12th IMEKO TC1 & TC7 Joint Symposium on
Man Science & Measurement
September, 3 – 5, 2008, Annecy, France

HW/SW HYBRID SIMULATION AS PART OF THE DESIGN OF WIRELESS INSTRUMENTATION SYSTEMS: DISCUSSION ABOUT AN EXPERIENCE

Jean-paul Jamont , Michel Occello

University of Grenoble II -UPMF, LCIS, Valence, France, {jean-paul.jamont,michel.occello}@iut-valence.fr

Abstract: This paper introduces hardware/software hybrid simulation of wireless instrumentation systems as a part of their lifecycle. It presents a state of art of simulators which are close to our hw/sw multiagent simulator, the MultiAgent Hardware Software simulator (MASH) and a discussion about our experience in the use of this tool.

Keywords: instrumentation system design, hw/sw cosimulation, multiagent systems.

1. INTRODUCTION

Our work focuses on the use of embedded multiagent systems (eMAS) to model and design open physical complex systems like wireless instrumentation systems (WIS), collective robotics, distributed control systems, etc. Multiagent systems are well suited to model these complex systems supported by new wireless technologies, because they are more and more distributed and decentralized. They involve numerous hardware/software (hw/sw) entities which enable logical/physical interactions between them and their shared environment.

Modeling and designing these open physical complex systems using MAS highlights two types of specific needs for this context: *needs concerning methods* (our contribution is the DIAMOND method (Decentralized Iterative Approach for Multiagent Open Networks Design [1])) and *needs concerning architectures* (our contribution is the MWAC model (Multi-Wireless-Agent Communication [2])).

This paper is a discussion which focuses on a particular point of the DIAMOND method: exploiting the hw/sw cosimulation in the WIS design. In this paper the word "cosimulation" is used to translate the fact that we include hardware agent and software agent in a same simulation.

In a first part we introduce the origin of our work on hardware/software simulation and related works. In a second part, we present our simulator called MASH. In a last part, we discuss about our experience of hardware/software simulation of MAS.

2. HARDWARE/SOFTWARE SIMULATION OF INSTRUMENTATION SYSTEM

In this part, we present the context and the motivation of our work. We begin by introducing the different ways to design WIS. Then, we present our solution for a lifecycle. In a last part, we present a state of the art of simulators closed to our own tool.

2.1. Why do we require more than a traditional software simulation?

WIS include particular routing protocols, coordination features, data fusion and cooperation algorithms, etc... Three different way exists to design a such system.

Naive solution. This solution consists in implementing the WIS components directly on the target platforms. This solution is possible for simple reactive systems not really complex. Indeed, this solution is very expensive financially and the system development takes an important amount of work. There are three main reasons:

- The genericity is very poor because the designed models are often application-specific and difficult to reproduce in other application contexts.
- The designers address the functional difficulties and the difficulties to embed the software concurrently (limited memory, low power, time constraint, ergonomic constraints and so on.)
- It is difficult to test such a system (case of episodical faults).

Simulation based solution. This solution is very popular. It consists in reproducing the behavior of the global designed system in a virtual environment. Generally, the time is simulated. When the simulation results are consistent with what is expected of the designed system, the designers proceed to its embedment.. The disadvantages of this method are:

- The relevance of the results depends on the quality of the models used in the simulation tool,
- The simulated application is completely decoupled from the physical constraints,
- Once the simulation is complete, the embedded code must developed again,
- Developing the embedded code can introduce some deviations with the originally simulated system. These deviations come from a degradation (due or not) of the models to fit with the available resources and other constraints of the real application.

Emulation. This solution consists in deploying the real code but with a simulation of the low layers (within the meaning of OSI). For example, we can simulate the physical layer and the link layer. The real sensors communicate with

simulated layers. The major disadvantages of this solution are:

- We start with the development of the embedded constrained code : it is a complex system, as said previously, it is important to separate the applicative requirements and the embedded development constraints,
- The quality of the results are strongly coupled with the emulation model,
- Oversizing the available resources is required by the emulator
- A poor control of the transit times in the simulated layers that can distorts the measurements.

MASH simulation. Our belief is that to decrease the difficulty of the deployment of WIS, new simulation type is requiered: the hardware/software hybrid simulation.
On the figure 1, we can see that a traditional software simulation approach focuses only on the MAS design and not on its implementation on a specific platform.

