few proposals to bridge the gap between MAS development and simulation by integrating JADE with simulation features, either by extending this framework with a simulation layer created from scratch, or by integrating it with an existing simulation framework, such as Repast. Some of these are discussed in the next section.

2.1 JADE Simulation Extensions

MISIA [10] 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 is no longer an active project, having evolved into other more specific tools.

MISIA's approachconsists of using 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 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".

JRep [11] proposes integrating JADE and Repast Simphony in a way that combines the macro and micro perspectives of the system with an interaction layer. 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.

Each agent takes care of interfacing its respective framework. 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.

Unlike the two previous frameworks, the PlaSMA [20] 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. Unlike MISIA and JRep, PlaSMA is still an active project.

2.2 Limitations

Distributed simulation of multi-agent systems brings non-desirable scalability issues, mainly due to synchronization overheads [17]. Furthermore, scenarios with a high communication-to-computation ratio [13], which are typical in many multi-agent applications, are largely affected by network connectivity, bringing large communication overheads. If it is often the case that agents remain idle

until a message is received, no real advantage is attained from having a distributed simulation. For this reason, general-purpose multi-agent platforms with multi-threading support, while being useful for deploying MABS on distributed resources, are not a viable approach for large-scale simulations.

JADE is a rich and powerful platform. As pointed out before, however, for many multi-agent simulation scenarios its overhead has a significant impact on simulation performance [13]. Repast Simphony is a very efficient simulation platform. However, it lacks support for agent programming concepts and multi-agent features, such as high-level communication and infrastructural components.

Even though both MISIA and JRep attempt to integrate the best of JADE and Repast, they still rely on JADE to run the agents themselves. As far as Repast simulations are concerned, JADE's multi-threaded infrastructure affects performance very significantly. This would be the main drawback of these approaches. The same is true for PlaSMA, naturally.

As we will describe in the following sections, the distinguishing feature of our approach is the possibility of using Repast with JADE features, without the need to interface with the JADE runtime system. In order to do that, we replace the JADE Agent class with a Repast-friendly one, whose scheduled execution we are able to control directly.

3 Bridging JADE-based Simulation and Development

As mentioned in Section 1, this work aims at enabling the development of rich multi-agent based simulations, capturing features available in a MAS development framework – JADE in our case. Besides taking advantage of a simulation infrastructure (such as Repast) to run the simulation proper, we also aim at using the same developed code both for the simulated run and for the actual deployment of the MAS (an approach that has been also suggested in [12]).

The contributions reported in this paper are two-fold:

- The **Simple API for JADE-based Simulations (SAJaS)** is an adapter API that enables running JADE-like simulations, connecting the underlying MAS with a simulation framework. Our rationale was to be as surgical as possible, seeking to take advantage of most JADE features which do not directly affect simulation performance.
- The **MAS Simulation to Development (MASSim2Dev)** code conversion tool is an Eclipse plug-in that offers a seamless automatic conversion between JADE and SAJaS (and vice versa). In order to do that, a mapping between JADE classes and their equivalent in SAJaS is provided, making the tool mostly independent of those two APIs.

SAJaS is an API meant to be used with simulation frameworks, enriching them with JADE-based features, such as behaviour-based agent programming, interaction protocols and agent management services. For this reason, only runtime-specific JADE classes have been replaced by new versions that enable the simulation framework to take control over agent execution. SAJaS own

classes are very similar to their JADE counterparts, in order to facilitate code conversion with MASSim2Dev. More importantly, this allows proficient JADE developers to create SAJaS-based simulations using a familiar JADE-based API.

SAJaS was initially created to be used with Repast Simphony. However, SAJAs design choices consider its straightforward integration with other simulation frameworks, as we will illustrate later.

MASSim2Dev currently provides programmers with two possible actions: (i) given a JADE-based project, convert it to a SAJaS-based project; (ii) given a SAJaS-based project, convert it to a JADE-based project.

3.1 FIPA Specifications

The need to extend simulation frameworks with multi-agent features is best addressed if we take into consideration agent technology standards, such as those proposed by FIPA. Since JADE is a FIPA-compliant platform, basing our approach on its implementation of FIPA standards allows us to inherit these multi-agent systems development features.

Through JADE, SAJaS includes FIPA standards divided into two broad categories: Agent Management and Agent Communication.

FIPA Agent Management specifications include the Directory Facilitator (DF) and the Agent Management Service (AMS). The DF is a component that provides a yellow page service. It endows agents with run-time register and search facilities that enables agents to announce themselves to the rest of the MAS and to find out about other agents in the system. The AMS is meant to manage the agent platform, namely creating and terminating agents. Agent registration in the AMS is mandatory, and results in the assignment of an agent identified (AID), needed e.g. for communication purposes. Communication is supported by a Message Transport System (MTS).

