# Decentralized Intelligent Real World Embedded Systems: A Tool to Tune Design and Deployment

Jean-Paul Jamont, Michel Occello, and Eduardo Mendes

Université de Grenoble-Alpes, LCIS, 26000 Valence, France
firstname.lastname@lcis.grenoble-inp.fr
http://lcis.grenoble-inp.fr

**Abstract.** This paper presents an approach and a tool, called MASH, to design of real world decentralized intelligent systems. MASH enables the simulation of distributed systems including virtual and real world embedded nodes according to realistic physical models. We present the key features of this tool and its architecture.

**Keywords:** embedded MAS, deployment, simulation.

## 1 Introduction

**Context.** More and more real world intelligent systems consist of lots of small interconnected devices which interact together. These components must be small, inexpensive, and therefore as simple as possible. Designing such solutions requires to take into account strong criteria like energy consumption, low CPU power, low memories etc.

In the literature, we can consider two types of approach to design such systems. *Decentralized approaches* as multiagent systems (MAS) and other distributed artificial intelligence solutions. Systems are seen as sets of interacting autonomous entities (agents) reasoning about a partial description of their environment. *Centralized solutions* as traditional automation based solutions. A global model of the environment is defined and maintained. The decisional process uses this model to decide and to act.

In the first case, interesting applicative properties (like stability) are difficult to prove because of the decentralized and distributed nature of the system. The possible number of exchanged messages due to the cooperation process between the distributed entities can be a problem too. In the second case, obtaining a realistic global model can be expensive in term of computation and in term of information transport.

We focus on *decentralized* intelligent systems. Designing software for such systems is a difficult task due to the inherent complexity at both conceptual and implementation levels. Systems can be observed at both an individual level and a social level. The *individual level* focuses on capacities, knowledge and goals of entities. The *social level* focuses on global aspects of the whole system, external

Y. Demazeau et al. (Eds.): PAAMS 2013, LNAI 7879, pp. 133–144, 2013.

expression of interaction and cooperation situations. From local individual interactions can emerge behaviors at the system level which are difficult to predict.

**Problem.** Designing and deploying a decentralized intelligent systems requires to simulate the behavior of the whole designed system. Because of the complexity of applicative problems, simulation tools must help the designer to investigate important features of local control strategies and to inspect their effects on the global behavior. The precision of the models used in the simulation (physical environment, energy consumption, wave propagation...) has a significant impact on the quality of the whole solution that will be really deployed in the real world. To support the system deployment, we must simulate decentralized intelligent systems constituted by virtual (simulated) parts and embedded parts really deployed in the real world.

Such a tool must support an important variety of models coming from different fields. These models are mainly *organizational models* (self-organization process, hierarchy management etc.), *interaction models* (contract net protocol, recruiting interaction protocol etc.), *physical environment models* (including different classes of application dependent models as wave propagation models, thermal dissipation model, fluid flow etc.) and *user models* (implementing different types of user's specific needs dependent behavior).

**Contribution.** MASH (MultiAgent Software Hardware simulator) tries to meet these requirements. This tool is used according to the following approach (figure 1). This approach is involved in a more complete system design methodology called DIAMOND [11] which is not presented here.

When the simulated solution meets the requirements it is necessary to embed the solution in the real world devices. A specific effort is needed to tune algorithms in order to fit resources of devices. Algorithms must be simplified to accommodate, for example, memory limitations, reduced computation capacities etc. Deviations of the global behavior may result from these modifications. Tools

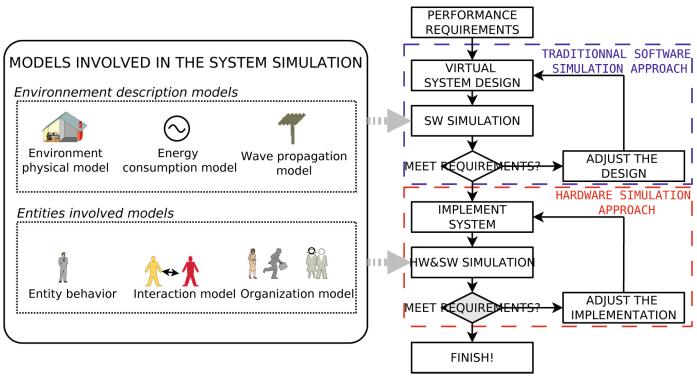



**Fig. 1.** Using MASH to design and to deploy real world MAS

must help to control effects of local changes in the behavior of entities on the global behavior.