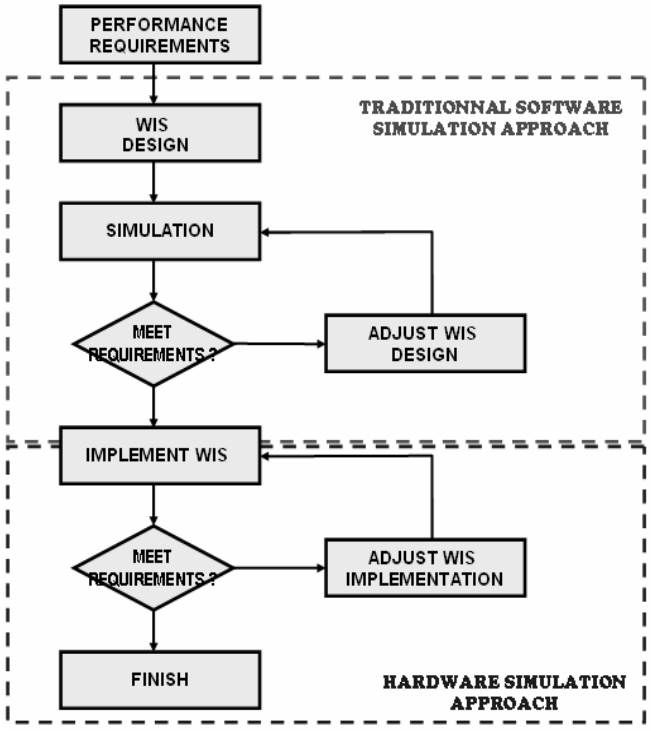

Fig. 1. The hardware simulation that we propose to complete the WIS simulation

The hybrid simulation allows verifying that there is no deviation between the functioning of the real embedded implementation of the WIS and the designed software part of the WIS. The deviation could result from a deterioration of the software model to take into account the hardware restriction i.e. the available memory and the power of the processor.
In a last step, the simulation can be a "pure" hardware simulation. In this case, the simulator replaces the real environment with a virtual environment: perceptions of the sensors in the environment and their neighborhood are modified. The simulation can be used to play a scenario of the real world to test some extreme use cases.

2.2. Related works

The fundamental ideas guiding our work on the hardware/software simulator are quite close to some rare and very recent tools.

In the context of wireless sensor networks, simulators are generally a part of a global framework or suitetool.
Some few works begin with the aim to include hardware entities in software simulation. We can cite some works on these frameworks that allow creating code and simulating it on a hardware device.
SensorSim [3] is a simulation framework for modeling sensor networks. This work is built on the top of the ns-2 simulator [4]. It provides additional features for modeling sensor networks (sensing channel and sensor models, battery models and hybrid simulation). Unfortunately [5], this work has not been updated to support the last releases of ns-2. The middleware based platform SensorWare provides lightweight and mobile control scripts that allow the computation, communication, and sensing resources at the sensor nodes to be efficiently harnessed in an application-specific fashion, through the use of abstraction services. Simulated nodes and real nodes are not time synchronized.
TOSSIM [6] is a discrete event simulator used for testing the code produced with TinyOS [6] and for running it on nodes before their deployment. TinyOS/TOSSIM can provide only a pure simulation or a real deployment of the code on the sensors. It maps the actual code into the simulation platform before run time. Therefore, the language supported by this tool depends on the compiler.
Em* [7] is a toolsuite which allow to develop applications over wireless sensor networks using Linux-based hardware platforms. It supports deployment, simulation, emulation, and visualization of live systems, both real and simulated. Code that has been debugged using all the modes has a good chance to work in a real-world deployment, where it must be scalable and must deal with the effects of the real environment. While deployed code may not work immediately, an "immense amount of real progress can be made in a more friendly environment". Unfortunately, it cannot emulate the real binary code that runs on the real platform.
J-Sim [8] is a component-based discrete event network simulation framework written in Java. J-Sim is useful for network simulation and emulation: it is possible to add one or more real sensor devices. This framework provides support for sensor channels and wireless communication channels, physical media such as seismic channels and power models.
Atemy [9] and Avrora [10] are sensor network emulators and simulators dedicated to AVR processor based systems. One of their advantages is that developers of TinyOS related software can directly use them. Because the processor is chosen in advance, the low-level emulation of the hardware sensor node allows acquiring high-fidelity results but, of course, its genericity is weak for platform based on other processors. Moreover, it seems that the run time