FIPA Agent Communication specifications include the notions of ACL Message, Communicative Acts and Interaction Protocols. An ACL message includes an "envelope" that contains several fields with communication details. Exploiting those fields, message templates may be used to filter incoming messages, allowing an agent to process them selectively. A communicative act is a central part of an ACL message, and is meant to disclose the communicative intention of the sender. 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.

3.2 JADE and Repast

Given our choices on JADE and Repast as target frameworks, we need to properly understand the main features and differences among them. As Table 1 shows, JADE agents execute in separate threads, which enables distributing agents among different machines. The downsize of this approach is its impact on performance when running locally a significant number of agents, which is a typical

scenario in simulation. In fact, experiments with JADE have shown that the platform's scalability is limited: the global system performance drops quickly for large numbers of agents [13]. This further strengthens the idea that using JADE or a JADE-Repast hybrid, as described in Section 2.1, is not the best course of action if performance is an important issue.

In JADE, agents are distributed across *containers*. Each host machine can contain multiple containers, and JADE allows agents in different containers and hosts to interact with each other through message exchange. In each JADE instance, a main container exists where some special agents reside (namely, the Agent Management System and the Directory Facilitator), which help in the management and address resolution of the agents. JADE agents can even hop into another container.

While not having an equivalent infrastructure, Repast Simphony does include the notion of *context* that indexes all scheduled objects. Because Repast does not support distributed applications, there is no need for address resolution services.

JADE agent actions can be executed during 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 cyclically. Other behaviours implement FIPA interaction protocols, which agents can use to interact with other agents. Together with ontology support, this comprises one of the most useful features of JADE when developing MAS. Given this strong support for communication, agents can be said to execute in an event-driven way in scenarios that rely strongly on interaction among them.

In Repast Simphony, agent execution is scheduled explicitly. Any class added to Repast's context can contain Java annotations that indicate which methods should be called and when (for instance, in every simulation tick or from time to time). Alternatively, an explicit *scheduler* may be used to define which actions to execute throughout simulation. While this approach is very flexible, more complex structures supporting agent development are non-existent in Repast. There is also no support to agent communication, which must be programmed through direct method invocations.

3.3 Usage Scenarios

The rationale behind the design of SAJaS and MASSim2Dev foresees a number of possible usage scenarios. Figure 1 illustrates the scenarios where this system is expected to be useful.

One possible scenario concerns a JADE developer who wishes to perform some tests and simulations of his JADE-based MAS, by running the system in a local and controlled environment. The developer can use MASSim2Dev to convert the MAS into a SAJaS MABS. Eventually, the application can be converted back if changes were introduced while performing tests.

A second possible scenario could be one where a developer intends to create a MABS with the goal of later converting it to a full-featured MAS. The developer could be fluent in Repast, desiring to create agent simulations that take advantage of communication and agent management tools (present in JADE and

Table 1. Comparison of JADE and Repast features.

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

SAJaS); the developed could also be experienced in JADE, intending to create Repast simulations using familiar JADE-like tools.

A third scenario may consist of a researcher that 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 describe in detail both the SAJaS API and the MASSim2Dev conversion tool.

4 SAJaS

As its name implies, the Simple API for JADE-based Simulations has been design by taking the most advantage of MAS development features offered by JADE, with the aim of providing a simulation development experience that is comfortable for JADE experienced programmers. From the point of view of the MAS programmer, working with the SAJaS API feels the same as working with JADE, although with some (minor) limitations, as we will later explain in Section 4.2. To achieve this result, only the components related with the JADE runtime infrastructure have been replaced to enable the simulation framework to control agent execution. Even so, the few core SAJaS classes are API-wise equivalent to their JADE counterparts. This facilitates usage from the point of view of the programmer and conversion through MASSim2Dev.

The most evident feature that is not supported in SAJaS is JADE's network layer that enables the creation of distributed MAS. This is, in fact, the one feature that we consider to negatively affect communication-intensive large-scale simulation performance.

Figure 2 shows a basic class diagram of SAJaS, where for clarity not all dependencies are visible. Apart from their package distribution (which follows JADE very closely), SAJaS classes can be grouped as follows:

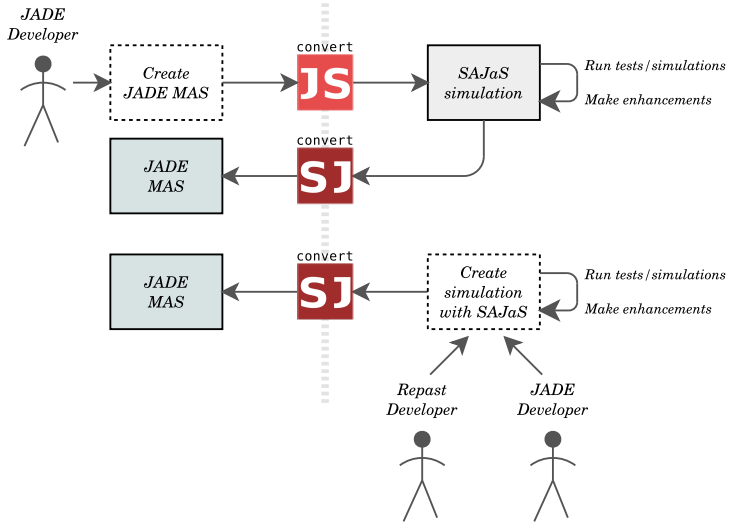

Fig. 1. Possible work flows for SAJaS/MASSim2Dev users ("SJ" and "JS" represent conversion from SAJaS to JADE and the reverse, respectively).

- *Core classes*: The need to reimplement the `Agent` class arose from the fact that in SAJaS each agent is no longer an independent thread. We thus need control over each agent's execution. Apart from that, most of JADE's own implementation of this class was kept. The reason for not extending `jade.core.Agent` is due to non-public method declarations that we needed to override. The new `AID` class, which extends the JADE version, provides a means to properly set and use a platform name.
- *Runtime infrastructure*: These classes have the purpose of launching the system, creating containers and agents, and starting agent execution: `Runtime`, `PlatformController`, `ContainerController` and `AgentController`.
- *FIPA services*: Classes `FIPAService`, `AMSService`, `DFService` and `DFAgent` correspond to FIPA services available in any JADE MAS. In particular, `DFAgent` is our implementation of the yellow page service agent, which naturally extends `sajas.core.Agent`. FIPA services had to be reimplemented due to the blocking approaches available in the JADE API, which do not work in SAJaS given its single thread nature (more on this in Section 4.2).
- *Agent dependencies*: As a consequence of having a new `Agent` class, every class in the JADE API that makes use of it had to be included in SAJaS as well, simply to redirect references to the SAJaS `Agent` class – this includes behaviour (`sajas.core.behaviours`) and protocol (`sajas.proto`) classes. Naturally, the same redirection was needed in the previous groups of classes.
- *Simulation interface*: This group of classes is specific of SAJaS, and represents its connection with the simulation infrastructure. The `AgentScheduler` interface allows agents to be added to the simulation scheduler; the specific

scheduler to use is set statically in the `Agent` class. This approach makes it easy to integrate SAJaS with different simulation frameworks. Currently, two of them are included: Repast 3 and Repast Simphony. For the latter, classes `RepastSLauncher` and `AgentAction` implement the needed Repast Simphony interfaces (also shown in Figure 2): `RepastSLauncher` is responsible for building the simulation context and setting the agent scheduler, after which application-specific JADE-related code is invoked (an abstract method is declared for that purpose); `AgentAction` implements the actual execution of scheduled agents. A similar pair of classes exists for Repast 3 (whose dependencies are omitted in Figure 2).

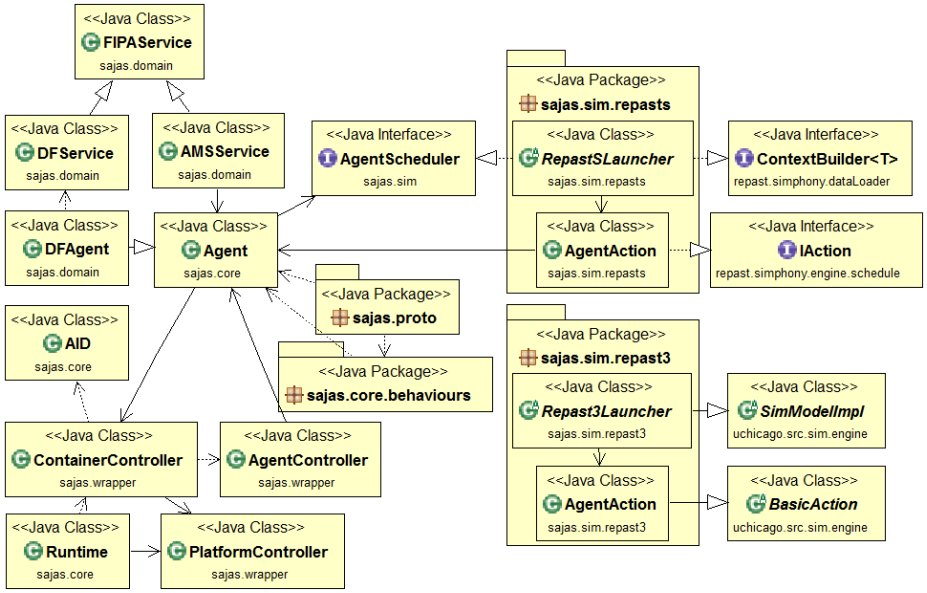

Fig. 2. A simplified UML class diagram of SAJaS.