This paper gives an insight to our pragmatic design approach and focuses on its associated tool. Applications are given in a companion demo paper within the same proceedings [12].

The structure of the paper is as follows: Section §2 introduces the MASH architecture. In section §3 we focus on the key features of our tool : the virtual/real world mixed society simulation and the use of realistic physical models. For each of these key features the related works are exposed.

## 2 An Introduction to MASH Architecture

### 2.1 Preliminary Definitions

Embedded MAS include different types of agents:

- **Embedded agents** are agents embedded in the real world which can perceive and act on physical environment. They are often constituted by a software part and a hardware part. Soccer robots, autonomous vehicle, intelligent sensors are embedded agents.
- **Software agents** are traditional agents which can perceive and acts on virtual environment.
- **Virtual agents** are a type of software agents used to simulate the behavior of real world embedded agents.
- **Avatar agents** represent embedded agents in virtual societies. They enable embedded agents to interact with software agents. They link the simulated MAS with behaviors computed on physical devices.

**Virtual environments** are environments in which parameter values are estimated/computed from their physical models. Parameter values of physical environments are acquired by sensors and they are modified by effectors.

We call **virtual society** the set of software agents of the multiagent system. Embedded agents are not taken into account contrary to their avatars.

We call **instruction** a method call on an agent (or an object) at a specified date `(date,<object_identifier>.<method_name>(<param1>,...))` . A **scenario** consists in a set of instructions.

### 2.2 Architecture Overview

MASH enables involving software simulation (involving software agents), the hardware simulation (only real world embedded agents operating on a simulated environment) and the virtual/real world hybrid simulations (involving software agents and real world embedded agents operating on a simulated/real world mixed environment).

In figure 2, we can see in the background the main windows which allow to view the system according to customized criteria (defined by the designer). It

is possible to spy agents i.e. to inspect their internal states and the history of events. The inspected agent is Agent2 which is a real world agent. Among the various events, we can see the bytes received by this agent and their translation into logical messages.

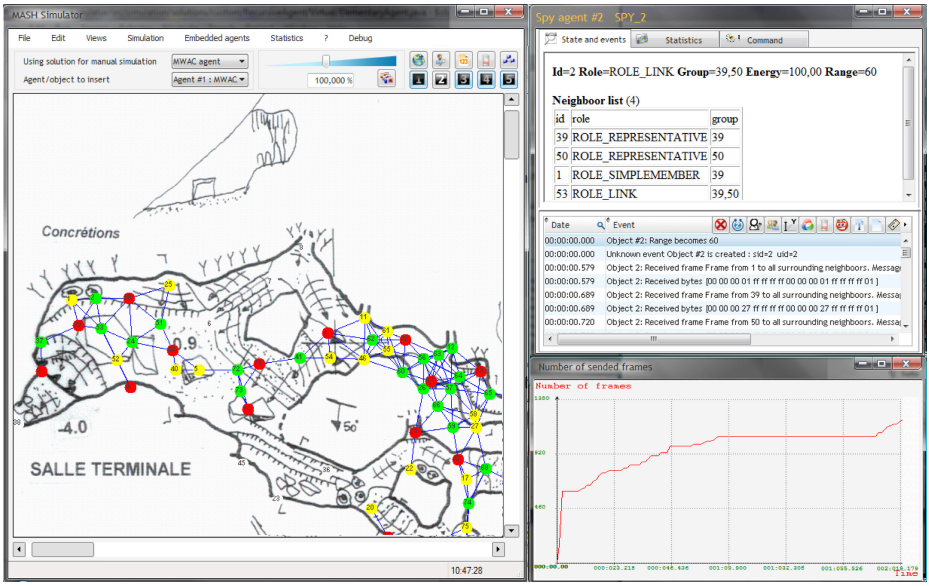



**Fig. 2.** The MASH main windows

Virtual agents and real world embedded agents are abstracted by an *Individual agent manager* (figure 3). The *Individual Agent Manager* enables the integration in the simulation of virtual and real world agents. Each agent possesses its own model and its own architecture. An agent can be implemented by a software agent (as a java class) or its behavior can be computed in a real world embedded agent. In this case, an avatar translates the logical call of methods and exchanged messages to its wrapped embedded agent. Physical connections are implemented according to a given bit specification (defined by user). The avatar allows to give a graphical representation of the real world embedded agent in the MAS graphical representation.

