ISBN 978-83-60810-22-4

Agent.GUI: A Multi-agent Based Simulation Framework

Christian Derksen
DAWIS
University of Duisburg-Essen,
Schützenbahn 70,
45127 Essen, Germany
Email:
christian.derksen@icb.uni-due.de

Cherif Branki
School of Computing,
University of the West of
Scotland, Scotland
Email: cherif.branki@uws.ac.uk

Rainer Unland
DAWIS
University of Duisburg-Essen,
Schützenbahn 70,
45127 Essen, Germany
Email:
rainer.unland@icb.uni-due.de

***Abstract*—Multi-agent based simulations (MABS) of real world scenarios are attracting growing interest. Complex real world scenarios require deep knowledge and expertise which can only be provided by specialists in the application area. However, it cannot be expected that such experts understand agent-based technology and simulation. Consequently, tools are required, which deliver a high level, easy usable interface.**

In this article we propose a new simulation framework based on the JADE framework. Besides extensions to deal with the time aspect, agent/environment interaction, visualization and load balancing, we also address the usability of the tool for specialists from different domains. For this, our framework, called Agent.GUI, provides an easy to use, customizable graphical user interface. Overall, Agent.GUI is a powerful tool for the development of multi-agent based simulations.

I. Introduction

Multi-agent based simulation (MABS) has been receiving increasing interest in the recent years. One reason for this can be seen in the fact that the agent-paradigm allows the mapping of real world entities to autonomous software agents as a first approximation. Exploiting further skills of an agent, like the ability to communicate, learn or reason, can result in further benefits and new solutions [6], [18]. In fact, MABS as a sophisticated alternative to traditional simulation techniques attracts growing interest in a broad range of disciplines. Examples for their practical as well as scientific deployment are traffic simulations [16], crisis management, energy markets or scheduling problems [10], [15], [11].

But complex scenarios from different domains often bring their own complexity with them. It can not be assumed that these domain-specific experts can understand, build or even control the execution of an agent-based simulation - simply due to its inherent complexity.

In this article we propose a new simulation framework that is based on JADE [2]. On the one hand, JADE is extended by specific functionalities for simulation purposes like time and agent synchronization, agent/environment interaction, visualization and load balancing in order to simplify the work for an agent-based developer on such concerns as much as possible.

On the other hand, a main focus is on users, who are not familiar with multi-agent systems or distributed simulations. Here our framework provides a multi-language based GUI that can easily be customized to domain specific requirements.

In this paper we will focus on two important aspects of our framework, the ability for extending the frameworks graphical user interface and the bidirectional interaction of agents with their environment by using our adapted service for simulations.

This article is structured as follows. The next section will give some background information and will motivate our work, while section 3 will present the above mentioned capabilities of our framework. In section 4 we will show an application, which compares the use of our simulation service to the use of ACL for the agent/environment interaction. Section 5 presents the related work. Finally, section 6 concludes the paper.

II. Background and Motivation

Agents can be regarded as autonomous, problem-solving computational entities with social abilities that are capable of effective proactive behavior in open and dynamic environments. There are a number of definitions for agents (e.g. [17], [12]) and most of them are associating the properties autonomy, social ability, reactivity, proactively and intelligence to agents.

Considering reactive or proactive behaviors of agents, agents can exhibit different levels of sophistication. Literature discusses different types of sophistication (e. g. deliberative, learning or simple reactive one) which we do not want to repeat here.

A Multi-agent system (MAS) is a loosely coupled set of agents which were composed in order to solve problems that monolithic systems (or single agents) can not solve. In order to find the solution of a problem, the agents have to rely on communication, collaboration, negotiation, responsibility delegation and trust Also this subject is discussed in detail in the literature [18].

A. JADE - Java Agent DEvelopment Framework

JADE[1] is a Java-based framework, which allows developers to implement agent-based systems. JADE is designed as a middleware platform that provides the runtime environment for implemented agents.

The framework provides relevant basic classes that are to be extended to adapt it to the specific need of the application in question. For example the base element agent can hold states and knowledge, which can be internally changed by using pluggable behaviors of different base types. These base behaviors in turn, can be composed to complex parallel or sequential behaviors, which can be hierarchically organized.

For communication purposes JADE provides a FIPA-based message mechanism, which relies on an asynchronously working message-box mechanism at the receiver of a message. Message contents can be composed by using simple textual information, complex but serializable content-objects or parts of individual ontologies. The latter can be built in order to provide a domain specific vocabulary for the involved agents [2], [5].

