JREP: Extending Repast Simphony for JADE Agent Behavior Components

Jana Görmer, Gianina Homoceanu, Christopher Mumme, Michaela Huhn and Jörg P. Müller*
Niedersächsische Technische Hochschule
Braunschweig, Clausthal-Zellerfeld, Hannover, Germany
Email: {Jana.Goermer|Christopher.Mumme|Michaela.Huhn|Joerg.Mueller}@tu-clausthal.de, G.Homoceanu@iti.cs.tu-bs.de

***Abstract*—When modeling and simulating agent communities, one is usually interested in the macro effects that result at the system level from the interaction of individuals. However, when developing a scenario, designers need to specify the individual agents' behavior, their communication and organizational structure at the micro-level. Existing agent frameworks focus on either the macro or the micro perspective: Repast Simphony is an example of the former, JADE of the latter. We propose JREP, a novel integration of JADE and Repast Simphony that efficiently combines the macro and micro perspective with an interaction layer. It allows to see not only the overall system behavior, but also the individual together with its interests, goals and the communication to others for local coordination and cooperation. Scheduling of the agents, (time) synchronization and the registration of new agents with the environment has been solved. An agent-based airport scenario is introduced a proof of concept for modeling and simulating in the JREP platform; for validating scalability and performance properties, a simple coin flip scenario is described.**

***Keywords*-agent-based simulation, agent applications, Repast, JADE**

I. Introduction

Multi-agent systems (MAS) [10] are a promising paradigm for constructing complex software-intensive systems providing multiple methods for modeling and simulating. However, when developing a multi-agent system, designers need to specify the organizational structure, the individual agents' behavior, and their communication. Existing agent frameworks focus on either the macro for manipulating the environment or for instance on simulating swarms [6] or on the micro perspective to show agent's behavior. Defining the different perspectives 1) on the *macro level* we understand the overall system level as a global view or "aggregated bottom-up view" evolved from a complex system which typically exhibits hierarchical self-organization under selective pressures, 2) for the *micro level* its agent behaviors, and 3) the linking component to join both is the *interaction level* which mainly is done through communication. Thus, our goal is to provide an efficient and usable simulation platform for complex systems supporting the macro and micro perspective linked by interaction. We aim at analysing the overall system behavior, as well as the individual with its goals and communication to others for local coordination and cooperation. Technically, we combine Repast Simphony (Repast S) [9] and JADE [2]. A close approach was proposed in [15] for enterprise value-adding networks, but their solution has some shortcomings: it only supports an older version of Repast, the generality is limited by the focus on Supply Chain Performance Analysis, and performance is restricted due to an inefficient polling strategy. In contrast to [15], we have chosen to use references for solving the polling problems. Repast S gives the information about the new *Tick* to each connected agent by establishing bidirectional communication. Agents are informed through a scheduler about the new system time. Therefore, JADE agents can act directly on the Repast S platform which could be generalized to an open platform (any agents) through a wrapper class.

The paper is structured as follows: in Section II we outline the requirements for simulating a smart airport as one example of a complex MAS; in Section III we give an argumentation for using the platforms Repast S for macro level and JADE the micro level. Section IV describes our JREP solution including architecture, scheduling and registering. Section V we present our JRep tool on a coin flip for evaluating the performance and on the smart airport showing details on the macro/micro/interaction level integration. Finally, Section VI summarizes and concludes the paper with future work.

II. Requirements for simulating a smart airport

We focus on complex MAS consisting of numerous heterogeneous agents and their interaction, an environment with different subsystems and control structures to handle the complexity of the system [5]. A joint demonstrator, currently built on the JREP platform, will demonstrate the results of our research, in a scenario on *autonomous transportation services* at an airport's departure area [4]. Different requirements of the environment are considered and the agents with their spheres of influence, in respect to the macro perspective, i.e. the global system level, the micro

This work was funded by the NTH Focused School for IT Ecosystems (www.it-ecosystems.org). NTH (Niedersächsische Technische Hochschule) is a joint university of Technische Universität Braunschweig, Technische Universität Clausthal, and Leibniz Universität Hannover.