interpretation overhead makes the emulation speed much slower than other approaches [11].
SEMU [11] is a recent framework of simulation for WSN where the hardware/software simulation model is similar to Em* but tries to increase the efficiency of the synchronization. It is based on Linux and use virtual machines for the real code emulation. The essential contribution consists in increasing the real code emulation speed.

We have noticed by their descriptions the limitations of these simulators. In order to be independent of a platform point of view and of a methodological point of view we propose a simulator where only interaction protocols are to be specified. Furthermore, to develop self-organized systems in instrumentation, we want to integrate multiagent features to take advantage of their numerous organizational models and interaction models.

3. THE MASH SIMULATOR

Our hardware/software simulator called *MASH simulator* (MultiAgent Software and Hardware simulator). MASH consists of several simulated/real agent nodes (SimAgent) interacting with an environment component (SimNetwork).

The basic architecture of our WIS simulator and its major simulation models are described on the figure 2.

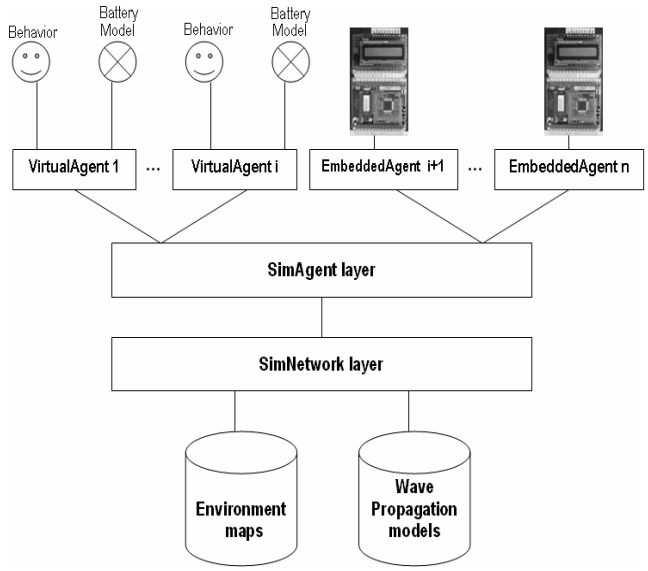

Fig. 2. The hardware simulation that we propose to complete the WIS simulation

It allows to simulate the WIS in three different ways: the software simulation, the hardware simulation and the hardware/software hybrid simulation.

To success real-time constraints, it could be necessary to distribute this simulator on a cluster of computer as done in [12]

SimNetwork Layer. This main component can appear as the inference mechanism for the simulation. Its aims are to provide a useful model of a real environment and its interacting agent. One of the main goals of this component is to find in the neighborhood of an agent, the agents that can physically receive the message transmitted by the sender. The SimNetwork component must provide sensor readable values and must allow effectors to modify the environment state. This part deals with the environment map and the wave propagation models.

The *environment map* describes the physical part of the environment i.e. block of rock, water, air, wall etc. A 2-dimensional grid models it.

The *wave propagation* model is implemented like circular wave propagation through the 2-dimensional grid. It estimates the signal strength measured by a receiver agent S_r when a sender agent sends the message with strength Ss. Considering S_r, we can estimate the probability of a good reception of the message by the receiver agent. Estimating S_r requires to know the geographical position of the sender and of the possible receivers. These positions are stored in the environment map and not in the agent because in a lot of applications, embedded systems cannot know theirs positions. From these positions, we can identify the different media crossed by the signal (air, water, wall etc.) during its propagation.

SimAgent layer. This layer enables the simulation of the hardware/software agent. Each agent possesses its own model and its own architecture.

An agent can be implemented by a software agent (a java class) or an hardware agent which, in the simulator, redirects the message to a serial port. Hardware agents are plugged on the simulator with a serial port.