The *Behavior component* is the applicative component. It simulates the execution of software on a single node. It processes messages received from other agents. The agent's decision cycle dedicated to the functional aspect of the application is implemented here.

Agents interact together and with the environment through the *society manager* and the *Environment manager*. The *Society Manager* defines the locality of an agent. In other words, it enables an agent (1) to identify its neighborhood i.e. the agents that can physically receive the messages it transmits, (2) to

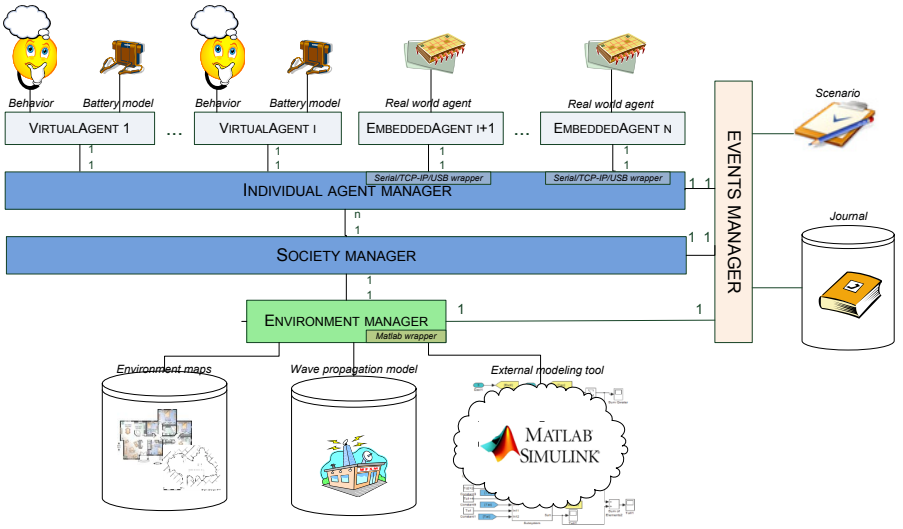



**Fig. 3.** Simplified architecture of the MASH simulator

access the environment values that its sensors can measure and (3) to act on the environment (to modify environment parameters) with their effectors.

The *Environment manager* computes different physical models to allow a realistic simulation. When an agent wants to capture an environment value , this component decides firstly the accessibility of this data by the agent (Does the agent have the appropriate sensor? Does the agent is in the good geographical area?). Secondly, it returns the value or it throws an exception.

Concerning the *scenario* processing, the simulator uses the reflection API of java which allows to examine or to modify the runtime behavior of Java applications. MASH uses it to examine properties of agents, including their declaration and their contents. Main advantages of use of this API is the possibility, when a scenario is running, to call dynamically user defined methods/services without any additional declaration of possible actions.

## 3 Key Features

In this part, we focus on main particularities of MASH. For each of them, we present a quick review of related works and give an insight of its implementation in MASH.

### 3.1 Simulation of Real World/Virtual Societies

Software actors and hardware actors mixed systems are increasingly common, but relatively few tools enable their development.

**Related Works.** Some works support simulation of systems including hardware entities and software entities. These works often belong to the wireless networks field. These simulators [14,7,22,23] are often specialized in the study of a specific type of model (battery discharge model, wave propagation model...). We identified no contribution in the field of decentralized artificial intelligence which enables to involve software agents and real world embedded agents in a same simulation. In [24], authors make a virtual discrete environment from the physical observations to plan actions. In [9], the simulator acquires data from sensors to obtain more realistic virtual simulations.

**Concerning MASH.** In our tool, all real world embedded agents have an avatar in the simulated society. These avatars are managed by the *Individual agent manager*. Their main role is to translate logical messages into physical ones and vice versa. They can be used to compare real world agent behaviours to the behaviour of their virtual implementation.

As an instance, figure 4 shows the expected role and the measured role of an hardware agent included in the simulator. At `t=152.45s`, we can see that the real world agent requires more time to choose its next role (the CPU clock is lower than the virtual agent clock). At `t+dt`, we can see that the real world agent has chosen an unexpected role (simple member). Reason is that a more important amount of time is needed to analyze the received messages.