4.1 Agent Execution and Interaction

JADE execution is concurrent and parallel, since JADE supports multi-threaded and distributed agent systems. Agent tasks are encapsulated in behaviours, which are executed in a sequential round-robin fashion. Multiple agents can be executing their behaviours simultaneously. It is up to the programmer to ensure that the application does not rely on the actual order of execution.

Execution in simulation frameworks like Repast, on the other hand, is not concurrent. Repast uses a time-share type of execution, granting each agent, in sequence but in no particular order, the right to perform its tasks. Scheduled

methods (e.g. agent executions) typically run consecutively but with variable execution order. Again, it is up to the simulation designer to ensure that simulation results do not depend on the order of execution. We thus have a different scheduling granularity: while in JADE each agent schedules its own behaviours, in Repast it is the agents that are scheduled in a shared execution thread.

Given the lack of support in Repast for agent communication, the typical way of implementing this kind of agent interaction is through method calls or shared resources. Albeit feasible, this approach brings additional challenges when considering the implementation of complex interaction protocols, such as the risk of stagnation if agents engage in a "long conversation". By taking advantage of JADE features, however, we are able to maintain its asynchronous communication mode. This enables the use of conceptually concurrent interactions among agents. In JADE (and in SAJaS), each agent has a message queue to which messages are delivered by the messaging service, and processes such messages by consuming them in appropriate behaviours.

Looking from a different perspective, we can also say that while simulations in Repast usually depend on the synchrony of the environment, by using message-waiting behaviours we are able to maintain a synchronous execution, while simulating an asynchronous one. With this approach, we can easily define protocol-based milestones that can be exploited in the course of a simulation.

To better demonstrate the differences between agent execution in both frameworks, Figures 3 and 4 represent a scenario where two agents send a message to a third one, who replies. In SAJaS single-threaded execution (Fig. 3), messages are delivered to agent C's message queue, and are processed only when it is C's turn to execute in the shared thread[4]. In JADE (Fig. 4), 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.

4.2 Current Limitations

As mentioned before, SAJaS takes a near-full advantage of JADE's features. The following are a couple of exceptions regarding the current version of SAJaS.

Handling time. The FIPA ACL message structure specification includes an optional "reply-by" parameter, to be filled-in with the latest time by which the sending agent would like to receive a reply. This parameter may be of particular use in interaction protocols, by halting waiting for the next sequential message when the indicated time has elapsed. Given the simulation bias of SAJaS, it is not clear yet to which time this should refer to. Accelerating simulation execution means that we should not use reply-by values larger than strictly necessary, which typically depends on the application in mind. Translating such timestamps to simulation ticks is probably the way to go, but enough simulation ticks

[4] This scenario is merely hypothetical; relevant is the variable agent execution order.

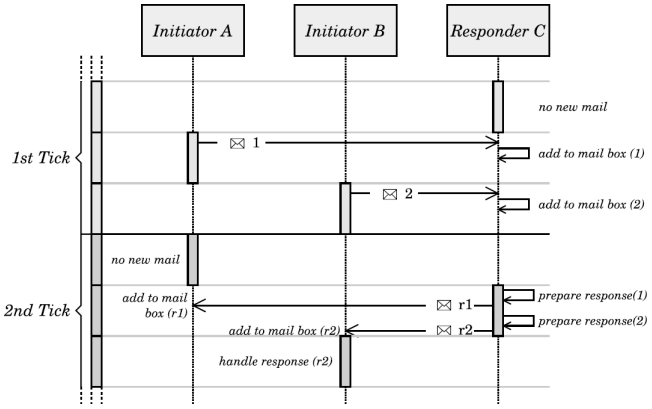

Fig. 3. Communication in SAJaS, in a shared execution thread.

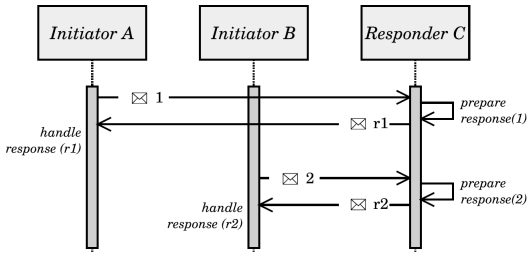

Fig. 4. Communication in JADE, each agent running in parallel.

should be allowed for responding agents to process the message and respond accordingly. This, in turn, requires having a mechanism that interfaces properly with simulation scheduling.

Two specific JADE behaviours are also hindered by the transition from JADE to SAJaS, and should thus be avoided. A `WakerBehaviour` specifies a task to be executed after a specific amount of time has elapsed. A `TickerBehaviour` specifies a periodic task to be executed in regular time intervals.