We also thank Aret Duraslan for fruitful discussions regarding JRep.

(objects' behavior) view and the interaction (the intermediate layer between the micro-macro view) are relevant.

In the following we describe our example scenario and the resulting requirements for a simulation infrastructure:

We consider a smart airport where autonomous agents transport passengers between stations (entrances, check-in counters, gates, and plane parking positions), referred in the rest of the paper as transport agents (TAs). The TAs are equipped with batteries, which have to be recharged after some time at a charging station. The charging stations are designed as agents with the assignment to handle traffic jams that may occur, i.e. when a high amount of TAs want to recharge simultaneously. The stations are connected by two-lane roads which we define to be a subsystem. Thus, in these subsystems agents have different rights and influences.

In order to model and simulate the airport we propose three levels and further we will describe the requirements for the development frameworks on each level:

Macro level technologies and techniques are related to the system as a whole (i.e. structure). At the macro level, the airport is viewed as a whole, to see the agents move within different regions. Analysis tools, good performance in terms of execution speed and support for the global decision making processes should be available. For combining different sub-views a step-by-step simulation with predefined function is desired. Monitoring functions to see the simulation progress, displaying functions to show different data on predefined performance criteria are desired for the designer or manager of the system add-ons for the global system view. The macro level represents the complex system with many interdependent individual actors working like a "state of insects". Agents have only a partial view of the system and no central controlling agent [10], the macro level (complex behavior of the entire system) is built based on the individual strategies and the interaction (i.e. cooperation) with other individual agents. We consider a bottom-up/ decentralized approach to form the complex airport system.

Micro level (agent level) technologies and techniques are concerned only with individual agents (i.e. procedures for agent reasoning, learning). Due to the diversity of agents in an airport, it should be possible to implement different agents from simple reactive robots to intelligent ones, which are able to make their own decisions based on their view on the environment and goals. The agents need to perceive the environment, to act accordingly and autonomous. Since agents can be heterogeneous, one can implement its own agent architecture in the micro level, like we did in [5].

Interaction level technologies and techniques concern the communication between agents (i.e. communication languages, interaction protocols and resource allocation mechanism). Macroscopic behavior of the total system is constructed through local interactions or microscopic behavior of the agents. Therefore, a major role is played by the interaction between agents, because the global system state emerges as a side effect of the interactions of subsystems. This is primarily accomplished through communication, the essential functionality underlying an agent's social ability [7]. Through different competences and abilities, agents need to coordinate themselves for achieving their goals, also in an airport. Communication (i.e. exchange information) is necessary, but due to the agents' heterogeneity they need to understand a common language.

III. Rationale for choosing a platform

There are several toolkits available for modeling and simulating complex systems as JADE, Swarm, NetLogo, Mason, Repast etc. and different surveys over these toolkits were given in the past few years as [8]. Considering the requirements of our approach stated in the previous section, we have chosen Repast Simphony and JADE:

Repast Simphony is pure Java extended by the Repast (open-source set of tools, originally based on Swarm [6]) portfolio for developing agent-based models. One main feature of Repast S is the run-time GUI that allows the user to control the simulation during run-time by pausing, resetting and restarting different models. For example, the user can easily compare start configuration of different models with their behavior during run-time. Repast S provides also statistical functions such as graphs which are helpful for simulation monitoring and for real-time monitoring. Through defined variables for monitoring, graphs can be designed during development time. During run-time these graphs are displayed with actual variable's values and its history can be recorded. Repast S supports the usage of external statistical programs, like R[1], for monitoring the simulation at run-time. Thus, these features apply the *model in the loop* idea [11], so that the construction of a complex system, such as our airport, can be supported.

One primary purpose of Repast S is to control which specific actions are executed when, in simulated time. For this, Repast S provides explicit methods for scheduling actions.