JADE offers several optional services, which can be configured by explicitly addressing their use in the parameter set for the platform start. Here are for instance services available for agent mobility, fault tolerance or the FIPA[2] compliant DirectoryFacilitator. JADE services can be considered as an intermediate layer between platform and the abode of the agent and are especially useful if information have to be shared over the whole platform. For more specific requirements JADE allows to build individual services.

Agents reside in so called agent-containers on the top of the platform. With the initial start of JADE the Main-Container will be started as well. In order to extend the platform to a distributed system, an administrator can start other JADE instances on remote systems. Defining the necessary set of parameters, the remotely started JADE instance will join the platform during runtime by adding a new container to it, which results to the fact that control over the remote system is required.

With respect to our work, we would like to mention here that JADE does not provide any specific support for MABS like for example the definition of a central and synchronized environment model or the measurements of the current system load. To use JADE for the development of such a system means to start from scratch.

B. Modeling activities for a Multi-agent based simulation

The modeling activities that developer has to face in order to provide an MABS to end users are manifold. In order to not overload this subject, the most important aspects are discussed here. For a comprehensive discussion it is referred to the literature [3], [7], [8].

[1] http://jade.tilab.com/ and [2]
[2] http://www.fipa.org/

Figure 1 below provides a rough overview to the elements on which developers have to work on. It is to recognize, that the development has to focus on several main parts. These are, in order to their importance and their influence on a simulation: the environment model, the scheduling of the simulation and the Multi-Agent system itself.

Since agents, by its definition, acting in their environment, one of the first things to be developed for a MABS is the environment model. According to the suggestions of Russel and Norvig [14] environments can be classified into different types considering aspects like accessibility and determinism. Furthermore they can be static or dynamic, discrete or continuous and episodic or non-episodic.

These classifications imply already the presents (or the absence) of time and shows the close relationship between the scheduling of a simulation in conjunction with the environment. Independently of whether this model has to be visualized or not - which additionally increases the developing effort - the type and the data model for it has to be defined first.

Additional modeling effort is also required for the scheduling of the simulation, which can be in principle event based, time-driven or a mixture of them. Furthermore, time dependent simulations can be either continuous, discrete or hybrid. In case of continuous simulations, each time step is very small, which results de facto in a continuous system behavior. Parts of the simulated system can be for example modeled and described through differential equations, which can be used in order to calculate a time dependent system reaction. Discrete simulations are using the time to look for statistically or randomly sized time intervals to cause certain events. These events will determine the (next) state of the system. Additionally, a simulation can be seen as hybrid, if the model has properties which are either continuous or discrete [6], [10], [13].

After defining the environment model and deciding which scheduling strategy has to be used, the Multi-Agent system has to be built. Every autonomous entity, object and relationship between them has to be modeled and, later on, to be implemented. From the MAS-specific perspective this means that:

- agents have to be identified and their types have to be laid down (e. g. deliberative or learning agent as well as predator or pray agent),
- necessary agent behaviors and their compositions have to be defined,
- communication and all other protocols have to be set as well as a specific ontology,
- negotiation and collaboration have to be considered.

One of the last points to be mentioned here is the interaction between environment and agent. Since an agent can act in its environment, the environment, in turn, may react or respectively act on the agent. If this is an inherent

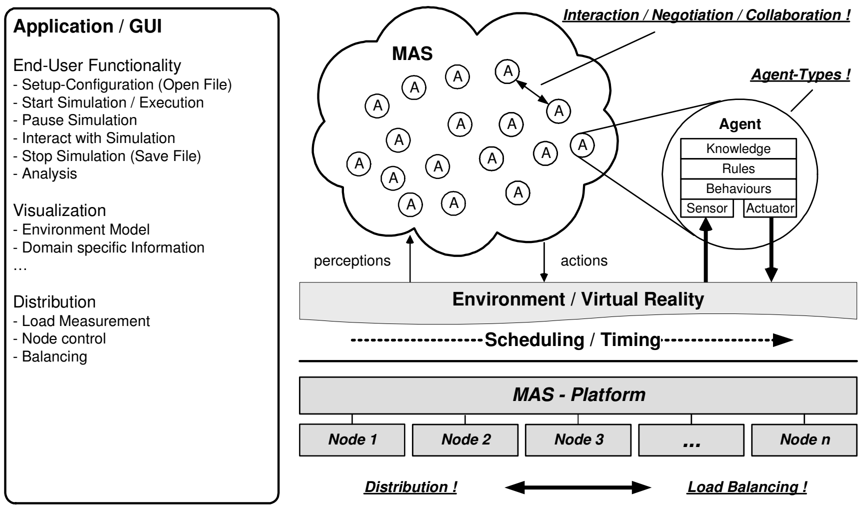


Figure 1: Elements in MABS

point of a simulation, it has to be modeled and implemented as well.