Blocking approaches. Although discouraged by JADE, programmers may use so-called blocking approaches when interacting with each other. The effect of these approaches is that certain methods in JADE's API will only return after a message is received by the agent (or even after a complete protocol has terminated). This is achieved through a `blockingReceive` operation defined in JADE's `Agent` class, or indirectly by making use of `doFipaRequestClient` in the `FIPAService` class. Given the single-threaded approach of SAJaS, or more precisely of the simulation frameworks (such as Repast) it may be aligned with, it comes to no surprise that blocking approaches do not work. Another typical usage of blocking approaches concerns (de)registering and searching the DF

(JADE's yellow page service), for which a number of blocking methods are available in the `DFService` class. Although the same effects may be obtained using non-blocking approaches (by establishing a communication with the DF agent), in SAJaS we opted to reimplement the available blocking methods in `DFService`, making sure they do not block, while achieving the desired functionality.

5 MASSim2Dev

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 finally 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)[5] used to develop MASSim2Dev is a group of tools integrated in the Eclipse IDE. Some of its most interesting features include automatic project cloning, handling of classes, imports, methods and fields as objects, as well as 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.

MASSim2Dev is an Eclipse plugin that makes use of SAJaS. It acts as a translator that changes the MAS application/simulation dependencies on one framework (JADE/SAJaS) to the equivalent classes in the other framework.When converting, a new Eclipse Java project is created: if this is a JADE project (a conversion from SAJaS to JADE), references to SAJaS classes are redirected to their JADE equivalent. On the other hand, if the new project is a SAJaS-based one (a conversion from JADE to SAJaS), references to JADE classes that have been reimplemented in SAJaS are redirected to these new versions. Any other references to JADE's API are kept.

5.1 Plugin Execution

In its current version, MASSim2Dev simply includes a couple of buttons, activating the conversion of a JADE project to SAJaS, or of a SAJaS-based simulation to JADE, respectively. When one of such buttons is pressed, the plugin is activated, performing a sequence of actions. In the case of a SAJaS-to-JADE conversion, the plugin:

1. Clones the selected project;
2. Changes all references to SAJaS classes into their JADE equivalent;
3. Removes the no longer needed SAJaS library from the new project;
4. Fixes hierarchies (e.g. classes that extended `sajas.core.Agent` must now extend `jade.core.Agent`).

[5] `https://www.eclipse.org/jdt/`

A similar sequence of actions is fired in the case of a JADE-to-SAJaS conversion. In that case, the SAJaS library is added do the project's build path.

In order to map class imports between JADE and SAJaS, a dictionary file is included. This approach accommodates future SAJaS upgrades, or its interface with further simulation frameworks, without having to change MASSim2Dev.

5.2 Handling JADE Updates

As mentioned in Section 4, when developing SAJaS we have tried to make the smallest possible changes to JADE's API, with the aim of incorporating in SAJaS-based simulations all the features that JADE programmers have available. Given the continuous development of JADE, however, new releases of that framework could imply a significant recoding of SAJaS.

It turns out that because of the generic mode of operation of MASSim2Dev, we are able to use it to perform most of that recoding effort, by providing a dictionary file that indicates which are the classes that need to be mapped. It should be evident from Section 4 which are the classes requiring our attention.

6 Validation

In order to illustrate the validation of both SAJaS and MASSim2Dev, in this paper we make use of two experimental scenarios that cover all relevant features. All experiments have been run[6] on three frameworks: JADE, SAJaS making use of Repast 3, and SAJaS making use of Repast Simphony.

The first experimental scenario covers most JADE programming features, and is described in Section 6.1, together with experimental runs and results.

The second experimental scenario starts from a JADE-based implementation of the Risk board game, which was developed prior to the start of this project. The Risk application is converted to SAJaS using MASSim2Dev. Section 6.2 describes this experimental scenario, together with experimental runs and results.

6.1 The Service Consumer/Provider Scenario

In this scenario, service consumers establish contract-net negotiations with service providers. A protocol initiator (the consumer) starts a FIPA-CONTRACT-NET by issuing a call-for-proposals (CFP) to all providers registered in the DF. Each responder (provider) PROPOSEs a price. Finally, the consumer chooses the cheapest proposal and replies with ACCEPT/REJECT-PROPOSALs accordingly. The execution of the service by the winning service provider may succeed or fail, as there are good and bad service providers. When the provider sees that it is going to fail, it may subcontract the service execution to another service provider randomly chosen, by sending a REQUEST (thus initiating a

[6] We have used a 64 bit Intel Core(TM)2 Duo CPU E8500, 3.16GHz, 6 GB RAM machine.

FIPA-REQUEST protocol); the subcontracted service provider may, again, succeed or fail. An INFORM or a FAILURE message is sent to the service consumer, respectively. Some service consumers will be paying attention to service execution outcomes. In this case, they will start contract-net protocols only with a number of the best providers in the market, according to their own experience.