Furthermore, in [13] a comparison of Repast S with other software platforms (Swarm, NetLego, MASON) is provided based on several criteria as model structure, scheduling or execution speed. The authors concluded that Repast S is a good Java simulation platform which can guarantee good performance on execution speed, but it has received poor feedback for modeling agents and their structure.

For our purpose Repast S covers the *macro level*. It is used for simulating the environment of our airport and for visualizing and monitoring the emergence of the macro behavior from collective behavior of different agents composing it.

JADE has a distributed system topology with peer-to-peer networking, and software component architecture with agent paradigm. The network topology affects how various

[1] http://www.r-project.org/

components are linked together, whereas the component architecture specifies the component's expectations from one another. JADE provides an agent programming framework, while in Repast S any Java class without an agent-based structure, can be defined as agent [13]. All tasks of agents are modeled as *behavior* objects. The behavior implements either the entire task or sub-tasks for implementing more complex ones. Thus JADE has no specific agent architecture. One can argue that Jadex [12] could be also a solution, but it limits us to use its BDI agent architecture. Another advantage of JADE, it includes FIPA(-ACL) and interaction protocols where FIPA agent communication specifications deal with Agent Communication Language (ACL) messages.

Considering the previous stated advantages and the conclusion drawn from several evaluation reports that JADE is highly efficient in terms of performance on the agent message transport layer [14], the internal database access and message exchange capabilities [3], made us choose JADE as a modeling platform. Mainly, JADE considers the micro level, used for defining and implementing our own agent architectures (TA, charging station, etc.).

The *Interaction level* plays a major role in our approach because it represents the intermediate layer that makes the connection between the macro and the micro level. As stated in Sec. II this is realized through interaction between agents. Here one of the JADE advantages is the support for FIPA ACL protocols. In ACL messages interaction protocols are exchanged, which are based on the speech act theory. This includes communicative acts and content language representations. JADE agents communicate remote and work decentralized with each other. The combination of the both platforms makes use of complement features provided by each platform: while Repast S lacks in support for providing the agent structure, the Jade framework provides for the complete modeling of the agent behaviour[15]. At the same time Repast S offers step-by-step simulation, planning schedules, visualization and monitoring of the global systems (the macro view) which are not supported by JADE.

IV. Combining JADE and Repast S: JRep

In this section we cover the JREP architecture by the following key issues:

Macro level: For the overall system we present the technical implementation of our *JREP architecture* and the *scheduling process* of the agents during runtime.

Micro level: For a single agent we created our architecture in JADE and we explain the *registration of an agent* to the Repast S environment.

Interaction level: The interaction between agents is provided by FIPA ACL.

A. JREP Architecture

As stated in Section III we use JADE and Repast S for a common JREP platform. One goal is that JADE agents can act in Repast S. Thus, we assume three minimum requirements needed by agents for acting in a simulation environment (cf. black box agent model by [7]):

1) Perception (Input): Agents must be able to sense the environment. Thus, they must perform "Perception" and thereby receive information regarding the environment.
2) Agents must be able to make plans and execute them, in order to achieve goals (e.g. path finding algorithms in the airport).
3) Action (Output): Agents must perform actions and they also have to be informed about the success or failure by performing that action.

In consequence, agents have to access the simulation environment. In addition, the simulation environment must inform the agents about the current simulation time.

Considering the previous statements, we developed an architecture of the platform (see Fig. 1) with the following components:

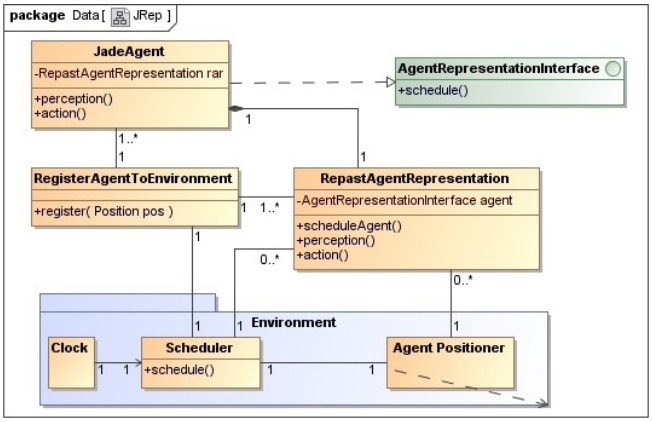

Figure 1. Class diagram of the JRep approach

The main idea to integrate JADEAgents into Repast S is to use an agent wrapper class that is implemented in Repast S:

RepastAgentRepresentation: Each agent has exactly one representation within Repast S, which is responsible for the execution of *Perception* and *Action*. Therefore, the class *RepastAgentRepresentation* possesses the methods perception() and action() (cf. Fig. 1). Furthermore the agents are informed of each new *Tick* by the *RepastAgentRepresentation*.

JADEAgent: The JADE agents can be connected to Repast S (and/or the appropriate *RepastAgentRepresentation*) directly by references. In order to avoid the problem of *Polling* occuring in [15], the communication between agents and the Repast S simulation environment must be possible in both directions. First agents are informed about the new system time of Repast S through the *scheduleAgent()* method of the *RepastAgentRepresentation* class and then

they can act directly on the Repast S platform (cf. Fig. 1). For this, it is necessary that the agents implement the Interface *AgentRepresentationInterface* with the *schedule()* method. Thus, a *RepastAgentRepresentation* object receives the reference from the agent (that is of the type *AgentRepresentationInterface* at the same time). The agents instantiate a *RepastAgentRepresentation* object independently when they connect to the environment and also have the reference to the *RepastAgentRepresentation* object (cf. Section IV-C). In a similar way, one can connect also Repast S agents to the simulation environment by using references.

Repast S already possesses an internal clock, where each *Tick* represents a simulation step. However, in JADE there is no given internal clock. If a JADE agent acts in a Repast S environment, it must be synchronized with Repast S's internal clock. Thus, our JREP architecture provides the concepts of *Clock* and *Scheduler*: The Repast S *Clock* gives the cycle of the simulation time and informs the *Scheduler* component. The *Scheduler* conducts the execution of actions by Repast S connected agents. In each *Tick* the *RepastAgentRepresentation* is informed about executed actions.

Further we will describe the scheduling process, providing first the necessary steps for this (see Fig. 2). The *Tick* is triggered by the system clock (1.) and the scheduler successively contacts all connected agents in a loop. Then the *scheduleAgent()* method (2.) of the *RepastAgentRepresentation* class is called. At the same time a flag is set in this class (3.), to denote that the agent was informed about the new *Tick*. The flag is released after the agent takes an action (a movement to the north, south, east, or west, or a delay) (16.). This is necessary to guarantee that the scheduler allows only one agent to perform an action (in order to keep the agents synchronous to simulation). After the scheduler informs the agent (2.) it asks repeatedly to sets its flag (4.) and informs the next agent only if the flag is approved. The JADE agent performs perception (6.) only after receiving the information about the new *Tick* (5.) from its *RepastAgentRepresentation* object. This is achieved in the environment by the *RepastAgentRepresentation* object (7.) and it gives the information about the environment to the agent (8. and 9.). Now the agent can react and can make an action in the environment (11.). Note that we also define "waiting" as an action. Then the *RepastAgentRepresentation* object tries to execute the movement (11.) by activating the agent positioner (12). The agent receives the outcome, i.e. whether the movement was successful (13. - 15.).