Beside this and considering a simulation, which is usable for experts of different domains, developers have also to focus on graphical user interfaces for the configuration and the interaction with the simulation.

C. Execution of distributed Simulations

The main intention for distributing a simulation is scalability and reduction of local workloads, in order to speed up the simulation, or to allow the simulation to be bigger in terms of the number of calculating nodes and agents. Thus, on the one hand, a simulator needs to be able to spread its simulation to an arbitrary number of computers while, on the other hand, the load over all nodes needs to be balanced in a meaningful way.

This is basically a statement that indicates a further aspect of our tool. The inherent problem of load balancing is widely discussed and not only a subject in the field of simulations. The number of publications is very high so that its treatment requires a separate discussion, which can not be covered in this presentation of our framework.

D. Usability for Developers and end users

Building simulation software that enables end users to work with Multi-agent based simulations, results in dealing with a set of further requirements, which exceeds the needs for “simply” providing a development framework. While a framework and its elements are to be well and transparently documented for developers, the end user application additionally has to be extendable, in order to match the user’s demand. Also it should be easy to understand and use. However, neither usability studies [9] nor agent-human interaction will be discussed in this paper. Nevertheless, it needs to be mentioned, that one motivation for the tool is to be seen in finding an optimum between a support for MAS or MABS developers while, additionally, keeping the predefined user interface as open as possible for all kinds of applications.

Overall we described here the difficulties and the enormous effort, which developer has to face in order to build a MABS from the scratch and providing such simulations to end users.

From this point of view it is certainly not possible, that we present a full description of our framework in this paper. Consequently, in the next section we focus on a general description of our framework, explanations on how the user application can be extended and the use of our simulation service, which is build for a bidirectional agent/environment interaction.

III. AGENT.GUI - INFRASTRUCTURE AND USAGE

Agent.GUI enables developer to create a JADE-based MAS application, which can be directly used by the end users through the prearranged application window of our framework. This end user application is able to control JADE, which means that end users can manage the JADE platform and their agents as well as the developed MABS.

The Agent.GUI end user application is based on Java-Swing. This multi language tool allows the handling of JADE agencies by considering the developed MAS and its resources as an encapsulated system, called project. Such a project can be configured within the application and can afterwards be executed and distributed.

A. General Functionalities for agent projects

Agent.GUI handles a JADE MAS and its resources as a project. Starting from an already fully developed multi-agent system with agents that are able to meet demands placed on them, the application requires only some information relevant for the handling of the MAS and its resources. To get control over these resources, an Agent.GUI project has to be defined and configured initially. In the context of a software lifecycle, we see this as the final step, which has to be done by the developer of the MABS. After this the system should be ready for use by the end user.

Besides assigning a general project name and some textual information the final configurations that a developer has to provide can be done as follows:

- The framework offers the usage of one of two predefined environment model types. This can be, up to now, either a continuous two-dimensional environment or a graph that can act as a central environment model. Both models have already a tailored visual representation which will appear according to its selection. This topic we are planing to discuss in a different presentation.
- External Resources can be picked from the local file system. There it can be chosen between compiled jar-files or complete folders (e.g. bin-Folders of an IDE), which will be added dynamically to the Java-Classpath at runtime.
- Extending and customizing the *PlugIn*-class of our framework, the developer can add her/his own GUI elements to a project (see pargraph *B* of this section).
- From our point of view an ontology can be more than just the central element for the communication of agents. It can also provide comprehensive domain models for a simulation. To use them within the application, the developer can select and add them to the project. Agent.GUI offers a reflective, graphical access to selected subparts of the ontology, so that classes can be initialized and filled with specific values. The ontology is to be created by the BeanGenerator of Protégé3 .
- JADE-Agents can be run by using start arguments. During the instantiation of an agent they can be passed to it as a simple array of objects. Knowing the required object types and their order for a single agent, the developer can assign this information to the project-definition for later use by the end users. The object types can be selected from the underlying ontologies for the project.
- During the JADE-Configuration the needed base services are to be identified and the definition of the port for the JADE middleware is to be done.