This scenario exploits most features that a programmer may want to make use of in JADE[7], including:

- The yellow page service, used for registering, searching and subscribing.
- Several kinds of behaviours, including protocol-related ones, cyclic behaviours and wrapper behaviours.
- Several kinds of protocols available in JADE, including FIPA-SUBSCRIBE (to the yellow page service), FIPA-CONTRACT-NET and FIPA-REQUEST, as well as responder dispatchers and register behaviour handlers.
- Languages and ontologies, used for the content of ACL messages.

We have run two sets of experiments based on this scenario. The first one tries to show the similarity of results when using each of the three frameworks (JADE, SAJaS+Repast3 and SAJaS+RepastS). While the exact scenario details are not determinant, we simply want to show that service consumers that filter out bad service providers tend to get more successfully executed services. In this experiment, we have 10 consumers negotiating with all providers, 10 consumers negotiating only with the five best providers, and 20 providers (half good, half bad). Each service provider makes random proposals within a fixed random range; furthermore, good providers have a 0.8 probability of successfully executing the service, while for bad providers this probability drops to 0.2. In each experiment, each consumer establishes a sequence of 100 contract net negotiations.

Figure 5 shows that the outcomes of simulation are similar for each of the three frameworks. Values shown comprise average results from 5 simulation runs. This proves that scenario conversion between JADE and SAJaS obtains equivalent systems. In fact, experiments free from random factors (which are not included in this paper) show identical results.

The second set of experiments compares execution times, for different numbers of agents. Figure 6 plots the results for each framework. The value of n represents the number of consumers of each type, while the total number of providers is $5 \times n$. Every thing else is configured as in the previous experiment.

It is clear that SAJaS, both when paired with Repast 3 or Repast Simphony, outperforms JADE in terms of simulation performance.

6.2 The Risk Board Game Scenario

RISK is a multi-player strategy board game played in turns[8]. The game implementation used for this experiment was developed with JADE before the

[7] We point the reader to the JADE documentation for details on these features.

[8] The reader can find details about Risk at `http://en.wikipedia.org/wiki/Risk_(game)`

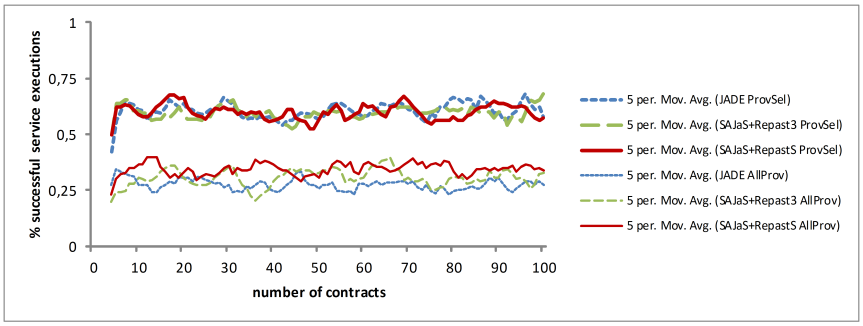

Fig. 5. Outcome comparison: thinner lines correspond to consumers negotiating with all providers (AllProv), while thicker lines correspond to those negotiating with the best providers only (ProvSel).

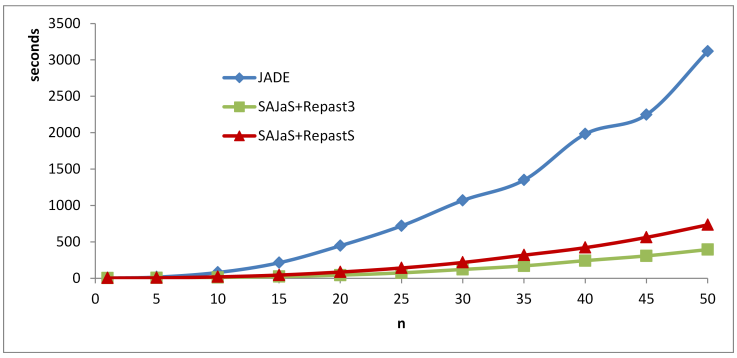

Fig. 6. Simulation performance for different numbers of agents (n consumers of each type and $5 \times n$ providers).

conception of the project described in this paper. The game is played automatically by software agents, competing against each other for the conquest of a map that loosely resembles a world map and its regions.

Playing agents have different playing skills 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 (supported by JADE through the `FSMBehaviour` class) to control game progress. To evaluate the performance of the game, logging features were introduced to the original source code of the application, in order to record the number of rounds executing in each second. No other changes were made to the original code.

For this experiment, a match with five "random agents" was set up. Random agents do not follow any particular strategy of attack, defence or soldier distribution; a game with random agents only is always never-ending. To analyse

performance using different runtime frameworks, the game was converted from JADE to SAJaS using MASSim2Dev.