B. Open platform

In our platform we implemented the agents as JADEAgents, but it is also possible to integrate other agent programming (AP) platforms (i.e. Jason, Jack) to JRep. A first approach to realize this is to enable Repast S to have a reference to the JADEAgent and vice versa. This is achieved

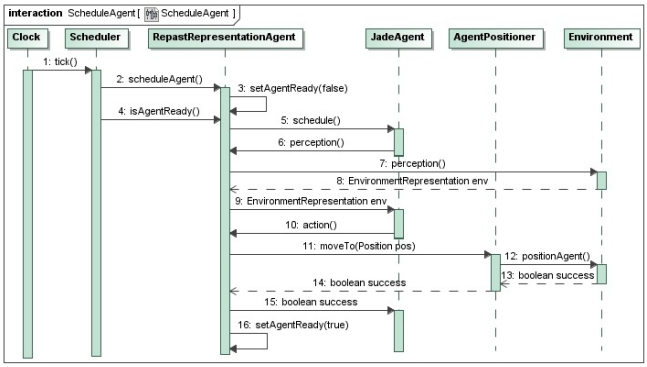

Figure 2. Scheduling agents in JRep

by representing an agent in JRep by a *RepastAgentRepresentation* wrapper class and by the fact that the JADEAgents implement the *AgentRepresentationInterface* Interface. A more general view of the approach reveals that any Java agent can implement the *AgentRepresentationInterface* Interface. Then, both Repast S and the Java agents can call methods using references to each other, because both are Java-based.

Another approach for integrating other AP platforms with Repast S is to use Remote Procedure Calls (RPCs). In this case the *RepastAgentRepresentation* class must allow an agent to call appropriate RPC methods from remote (i.e. the methods must be available). RPCs connect any AP platform independent from its programming languages to Repast S. For instance one can implement XML RPCs for information exchange (i.e. perception and action) between Repast S and an agent. However, this approach can slow down the overall system performance, because the communication between an agent and the simulation environment can take more time than a simple call by reference. For our purpose we have implemented JRep with JADEAgents that are connected to Repast S by references.

C. Agent perspective

While in Section IV-A we presented how to combine a JADEAgent with Repast S from an overall perspective, now we focus on the single agent perspective. First we describe the connecting process to the simulation platform:

A JADEAgent can be added to the simulation environment at the simulation run-time as illustrated in Fig. 3. The class *RegisterAgentToEnvironment* is implemented in Repast S which makes the connection between Repast S and JADE. It ensures that new agents are positioned initially. After instantiation of an agent, it must receive a reference to the appropriate *RegisterAgentToEnvironment* object (e.g., by delivery in the constructor or by web services). Now, the method *registerAgent(Position pos)* in the *RegisterAgentToEnvironment* class (1.) is called. The agent gives the preferred coordinates to start in the environment. Then the *RegisterAgentToEnvironment* generates a *RepastAgentRepre-*

sentation object for that agent (2.). This places the agent in the desired position if that space is available (e.g. not occupied by another agent) (3.). If the agent is placed, the reference of the *RepastAgentRepresentation* object will be handed over (4., 5. and 6.).

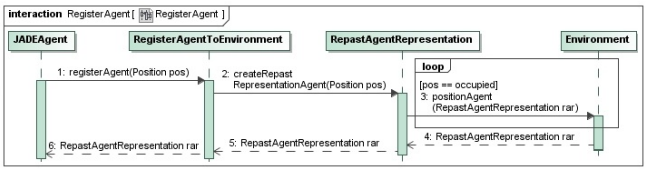

Figure 3. Register a new agent to the simulation platform

We have implemented our own agent architecture uniformly used in our airport scenario [5], but the JRep platform obviously allows for the implementation of different kinds of agents: i.e. simple reactive agents and agents with complex architectures. This is because JADE is used, which comes without agent architecture specification, but provides some features that simplify the implementation of agents.

D. Agent interaction

In MAS, agents interact with each other either through the communication or by reacting to the behavior of the other agents. We have chosen the first approach. However, in order to communicate agents must be able to understand each other using a common language. In our approach we have used the first approach, making use of the FIPA ACL standard, which provides a common language for the agents to understand each other.

V. Case Studies

Further, we present two case studies to demonstrate the capabilities of the JREP platform.