The time delay added by this serial wrapper is not really a problem because for our application field because, in a real case, it is possible to have some agents with a variable transit time.
It is not realistic to adopt the approach that each entity uses the same computing time. We consider this time as a parasitic time. Each SimAgent transmits its requests to the SimNetwork component that sends the information to all agents that can receive them, in the environment.

Behavior component. The behavior component is the applicative component. It simulates the execution of software on a single sensor node. It receives messages from the other agent. The SimNetwork answers physical component requests like reading sensor values or controlling actuators.
The simulator user must code the applicative part of the agent by deriving a new class from the Application class to implement directly an application.

The battery model. This model is very important to obtain realistic results from the energy efficiency point of view. It is necessary to the software simulated agent to include the energetic consumption in the simulation because all embedded agents must integrate the energy point of view in their reasoning.

The battery model simulates the capacity and the lifetime of the agent energy source. It is difficult to define a universal model because the battery behavior strongly depends on the material used to build the agents. For an

embedded agent, one of its main goals is to increase as much as possible the lifetime of its energy storage. Our simulator implements one of the most simple model of battery: the linear model. Other models are described in [13][14].

In fact, in the case of our WIS simulation we do not need the same precision that is required for the hardware/software partitioning for example, where the aim is to find the best hardware/software compromise. This model allows user to see the efficiency of the user's application by providing how much capacity the agents consume.

In this case it is necessary to have a battery model library (discharge rate dependant model, relaxation model etc.).

The model that we have implemented in our simulator defines the battery like a linear storage of current. The remaining capacity `C` after operation of time t_d can be expressed by the following equation where `C'` is the previous capacity and `I(t)` is the instantaneous current used by the hardware at time `t`:

$$C = C' - \int_{t=t_0}^{t_0+t_d} I(t)dt$$

We can note that if the consumed current always remains the same (the circuit does not switch between sleep mode and active mode for example) we can simply write that:

$$C = C' - I.t_d$$

In our simulation, we define the consumed current depending on some states: radio emission (8.1 mA), radio reception (7.0 mA), cpu active mode (2.0 mA), cpu sleeping mode (1.9 mA).

The simulated applicative agent part, must define other current consumptions for effectors or sensors.

4. MASH SIMULATION WITH A WIS: DISCUSSION ABOUT AN EXPERIENCE THE MASH SIMULATOR

We discuss here about the hw/sw simulation of one of our eMAS used in the EnvSys project [2]. The general idea of this project is to study the feasibility of an underground wireless sensor network. It will allow wireless instrumentation of a subterranean river system. Such a network would present an important interest in many domains: the study of underground flows, the monitoring of deep collecting, flooding risk management, river system detection of pollution risks, etc.

4.1. About the embedded agent architecture

These agents are embedded on autonomous processor cards. These cards are equipped with communication modules and with measuring modules to carry out agent tasks relative to the instrumentation. These cards supply a real time kernel. The KR-51(the kernel's name) allows multi-task software engineering for C515C microcontroller. We can produce one task for one capability. We can then quite easily implement the parallelism inherent to agents and satisfy the real-time constraints.

We have chosen for sensors a classical three-layer embedded architecture (physical layer/link layer/applicative layer). We use the physical layer, employed by the NICOLA system, a voice transmission system used by the French speleological rescue teams [15]. This layer is implemented in a digital signal processor rather than a full analogical system. Thereby we can keep a good flexibility and further we will be able to apply a signal processing algorithm to improve the data transmission. The link layer used is a wireless version of the CAN (Controller Area Network) protocol stemming from the motorcar industry and chosen for its good reliability. The applicative layer is constituted by the agents' system.

Hybrid architectures enable to combine the strong features of each of reactive (to the message) and cognitive capabilities (to detect inconsistencies and start a local re-organization). The ASTRO hybrid architecture [16] is especially adapted to a real time context. The integration of deliberative and reactive capabilities is possible using parallelism in the structure of the agent.

Separating Reasoning/Adaptation and Perception/Communication tasks allows a continuous supervision of the evolution of the environment. The reasoning model of this agent is based on the Perception/Decision/Reasoning/Action paradigm.

The cognitive reasoning is thus preserved, and predicted events contribute to the normal progress of the reasoning process.

4.2. Discussion about the different simulations

We discuss in this part about the three different types of simulation in evolved in the MASH simulator.