The game was repeated 3 times. Average results are shown in Figure 7, for the first 20 seconds of the game. As can be seen, SAJaS has a much better execution performance in an initial simulation period, which we attribute to the much faster setup phase as compared to JADE. Given the low number of agents that are executing and the turn-based nature of the Risk game, the two frameworks have a comparable execution performance after this first period. Although not imposing a strong overhead in terms of parallel execution threads, the executing behaviour of agents in Risk strongly relies on communication (agents are idle most of the time), which explains the fact that the lines in Figure 7 are mostly parallel from second 15 onwards.

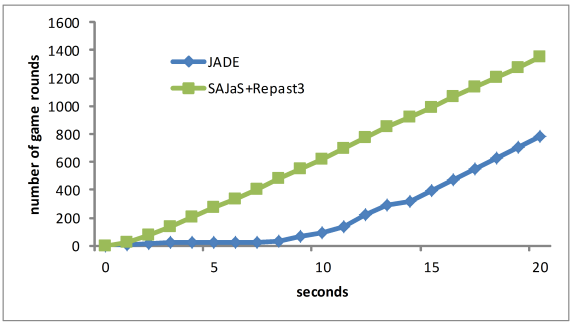

Fig. 7. Performance of a Risk match with 5 random agents.

7 Conclusions

When developing multi-agent systems, it is useful to run simulations for the purpose of testing. Most MAS development frameworks, however, are not well suited for simulation, mainly due to scalability limitations. In fact, some of such frameworks, such as JADE, focus instead on deployment features of the developed MAS, such as the possibility to run the system distributed among machines – a crucial aspect if multi-agent systems are to be applied in real-world scenarios.

Agent-based simulation frameworks, on the other hand, do not offer significant support for agent and multi-agent programming, such as high-level communication and agent building blocks, leaving the programmer with simple Java objects to program with. Using simulation frameworks for simulating preliminary versions of multi-agent systems is thus a hassle, also because of the need to recode a significant part of the simulation if it is to be later deployed as a full-featured MAS.

Given the growing interest in solutions that provide a richer set of programming tools for developing MABS, our proposal is meant to take advantage of the

best of both worlds, by providing a simulation engine-based infrastructure for developing multi-agent systems. Two advantages are offered with SAJaS. First, the MABS programmer has a rich set of multi-agent programming features offered by JADE, while being able to explore simulation-related features offered by the simulation infrastructure, such as Repast. Second, a same implementation can to a great extent be used both for simulation and for deployment purposes. The MASSim2Dev tool helps on automating this transition, in both directions.

Our experiments have shown the equivalence of a multi-agent implementation when used for simulation and deployment. Furthermore, by using a simulation infrastructure, JADE-based simulations become practical. Repast simulations scale much better than those executed with the JADE runtime infrastructure. A conscious effort was put in keeping a clear separation within SAJaS between Repast-specific elements and the core of SAJaS API. This enables the future integration of SAJaS with other simulation infrastructures.

It should be noted that simulation performance gains are not universal. The application scenarios where MAS simulations are expected to obtain higher gains by making use of SAJaS are those where agent interaction through communication plays a main role. If, on the other hand, communication overheads are bearable and outbalanced by the benefits of having a distributed simulation infrastructure that enables a parallel execution of the agents in the MAS, then SAJaS might not be the best approach. In such cases we can say that computation plays a more important role, and thus distributing it through a number of cores is a more sensible approach.

8 Future Work

Some lines of future work on SAJaS and MASSim2Dev will be pursuit. Although SAJaS is already strongly integrated with JADE, as mentioned in Section 4.2 it currently has at least two limitations. The notion of time is the most critical in terms of correspondence between the simulation and deployment versions of a multi-agent system. The fact that time is meaningful in a MAS is related to potential network problems or computation time. These issues mostly disappear when running a local simulation: no network problems can affect it, and in many cases agent-based simulations assume simple agent behaviours, with minimal computation times. It is therefore not clear how time-handling in JADE-based communication should be ported to SAJaS, if not to simply ignore the possibility of timeouts. This is something we intend to investigate.

The second limitation is related with JADE's blocking communication approaches. Encompassing this feature in SAJaS may be justified for extending the coverage of JADE features. But in any case, it is always possible to reprogram a MAS that makes use of these approaches to a version relying only on non-blocking approaches.

Apart from the possibility of exploiting message-waiting behaviours to achieve synchronisation, SAJaS does not include any other synchronization mechanism. This is an important distinction as compared to other works, such as [10]. Given

its reliance on the underlying simulation framework, however, it should not be too difficult to implement a more robust synchronization approach, by upgrading the `sajas.core.Agent` class with appropriate data members and methods.