After these above mentioned configurations the MABS should be ready to use for end users and the work for developer is done. Since an end user is meant to use the JADE Multi-agent System for her/his own purpose Agent.GUI enables her/him to configure different start setups for the agency. Furthermore, if selected, one of the predefined simulation environments can be configured in order to apply the project agents to different situations and environments.

[3] http://protege.stanford.edu/

B. *Programming interfaces for developer*

Developers can use Agent.GUI and their libraries with their IDE simply by adding the core jar-file to their own project as an external library. From here on, the programming interfaces for customizing a MABS and its visualization can be separated into three types: interfaces of the Agent.GUI framework, external interfaces that are coming "naturally" with the application, because they are integral components of the Agent.GUI project (e. g. the JADE libraries) and external resources, which can be individually added to the MABS during the development phase. Such external resources have to be added by selecting them as a part of the IDE and, then, configuring them as jar-resources in the specific Agent.GUI project.

In order to access the object structure of our framework the singleton class *agentgui.core.application.Application* can be used. Starting from this point, developer can access everything the framework provides. Additionally, the framework provides a translation interface in order to allow developers the use of the language of their choice in the source code. The API as well as some tutorials are available with the framework resources.

In case of a needed customization of the visible program, Agent.GUI allows the extension of the main application window and its elements as well as the extension of the project window for the MABS. For this, our *PlugIn*-mechanism and the extendable *PlugIn*-class were designed. The *PlugIn*-class provides access to menus, the main toolbar and allows adding further tabs to the project window. If the customized *PlugIn*-class is configured in the project, the tailored elements will be automatically added to the visual program. The following screenshot in Figure 2 shows the example of an extended application window and some additional tabs which were added to the project window.

Beside this developers can react on application events. For this the project data model and its visual representation was designed by using the common MVC pattern. Individual tabs for the project window as well as the configured *PlugIn* classes will be informed about changes in the project settings or if a simulation is to be started.

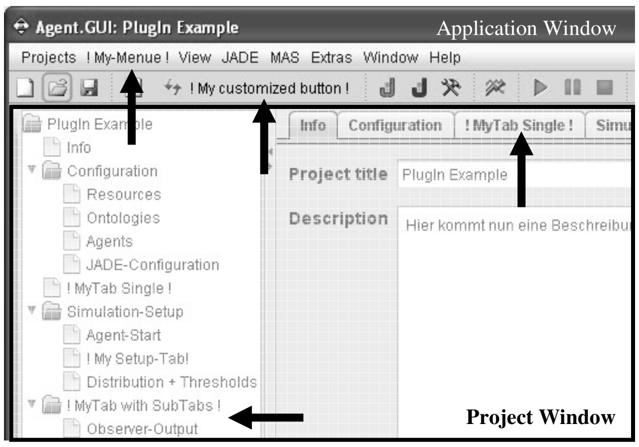


Figure 2: Extended GUI for MABS

C. Simulation Service

One motivation to build our Simulations Service can be seen in the handling of agencies, with a large number of agents. Such scenarios can be found for example in the current research topic of smart grids and intelligent power supply. Here a big number of participants can be found due household, industrial consumer and power producers. They all rely on an interconnected network, which can be seen as their environment. Analyzing the connection between the participants, seen as agents, and their physical connection to the grid, it is obvious that not only the agents are working in their environment, by consuming electrical power. In turn, the grid delivers the electrical power or, if the supply doesn't meet the user demand, the generation can collapse. For an agent/environment interaction this leads immediately to a situation where an environment acts on agents too, which shows this bidirectional relationship. Testing such interaction with a large number of agents by using ACL messages, we found out that the JADE platform became overloaded, so that we had to find a different solution (see also section IV).

Based on an extended JADE base service we designed our Simulation Service for a bidirectional interaction between agent and environment, taking into account that scheduling strategies can differ, depending on the kind of simulation. Therefore we equipped the simulation service, which is particularly used in order to transport the environment model information to the agents, with a set of methods, which can be individually used depending on the specific simulation schedule.

In general we assume that at least one entity, in our case an agent, has to manage the information about the environment. This agent is to be assumed as the *SimulationMangerAgent*. Other agents, which are acting in this environment, our so called *SimulationAgent*'s, have to be connected to the environment due our *SimulationSevice* and its inherent sensor/actuator functionalities. Figure 2 shows this relationship and the herein used classes.