Our simulator allows to evaluate the WIS and to quantify the emergence inferred by the system.

It allows comparing our multiagent approach with other approaches because it provides a scenario editor/player.

The figure 3 shows how windows of a software/hardware joint simulation that evolves 95 agents: 91 software agents and 4 hardware agents (see fig. 4).

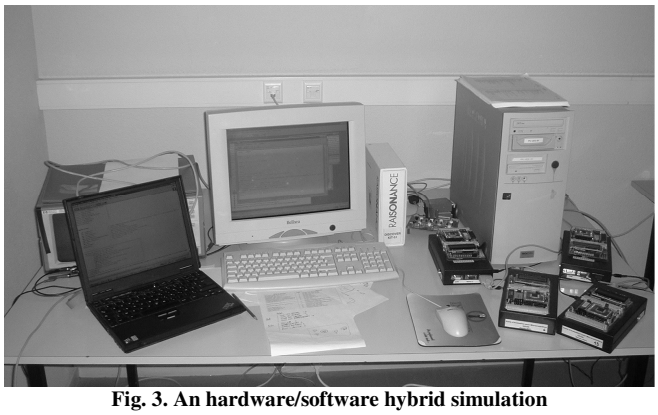

Fig. 3. An hardware/software hybrid simulation

Pure software simulation. At this level, the simulator allows to evaluate and improve such agents' software architectures and the cooperation techniques that they involve. The scenario player allows to compare the MAS solution to traditional solutions or other MAS solution.

In this use case, we have compared our solution with three other solutions based on ad-hoc protocols like the DSDV protocol (Destination-Sequenced Distance-Vector protocol