The modularity of SAJaS allows future extensions without changing the API and opens doors to future integration with simulation frameworks other than Repast. Doing so may enlarge the community of SAJaS potential users.

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[9]. 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 MASSim2Dev plugin include providing support for user configurations, such as the selection of the name and location of the newly generated project, and the automatic creation of "stub launchers" that would allow to quickly test if the generated project executes correctly. As mentioned in Section 5.2, with proper configuration MASSim2Dev can also be used to automatically generate new SAJaS versions triggered by JADE updates.

Finally, SAJaS and MASSim2Dev are being released[10] to the academic community for further development, discussion and use.

Acknowledgments. The author would like to thank João Lopes for his initial work on SAJaS and MASSim2Dev, and also João Gonçalves and Pedro Costa for providing the source code of their JADE-based Risk game implementation.

References

1. T Ahlbrecht, J Dix, M Köster, P Kraus, and Jörg P Müller. A scalable runtime platform for multiagent-based simulation. Technical report, Technical Report IfI-14-02, TU Clausthal, 2014.
2. Rob Allan. Survey of Agent Based Modelling and Simulation Tools. Technical Report DL-TR-2010-007, Science and Technology Facilities Council, Warrington, U.K., 2010.
3. M Balmer, K Meister, M Rieser, K Nagel, Kay W Axhausen, Kay W Axhausen, and Kay W Axhausen. *Agent-based simulation of travel demand: Structure and computational performance of MATSim-T*. ETH, Eidgenössische Technische Hochschule Zürich, IVT Institut für Verkehrsplanung und Transportsysteme, 2008.
4. Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.
5. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Ltd, 2007.
6. N Collier. Repast: An extensible framework for agent simulation. *The University of Chicagos Social Science Research*, 36, 2003.

[9] We did not take advantage of these features when collecting results from our experiments, since they are not available in JADE.

[10] `http://web.fe.up.pt/~hlc/doku.php?id=SAJaS`

7. Paul Davidsson. Multi agent based simulation: Beyond social simulation. In Scott Moss and Paul Davidsson, editors, *Multi-Agent-Based Simulation*, volume 1979 of *Lecture Notes in Computer Science*, pages 97–107. Springer Berlin Heidelberg, 2001.
8. J Dávila and M Uzcátegui. Galatea: A multi-agent simulation platform. In *Proceedings of the International Conference on Modeling, Simulation and Neural Networks*, 2000.
9. G. Fortino and M. J. North. Simulation-based development and validation of multi-agent systems. *J Simulation*, 7(3):137–143, Aug 2013.
10. E García, S Rodríguez, B Martín, C Zato, and B Pérez. Misia: Middleware infrastructure to simulate intelligent agents. In *International Symposium on Distributed Computing and Artificial Intelligence*, pages 107–116. Springer Berlin Heidelberg, 2011.
11. J Gormer, G Homoceanu, C Mumme, M Huhn, and J Muller. 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, 2011.
12. Franziska Klgl, Rainer Herrler, and Christoph Oechslein. From simulated to real environments: How to use sesam for software development. In Michael Schillo, Matthias Klusch, Jrg Mller, and Huaglory Tianfield, editors, *Multiagent System Technologies*, volume 2831 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin Heidelberg, 2003.
13. D Mengistu, P Troger, L Lundberg, and P Davidsson. 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, 2008.
14. C Nikolai and G Madey. Tools of the trade: A survey of various agent based modeling platforms. *J. of Artificial Societies & Social Simulation*, 12(2), 2009.
15. M North, T Howe, Collier N., and R Vos. The repast simphony runtime system. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, 2005.
16. P O'Brien and R Nicol. Fipatowards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.
17. Dirk Pawlaszczyk and Steffen Strassburger. Scalability in distributed simulations of agent-based models. In *Winter Simulation Conference*, WSC '09, pages 1189–1200. Winter Simulation Conference, 2009.
18. Ilias Sakellariou, Petros Kefalas, and Ioanna Stamatopoulou. Enhancing netlogo to simulate bdi communicating agents. In John Darzentas, GeorgeA. Vouros, Spyros Vosinakis, and Argyris Arnellos, editors, *Artificial Intelligence: Theories, Models and Applications*, volume 5138 of *Lecture Notes in Computer Science*, pages 263–275. Springer Berlin Heidelberg, 2008.
19. S Tisue and U Wilensky. Netlogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, pages 16–21, 2004.
20. Tobias Warden, Robert Porzel, Jan D. Gehrke, Otthein Herzog, Hagen Langer, and Rainer Malaka. Towards ontology-based multiagent simulations: The plasma approach. In *European Conference on Modelling and Simulation, ECMS 2010, Kuala Lumpur, Malaysia, June 1-4, 2010*, pages 50–56, 2010.
21. Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.