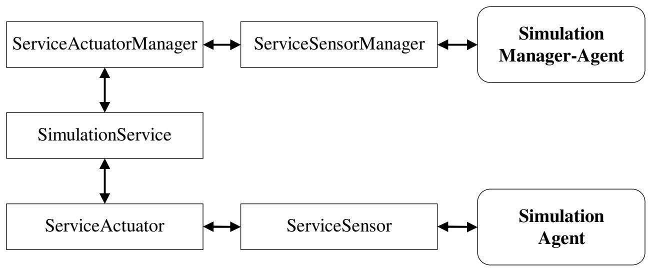


Figure 3: Classes used for the agent/environment interaction

Every agent is equipped with a so called *ServiceSensor*. This applies for the *SimulationMangerAgent* as well as for the other agents. Using the *SimulationService*, this sensor can be connected with the *ServiceActuator* on site; this is in fact the *ServiceSlice* of the JADE agent container, in which the agent is currently located. Colloquially as a metaphor, one could consider this as the insertion of a plug into an electrical outlet.

In case that a discrete simulation steps forward or an event based notification for an agent occurs, an actuator can transport a new environment model or notifications to the connected sensor of the agent. In order to prevent the loss of the autonomy of the agents, we designed this stimulus in an asynchronous way.

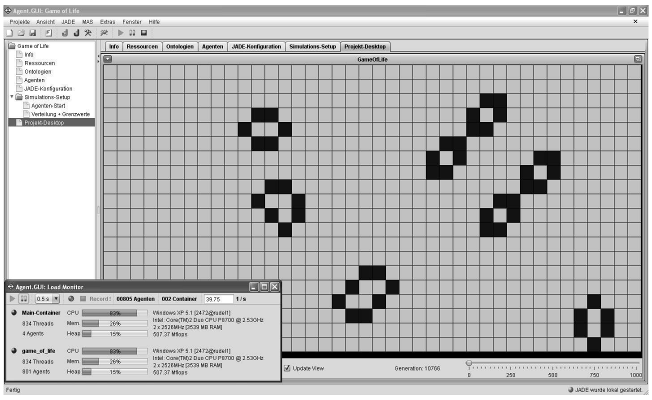
There are several opportunities for the manager agent in order to schedule or organize the simulation process. There are for example methods available that are allowing to run the system in an episodic way (see example in section IV), while other methods allow to address single notifications to agents, which is relevant for event based simulations. Furthermore the *SimulationService* provides synchronized time to all (distributed) containers on the JADE platform; this can be used for timed simulations. The kind of scheduling is therefore still subject of the agent design and is not limited through our framework.

As a generalized environment Agent.GUI provides a class structure that consists basically of three sub parts, which allows describing the state of the environment. They are: (i) a *timeModel*, (ii) a domain specific *abstractEnvironment* and (iii) a *displayEnvironment*. In this a time model can be a simple counter, which steps forward with every state of the environment model (called *TimeModelStroke*) or it can be a concrete time, which can be changed for every state of the environment. The latter two attributes of the model are simple *Object* types, which allow applying a variety of structures to them. Hereby the abstract model should be used for general structures as they can be defined by ontologies, while the latter attribute should contain model information, which can be also displayed. Furthermore, using the so called *TransactionMap*, the simulation service is able to manage different states of the simulation over the simulated time.

IV. Application: Testing the simulation service

Since Agent.GUI is build on top of JADE its overall performance depends mainly on the JADE implementation. Nevertheless, a few optimizations were conducted because the simulation service relies on method execution instead of sending ACL messages for the agent/environment interaction. This let to a significant increase in the simulation speed, which was validated through the following small experiment.

Two MAS for a Game of Life (GoL) were implemented that consists of simple cellular automata (Figure 4). In order to evaluate the efficiency of our simulation service, one implementation used ACL messages for the agent/environment interaction; the other one used our *SimulationsService* introduced in the previous section.



0.5 s | 00805 Agenten | 002 Container | 39.75 1 / s

Main-Container: 834 Threads, 4 Agents; CPU 83%, Mem. 26%, Heap 15%; Windows XP 5.1 [2472@rudel1], Intel: Core(TM)2 Duo CPU P8700 @ 2.53GHz, 2 x 2526MHz [3539 MB RAM], 507.37 Mflops

game_of_life: 834 Threads, 801 Agents; CPU 83%, Mem. 26%, Heap 15%; Windows XP 5.1 [2472@rudel1], Intel: Core(TM)2 Duo CPU P8700 @ 2.53GHz, 2 x 2526MHz [3539 MB RAM], 507.37 Mflops

Generation: 10766

Figure 4: Game of Life in the Agent.GUI application window

In both cases a single agent represented one field while one agent was the manager of the environment, which in turn consisted of all area-agents of the playing field. At initializing of the GoL, the simulation manager created the visual representation first, which allows user to define the initial game situation, before actually starting the game. Executing the ‘simulation’ by the manager agent then starts the cyclic simulation of the GoL as shown in the next sketch below.