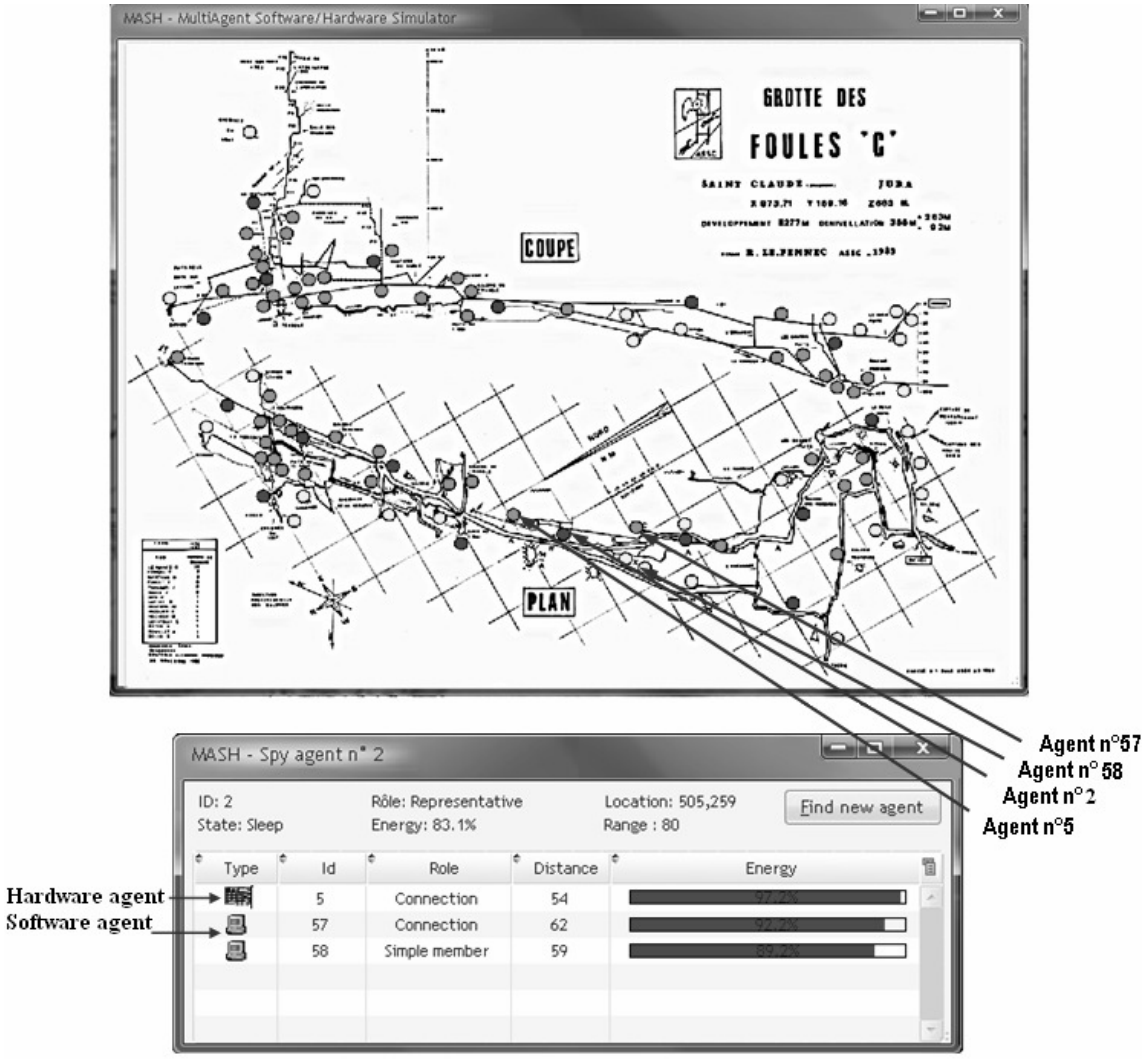

Fig. 4. Screen copy of the simulator

[17]) and the natural DSR protocol (Dynamic Source Routing protocol [18]).

The simulator enables to measure different criteria as the group average density, the global transmitted volume, the transmitted volume variation and the average memory used by the agent. Another important criterion is the efficiency that is defined in our application as the theoretical useful volume of the optimal way divided by the volume of each transmitted communication. These measures allow to adjust the sensitivity of the parameters of the self-organization process.

Design an artificial complex system features like the a self-organization process based embedded system is easier to design in a pure software way. In fact, the difficulties raised by this type of process are isolated from the difficulties of the hardware design and specific perturbation of the real world. Therefore, we can write that the software simulations allow us to prepare the embedded part of the multiagent system.

However, a major problem of these software simulations is that the quality of the simulation depends on the quality of the different models. The embedded multiagent simulation integrates many models that come from non-multiagent community. These models concern the environment [19], the wireless channels [20], the battery devices [14] etc.

Hybrid software/hardware simulation. One of the hardware/software hybrid simulation aims is to test embedded agents in a very large system with a low financial cost. For example, we can use 1000 virtual agents in a same simulation and only four embedded agents. Of course, the embedded agents must be judiciously situated in the topology (the simulated environment) in regard with that the designer wants to see (a re-organization of the self-organization process, a fault tolerance feature, a possible reaction of the MAS against a disturbance etc.).

Another advantage of the hardware/software hybrid simulation is the support provide in the debugging phase. In fact, we can use the simulator in a hardware agent-debugging step. Debugging is essentially a process of exposition of program's internal states relevant to its abnormal behavior and pinpointing the cause. Visibility of execution states is a determining factor of how difficult the debugging task is. With this type of simulation it is easy to compare the deviation between the hardware behavior of an agent and the simulated behavior of the same agent. Contrary to conventional debuggers, we do not inspect program counters, memories and registers but we focus on the agent data.

Pure hardware simulation. The pure hardware simulation aim is to test the final eMAS entities in an aggressive simulated environment. In this simulation all the agents are run natively (the code is running on the real platform) but these agents interact together through the simulated world. It allows simulating some event like movement more easily than in the real world: we can simulate easily the agent move without physically move the agents. It is important in an efficiency evaluation phase to be sure that all the compared solutions are tested with the same scenario.
For example, if we want to test the eMAS in an underground river system, it is easier to model the flow, the possible rock fall, etc.
In a debugging phase, it is possible to debug the agent with a serial debugging backchannel (independently from the MASH simulator).
However, the simulator can be used as a visualization and analysis tools (the agent representation on the simulator allows to spy the intern state of the physical corresponding agent).

A great disadvantage of this simulation is that, to have realistic simulations, one needs many devices and the associated financial cost is very important.

5. CONCLUSION

We presented in this paper the tool MASH and the use of different simulations it proposes to

- design a virtual wireless instrumentation multiagent system
- prepare for the boarding of this system,
- test the wireless instrumentation multiagent systems and debugger,
- evaluate the performance of this system.