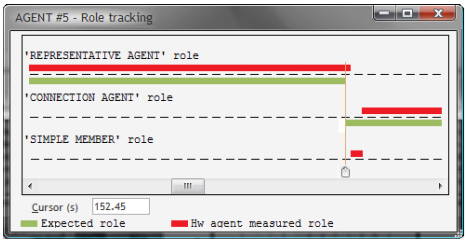



**Fig. 4.** Tracking an agent role

### 3.2 Simulation Using Realistic Physical Models

The majority of distributed intelligence systems concerns only virtual applications. For example, in the context of home automation, simulators belonging to the distributed intelligence field focus on software problems like interaction protocols used to negotiate users' needs, decision making and distributed problem solving to adapt these needs to energy limitation [2,1], data-mining to match the specific situation to a previously observed one [6].

The associate tools like [3,18,13,17] are not suitable for the design of real world systems especially because physical laws of these environments are indeed reduced to their simplest expression.

**Environment Models.** The environment models in which agents evolve are often reduced to their simplest form (in most of works of ambient artificial intelligence). However their impact on simulation results is very important. We present related works and a use of more realistic models in MASH.

**Related Works.** The most interesting solutions are those that use the well known tools Labview [5] or Matlab[16]. In [20,4], authors use Labview to implement a MAS. In their simulations an agent is implemented as a virtual instrument. Of course, by this way authors lose the advantages of multiagent simulators (large scale simulation, multiagent specific models...). Very recently, in another context a matlab/simulink multiagent toolkit for distributed networked fault tolerant control systems has been proposed [15]. In [21], MACSimJX an extension of Jade is proposed to enable interaction between Simulink and this tool.

**Concerning MASH.** Matlab is undoubtedly the most suitable tool to model physical systems even if the creation of a model requires to have serious knowledge about the physical law that we want to model.

A Simulink model is presented as block diagram. Such diagrams enable to solve set of algebraic equations and ordinary differential equations. This block organization enables easy reuse of already developed blocks and allows to have a better comprehension of the whole model.

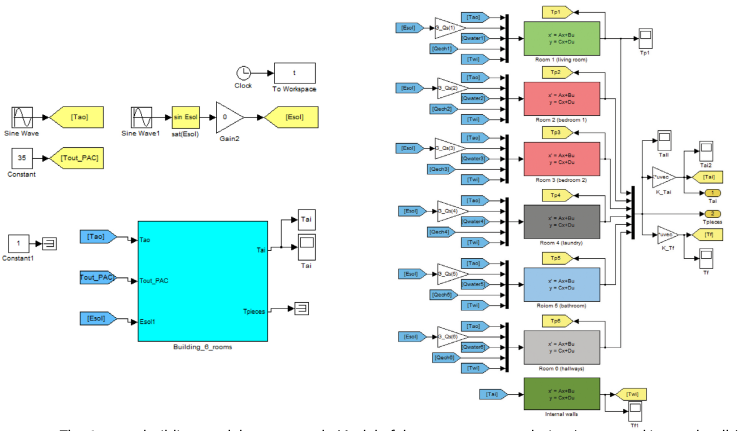
In [10], we propose to model[1] temperature evolution of each room of a building. The temperature of a room depends, on one hand, of elements that are only dependent on that particular room and of external parameters, and on the other hand of shared components, such as walls and doors, with other rooms. Consequently, we propose the following methodology: making one model for each piece with its own components (air, floor, ceiling, external walls, windows, internal heating sources...) and one model for each linking component (walls and doors between rooms, floor/ceiling for multilevel building...). Mainly, the first models (room model) are connected using the second models (internal building walls). This methodology permits to connect and disconnect the rooms very efficiently without having to rewrite the equations of the different components. One has first to define the components (rooms and internal walls) connectivity and second the physical parameters of each component.

The resulting modular model is implemented into MatLab/Simulink. The model enables to change the building configuration easily. Some parts of the model are exposed in figure 5. We find on this figure the entire model of the 6 rooms building that we simulate (fig. 5a) and how we compute the temperature of the rooms and of the internal walls composing the building according equations described in [10] that are in state space form (fig. 5b).

To use a physical model defined with Matlab/Simulink, MASH can interact with this tool by two ways : using a client/server approach or by files exchange.