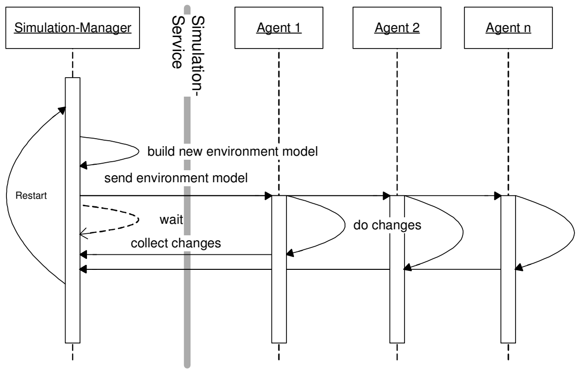


Figure 5: Simulation Cycle for the Game of Life

Collecting all changes from the visual representation and take them over into the private environment model of the ‘Simulation-Manager’, this model was send as new environment state to every agent. Knowing the name of the agents in the neighborhood, the field agents were able to get these states and calculate their own next state. This new state was send back to the manager agent, who builds up the new environment model and started the next generation of the Game of Life again. The grey line in Figure 5 indicates the use of the *SimulationService* instead of ACL messages in our benchmark study.

For this comparison the following system was used:

CPU:	2 x 2527MHz (Intel® Core™ 2 Duo CPU P8700)
RAM:	3539 MB RAM
OS:	Windows XP 5.1
Java:	jdk1.6_014

Executing between 20 up to 3000 field-agents in a single JADE-container, 5000 simulation cycles (generations) were done according to the sequence showed above. The time for the simulation cycles was measured and smoothed within the *SimulationManager*.

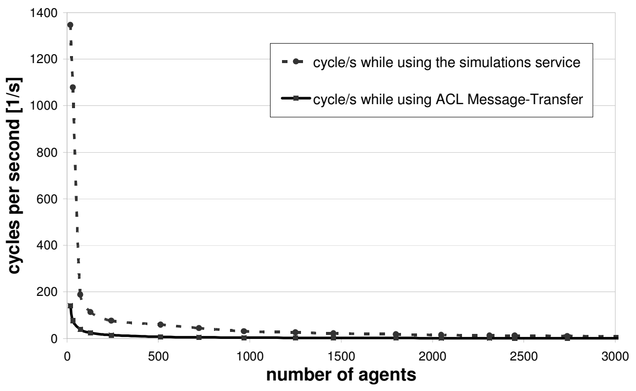


Figure 6: Simulation cycles and number of agents for agent/environment interaction

The measured increase for the agent/environment interaction was 9 times in average. Figure 6 above shows this comparison. Further tests showed that using ACL messages for the agent environment interaction leads more quickly to an overloaded system.

V. RELATED WORK

To the best of our knowledge there are only very few extensions or tools with similar objectives than Agent.GUI for the JADE platform. We distinguish between those tools that are also based on JADE as agent platform and those that are providing their own agent concepts, but similar core functionality. To the first category belongs SIMJADE [10] or DisSimJADE [4].

SIMJADE is an extended BaseService for the JADE platform. It supports distributed simulations by providing an optimistic Time Warp based synchronization scheme. Furthermore, SIMJADE comes with some basic classes that can be extended to cater for individual needs. This is similar to our time and environment synchronization concepts. However, in contrast to SIMJADE, we have implemented no concrete synchronization process but open interfaces for different schedule mechanisms. Topics like load balancing, automated distribution of simulations, visualization techniques or usability for end users were not addressed with SIMJADE.

DisSimJADE was introduced by Gianni et al in 2009. This simulation framework enables the incorporation of distributed simulation facilities into agent-based systems. It was build on top of JADE. By using the general purpose architecture HLA (High Level Architecture), this framework enables the interaction (distribution of data and the synchronization of actions) between different simulation systems that follow the IEEE standard[4]. In case of a distributed simulation DisSimJADE does not use the interfaces and infrastructure for distribution which JADE already provides. Instead, the standardized HLA-based communication structure is used for the interaction between distributed agencies. This was basically motivated by the interoperability to other simulation systems with similar interfaces.