A. Coin Flip Scenario

In the coin flip scenario, autonomous agents work in two teams which can communicate with each other. Each team is allocated one side of a coin. The task of the teams becomes flipping the coins such that their allocated side of the coin shows on top. On the macro level the aim of this scenario is to show that for certain parameters the count of both sides of the coins is in equilibrium.

In the implementation of the scenario the agents are located in a grid based environment. The teams are of equal size. The coins are represented by blocks (colored red or blue) which can be flipped by the agents. The scenario has been first tested for 100 agents working on an 30 by 30 grid and 200 blocks, illustrated in Figure 4. The micro behavior of an agent is to look for the nearest block not being of its team color and move (by A* search algorithm) to the block to flip it. The diagram at the bottom of Fig. 4 is a feature of Repast S and shows the distribution of the red and blue

Table I
Scalability and performance of JRep

Configuration (#Grid;#Agents;#Blocks)	Ticks in 10 sec.	Memory Footprint
500x500;100;100	725	2 MB
500x500;500;500	553	14 MB
500x500;1000;1000	335	37 MB
500x500;5000;5000	261	169 MB
500x500;10000;10000	148	332 MB

blocks's count. On the right side of the figure the JADE Remote Management GUI is displayed. It shows the agents that are implemented in JADE and connected to Repast S.

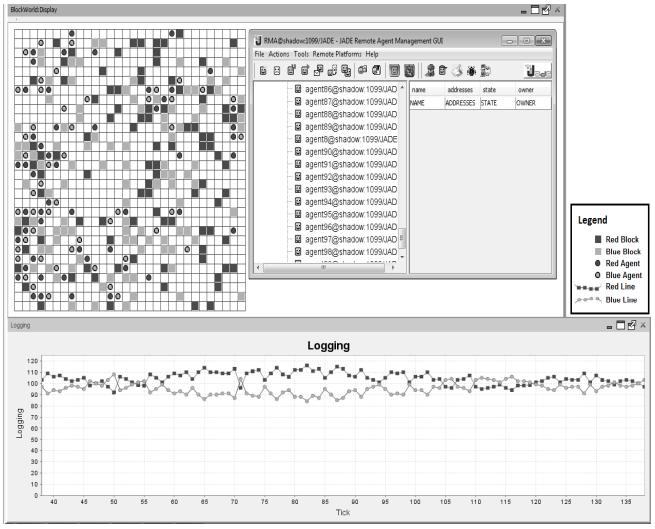

Figure 4. Equilibria Scenario

Scalability is evaluated by testing the coin flip scenario with different starting configurations (see Table I). We compared the memory footprints of several start configurations. The performance is measured by counting the simulation steps (ticks) within the first 10 seconds of simulation. Table I shows the results. It can be seen that the ticks decrease linear, whereas the memory footprints increase super linear. We interpret these results positively, because JRep still scales and performs well for 10000 acting agents.

B. Airport Scenario

Further ongoing work is presented in [5], where we show an airport scenario with *autonomous transportation services* including the macro/micro and interaction level. The airport departure area and the behavior of the autonomous transportation services is illustrated in Repast which provides an efficient visualization of the macro level. The airport infrastructure is represented by a grid in Repast and it is defined by a set of environment elements (e.g., roads, stopovers) that are arranged at design time. In order to sign in the environment the agents (e.g. TAs, charging stations) have to connect to a registry (cf. Fig. 3). This registry

stores references to agents for accessing their properties (e.g., positions). On the micro level, considering the agent architecture presented in [5], an TA has its own plan, goals but it can also perform joint plans. Also, each TA can perceive other TAs that are in his horizon and has to take decisions based on gathered information. For example, TA are signed in a service that collects passenger orders and offers tickets (pickup/drop positions, times) to the TAs. This leads to competition on tickets, roads and charging station, so TAs have to negotiate. For this, the registry provides its reference to enable a connection establishment between the agents. The registry does not coordinate the agents centrally, but coordination, negotiation and conflict handling are done by direct communication between agents. Manipulations of the environment are based on interactions among agents, where complex interactions (e.g., for negotiations) presuppose communication.