This work takes place clearly within the engineering of wireless instrumentation systems. Our goal is to contribute make more relevant the use of cooperative agents to design such systems. Further, evolving physical agents and virtual agents in a same societies could initiate a more theoretical research on the nature of these hybrid societies as it is already the case with the collaboration of human agents and software agents.

REFERENCES

[1] J.-P. Jamont, M.Occello, Designing embedded collective systems: The DIAMOND multiagent method, IEEE International Conference on Tools with Artificial Intelligence, IEEE Computer Society, vol 2, 2007.

[2] J.-P. Jamont, M. Occello, A self-organization process for communication management in embedded multiagent system, IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IEEE Computer society, 2007.

[3] S. Park, A. Savvides and M. B. Srivastava, Simulating networks of wireless sensors, Proceedings of the 2001 Winter Simulation Conference, pp 1330-1338, ACM, Dec. 2001.

[4] K. Fall and K. Varadhan, The ns ManuaL (formerly ns Notes and Documentation), December 2007. *http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf*,

[5] I. T. Downard, Simulating Sensor Networks in NS-2, Final Technical Report No. ADA423595, Naval research lab, Washington DC, 2004.

[6] P. Levis et al., TOSSIM: accurate and scalable simulation of entire tinyOS applications, Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, 2003.

[7] L. Girod, T. Stathopoulos, J. Elson, M. Lukac, A. Parker, N. Xu, R. Kapur and D. Estrin, EmStar: A Software Environment for Developing and Deploying Wireless Sensor Networks, Proceedings of the USENIX Annual Technical Conference, USENIX, pp. 283-296, June 2004.

[8] A. Sobeih and J.C. Hou, A simulation framework for sensor networks in J-sim, Technical Report UIUCDCS-R-2003-2386, University of Illinois at Urbana-Champaign, Department of Computer Science, November 2003.

[9] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. Baras, and M. Karir. ATEMU: A fine-grained sensor network simulators. In Proceedings of SECON'04, First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004.

[10] B. Titzer , D. K. Lee and J. Palsberg, Avrora: scalable sensor network simulation with precise timing, Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, pp 477-482, 2005.

[11] S.-H. Lo, J.-H. Ding, Sheng-Je Hung, J.-W. Tang, W. Tsai, and Y.-C. Chung. Semu : A framework of simulation environment for wireless sensor networks with co-simulation model, Advances in Grid and Pervasive Computing : Second International Conference, pages 672–677, Paris, France, May 2007. LNCS 4459, Springer Verlag.

[12] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, Mac Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In 5th Symposium on Operating System Design and Implementation. USENIX, December 2002.

[13] S. Park, A. Savvides, and M. B. Srivastava. Simulating networks of wireless sensors. In Proceedings of the 2001 Winter Simulation Conference, pages 1330–1338. ACM, December 2001.

[14] M. Handy and D. Timmermann. Simulation of mobile wireless networks with accurate modelling of non-linear battery effects. In Procedings of Applied Simulation and Modelling. Acta Press, September 2003.

[15] N. Graham, The Nicola Mark II – a New Rescue Radio for France, the CREG Journal, vol 38,1999.

[16] M. Occello, Y. Demazeau, and C. Baeijs. Designing organized agents for cooperation in a real time context. In Collective Robotics, volume LNCS/LNAI 1456, pages 25–73. Springer-Verlag, March 1998.

[17] C.E. Perkins, E.M. Royer, and S. Das. Highly dynamic destination-sequenced distance-vector (dsdv) routing for mobile computers. In ACM SIGCOMM'94, 1994.

[18] D.-B. Johnson and D.-A. Maltz. Dynamic source routing in ad hoc wireless networks. In Mobile Computing, pages 153–181. Kluwer Academic Publishers, 1996.

[19] Y. Z. Mohasseb and M. P. Fitz. A 3d spatio-temporal simulation model for wireless channels. In IEEE International Conference on Communications, volume 6, pages 1711 – 1717. ACM, June 2001.

[20] G. Judd and P. Steenkiste. A software architecture for physical layer wireless network emulation. In Proceedings of the First ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and Characterization, pages 97–107. ACM, September 2006.