The comparison of Agent.GUI to these two approaches for MABS shows that the idea of using JADE for simulation is not new. However, until now it was never implemented consistently in a comprehensive framework.

The category of general framework and end user aspects contains a big number of available tools for agent-based systems. Here we concentrate on programs, which consider an environment as an important factor for a MABS. Especially for such systems, Arunachalam et al [1] pointed out four main criteria for a comparison of such tools. These are in short:

- The design criteria, specified through the definition of an environment, the environment distribution and the coupling of agent and environment.
- For the model specification they rated the ease of specifying the environment, what features a system offers to define the environment and what knowledge an agent can have about this environment.
- With respect to the model execution the quality of visualization and the possibility of property modifications during runtime were considered to be most important.
- Finally, the quality of the available documentation was investigated.

On the basis of these categories *NetLogo*, *MASON*, *Ascape*, *RePast Simphony* (*RePastS*) and *DIVas* were examined. For a more extensive survey on tools for "Agent Based Modeling and Simulation Tools" (ABMS), we refer to the website of Ron Allan[5].

The direct comparison to such frameworks and to the above mentioned JADE extensions discloses that there is still a lot of space for improvements for us. But in relation to the above mentioned criteria's it is our opinion, that Agent.GUI can be seen as a competitive framework with some unique features. Table I, on the right, shows the result of our own assessment in comparison to the mentioned frameworks discussed in [1].

Since Agent.GUI was designed as a general purpose tool for MABS and because those kinds of developments are highly domain specific tasks, some of our ratings can not be clearly applied in the sense of the tool comparison of [1]. This applies for example at the agent and environment coupling at the design criteria's for environments. While our framework offers a generalized environment model (A.1.) that can be used in a distributed manner for more or less any kind of environment model, by using the *SimulationService* (A.2.), the coupling given through the bidirectional sensor/actuator relationship implies to be a very high one (A.3.). In the sense of the tool comparison this seems to result to the lowest rating possible there. As a developer, however, is free in the decision to use our service or not, we argue that this is essentially a question of the applied domain, the chosen MAS architecture and last but not least the desired agent/environment interaction. For this, with Agent.GUI, a MABS can be designed with a very low coupling between agent and environment as well, which would finally result to a rating of "Very High".

TABLE I
SELF-RATING OF THE AGENT.GUI FRAMEWORK

Criteria	Rating
A. Design Criteria	
1. Environment structured complexity	Very High
2. Environment distribution	Very High
3. Agent and Environment coupling	-
B. Model Specification Criteria	(P2D / NM [6])
1. Specification features offered	High / Very High
2. Programming skill of end user	Low / Low
3. Environment knowledge in agents	High / High
C. Model Execution	
1. Quality of visualization	High / Very High
2. Simulation view	Low / Low
3. Model property modification	Medium / High
D. Documentation	
1. Quality of documentation	Medium
2. Effectiveness of the documentation	-

With our framework the responsibility for providing tools for the definition of an environment is basically up to the developer. Nevertheless, since Agent.GUI provides two predefined environment model types (a continuous 2D model and a network model consisting of nodes and arcs), we rated criteria B.2. for both models in all conscience. As it is one of our main intentions that developers provide end-user applications, the aspect of programming skills for end users (not developers) is rated to "Low". In relation to the above mentioned point of the agent/environment coupling, the aspect of the agent's knowledge about the environment was rated to "High". This basically depends also on our predefined environment model types and will differ depending on the concrete application.

Regarding the model execution in C., our framework uses basically the same classes and types for the visualization of a simulation as they were used for the definition of an

[4] IEEE Standard 1516: Standard for Modeling and Simulation High Level Architecture

[5] http://www.grids.ac.uk/Complex/ABMS/ABMS.html

[6] P2D: Physical 2D environment / NM: Network Model

environment; for this standardized interfaces are used. This is why we have here the same evaluation as in B.1 which relies on the connection with our predefined environment types. A 3D or toroidal simulation view, as it was asked for in the comparison of [1] (C.2.), can not be provided by our framework until now. Our rating in relation to the model modification during runtime has also to be seen in connection to our two environment models. Additional it should be mentioned here, that for the Game of Life agency for example this aspect must be rated with "Very High", as a modification at runtime was even desirable, which shows again the strong dependencies to the domain.

The aspect of documentation under D. is one of the next important tasks for the development of our framework, as indicated by the relatively low rating (D.1.). Since outside of our group so far only a very few developers have worked with Agent.GUI, it is not possible to evaluate the effectiveness of the documentation at this time.