VI. Conclusion and Future Work

In this paper, we describe the JRep platform, integrating Repast Simphony and JADE, and its application to modeling and agent-based simulation of complex MAS applications (the smart airport). JRep provides the machinery to comfortably model the behavior and interactions on the agent level (micro level), as well as the instruments to investigate the emerging effects on the overall system (macro level). JRep uses the strengths of both platforms, by efficiently combining their complementary features. Previous work [15] also addressed the combination of JADE and Repast, but their approach has some drawbacks. JRep is an open platform that enables the connection with other agent programming platforms. It is sufficiently generic that it can be used for different application fields. Importantly, we have developed generic concepts and an architecture which provides a novel contribution in itself. We showed how our platform handles (time) synchronization with the environment, scheduling of agents, and registering of new agents. The case study demonstrated the flexibility, scalability and functions of JRep. JRep provides the machinery to comfortably model the behavior and interactions on the agent level (micro-perspective) as well as the instruments to investigate the emerging effects on the overall system (macro-perspective).

In future we plan to enhance JRep by a standardized interface between the agents' behavior and the environment as proposed in [1]. Such an interface will not only ease the integration to alternative simulation and execution backends, but also improve separation of concerns as the environment is decoupled from system level issues like agents' registry and scheduling.

References

[1] T. Behrens, K. Hindriks, and J. Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, pages 1–35, 2010.

[2] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, April 2007.

[3] K. Chmiel, D. Tomiak, M. Gawinecki, P. Karczmarek, M. Szymczak, and M. Paprzycki. Testing the efficiency of jade agent platform. In *Proc. of the 3rd Intern. Symposium on Parallel and Distributed Computing*, ISPDC '04, pages 49–56. IEEE Computer Society, 2004.

[4] C. Deiters, M. Köster, S. Lange, S. Lützel, B. Mokbel, C. Mumme, and D. N. (eds.). Demsy - a scenario for an integrated demonstrator in a smartcity. Technical Report 2010/01, NTH Research School for IT Ecosystems, Clausthal University of Technology, 2010.

[5] M. Huhn, J. P. Müller, J. Görmer, G. Homoceanu, N.-T. Le, L. Märtin, C. Mumme, C. Schulz, N. Pinkwart, and C. Müller-Schloer. Autonomous agents in organized localities regulated by institutions. In *Proc.of IEEE DEST 2011: 5th IEEE International Conference on Digital Ecosystems and Technologies*, 2011.

[6] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations, 1996.

[7] J. P. Müller. *The Design of Intelligent Agents - A Layered Approach*, volume 1177 of *LNCS*. Springer, 1996.

[8] C. Nikolai and G. Madey. Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12, 2009.

[9] M. J. North, T. R. Howe, N. T. Collier, and J. R. Vos. The Repast Simphony Runtime System. In *Proc. of the Agent 2005 Conf. on Generative Social Processes, Models, and Mechanisms*, 2005.

[10] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[11] A. R. Plummer. Model-in-the-loop testing. In *Proc.of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, pages 183–199, 2006.

[12] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In R. B. et. al., editor, *Multi-Agent Programming*, pages 149–174. Springer Inc., USA, 9 2005.

[13] S. F. Railsback, S. L. Lytinen, and S. K. Jackson. Agent-based Simulation Platforms: Review and Development Recommendations. *SIMULATION*, 82(9):609–623, September 2006.

[14] E. Shakshuki and Y. Jun. Multi-agent development toolkits: An evaluation. In *Proc. of the 17th Intern. Conf. on Innovations in Applied Artificial Intelligence*, IEA/AIE'2004, pages 209–218. Springer Verlag, 2004.

[15] M.-J. Yoo and R. Glardon. Combining JADE and Repast for the Complex Simulation of Enterprise Value-Adding Networks. In *AOSE*, pages 243–256, 2008.