Due to the fact that Agent.GUI is based on JADE, our framework is compliant to FIPA standards too. The above mentioned design criteria is extensively addressed; especially in an active and bidirectional relationship between agent and environment. Regardless of our two predefined environment model types, we see a big advantage in the fact that Agent.GUI provides open interfaces for any kind of environment model and simulation scheduling, which allows a nearly unrestricted development (e.g. for 3D-models etc). Another benefit comes with the functionalities regarding the customizable load balancing approach, which we could not find in any other tool or framework for MABS.

VI. Conclusion

In this paper we introduced a framework for Multi-agent based simulations called Agent.GUI that is built on top of the JADE platform. Based on our frameworks base-GUI it allows the programmer to realize a domain specific end user application for Multi-Agent based simulations. For this purpose Agent.GUI provides open and adaptive interfaces.

An example scenario that illustrates the efficiency of our bidirectional simulation service in comparison to the use of ACL messages for the agent/environment interaction was shown and discussed.

Beside an in-depth comparison study of our framework to other agent frameworks, it is already planned to present further aspects of our framework in the near future. Here we would like to discuss the load balancing abilities of Agent.GUI for distributed large scale simulations. Furthermore the usage of predefined environments for smart energy networks will be shown soon. Currently, work is in progress to use Agent.GUI for simulations of intelligent and self configuring high pressure gas grids.

References

[1] S. Arunachalam, Rym Zalila-Wenkstern, and Renee Steiner. Environment mediated multi agent simulation tools. In *SASO Workshops*, pages 43–48. IEEE Computer Society, 2008.

[2] Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, April 2007.

[3] Jacques Ferber Fabien Michel and Alexis Drogoul. Multi-agent systems and simulation: A survey from the agent community's perspective. 2009.

[4] Daniele Gianni, Andrea D'Ambrogio, and Giuseppe Iazeolla. Dissimjade: a framework for the development of agent-based distributed simulation systems. In Olivier Dalle, Gabriel A. Wainer, L. Felipe Perrone, and Giovanni Stea, editors, *SimuTools*, page 21. ICST, 2009.

[5] Thomas R. Gruber. A translation approach to portable ontology specifications. *KNOWLEDGE ACQUISITION*, 5:199–220, 1993.

[6] Alexander Helleboogh, Tom Holvoet, Danny Weyns, and Yolande Berbers. Extending time management support for multi-agent systems. In Paul Davidsson, Brian Logan, and Keiki Takadama, editors, *MABS*, volume 3415 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2004.

[7] B. Logan and G. Theodoropoulos. The distributed simulation of multiagent systems. *Proceedings of the IEEE*, 89(2):174 –185, February 2001.

[8] Charles M. Macal and Michael J. North. Tutorial on agent-based modelling and simulation. *J. Simulation*, 4(3):151–162, 2010.

[9] Jakob Nielsen. Usability engineering. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 1440–1460. CRC Press, 1997.

[10] Dirk Pawlaszczyk and Ingo Timm. A hybrid time management approach to agent-based simulation. In Christian Freksa, Michael Kohlhase, and Kerstin Schill, editors, *KI 2006: Advances in Artificial Intelligence*, volume 4314 of *Lecture Notes in Computer Science*, pages 374–388. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-69912-5_28.

[11] Evangelos Pournaras, Martijn Warnier, and Frances M. T. Brazier. A distributed agent-based approach to stabilization of global resource utilization. In Leonard Barolli, Fatos Xhafa, and Hui-Huang Hsu, editors, *CISIS*, pages 185–192. IEEE Computer Society, 2009.

[12] Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. In Victor R. Lesser and Les Gasser, editors, *ICMAS*, pages 312–319. The MIT Press, 1995.

[13] Reuven Y. Rubinstein and Benjamin Melamed. *Modern Simulation and Modeling*. Wiley & Son, 1998.

[14] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[15] J. Vázquez-Salceda, W. W. Vasconcelos, J. Padget, F. Dignum, S. Clarke, and M. Palau Roig. Alive: an agent-based framework for dynamic and robust service-oriented applications. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1637–1638, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[16] Junwei Wu and Xiaojun Cao. Intelligent traffic simulation grid based on the hla and jade. In Wu Zhang, Zhangxin Chen, Craig C. Douglas, and Weiqin Tong, editors, *HPCA (China)*, volume 5938 of *Lecture Notes in Computer Science*, pages 456–464. Springer, 2009.

[17] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 1994. Submitted to Revised.

[18] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley & Sons, 2nd edition, July 2009.