Concerning the first solution, the Matlab Real-Time Workshop is able to provide the code to implement both a server and a client. We use only its server code generation. MASH supplies the client code to access to the physical model

[1] This model results from our collaboration with automation researchers.




a. The 6 rooms building model b. Model of the temperature evolution (rooms and internal walls)

**Fig. 5.** Environment physical model : Part of the Matlab block model

parameters. Because this solution requires to buy a specific module, we preferred to develop a little Matlab script which enables a double queue file communication. These text files (default names: `data.in` (inputs of the physical model i.e. outputs of MASH) — `data.out` (outputs of the physical model i.e. inputs of MASH) ) enables to share parameters and measures. An example of these exchanged file is given in figure 7.

Then, MASH therefore uses a very simple algorithm (alg. 1) to put the update data relating to the environment in which agents evolve.

**Algorithm 1.** Integrating Matlab/Simulink computed values

**repeat**
  Read environment values in `data.out.xml`
  Update MASH virtual environment
  Agents act
  Write controllable values by agents in `data.in.xml`
**until** simulation.isRunning()

**Energy Consumption Model.** Energy consumption models are very important to obtain realistic results from the energy efficiency point of view. It is necessary for 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.

**Related Works.** 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 behaviour 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

```
<?xml version= "1.0">
<!-- Written on 24-Aug-2011 10:52:41 using the XML Toolbox for Matlab -->
<root>
   <data1>
     <name>simulated_time< /name>
     <type>float< /type>
     <value>30< /value>
     <unit>day< /unit>
     <description>Simulation time : day< /description>
      <direction>out< /direction>
   < /data1>
    ...
   <data17>
     <name>Tai_03_03< /name>
     <type>float< /type>
     <value>32.678271140525< /value>
     <unit>celcius< /unit>
     <description>Ambient air temperature < /description>
     <location>room 3< /location>
     <direction>out< /direction>
   < /data17>
    ...
< /root>
```

**Fig. 6.** Example of XML exchanged files between Matlab/Simulink and MASH

the lifetime of its energy storage. One of the most simplest models of battery is the linear model. Other models are described in [19,8].

In a lot of simulators of decentralized intelligent systems, energy consumption models are not really available. Considering the large variety of batteries existing in the real world, it will be interesting to supply several models.

**Concerning MASH.** We defined an open structure which allows to implement several models. The more simple one implemented in MASH consists in defining the battery as a linear storage of current. The remaining capacity $C$ after operation of time $td$ 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 $t$ : $C = C' - \int_{t=t_0}^{t_0+t_d} I(t)dt$. The designer must tune the values involved in this model. For instance, we can 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). A designer can add new states to model others current consumptions (sensing consumptions, acting consumptions...).

**Wave Propagation Model.** If for most applications, the use of a particular model of wave propagation is not essential, it may be necessary when the system environment is disturbed non-uniformly. We present here the use of such models in MASH.

**Related Works.** In tools specialized on wireless network simulation like NS2, numerous models are available. These models are dedicated to specific environment (cities, home indoor...). We can tune some parameters like the bit rate, the bit error rate and so on. In a lot of distributed intelligence simulator, these models don't really exist. Authors consider an arbitrary range or that the environment

is fully covered by access points. Associated energy consumption models are often poorly understood by software specialists. An example of false assumption often found in papers is that a system (as a WSN) spends more energy during transmissions rather than during receptions.

**Concerning MASH.** The wave propagation model can be modelled as a part of the Matlab environment model. Even that, to simplify reuse of such a model, we separate this specific model. So it is possible to have two concurrent Matlab simulation (one for modelling the wave propagation and one for others physical phenomena like heat transfer).

If a so advanced model is not necessary, MASH offers a simple model implementing the Friis free space transmission equation (used in telecommunications engineering) which does not need to use Matlab. The wave propagation model is implemented like circular wave propagation through the 2-dimensional grid. MASH estimates the received power measured by a receiver agent $Pr$ when a sender agent sends the message with transmitter power $Ps$. Considering the receiver signal strength, we can estimate the probability of a good reception of the message by the receiver agent. Estimating $Sr$ 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 their positions. From these positions, we can identify the different media crossed by the signal (air, water, wall etc.) during its propagation.

## 4 Conclusion

This paper presents an approach to design and deploy real world systems based on decentralized intelligence based on MAS. We presented MASH a tool supporting this approach.

Building such systems imposes to design and to tune embedded agents. MASH enables using both simulated and really embedded agents in a same system. One of its major benefits is so the possibility to control possible deviations obtained at the system level from the individual level behaviour modification. A second ability of MASH is it supply realistic physical models for agent environment. Coupling virtual agents/embedded agents with physical models leads to a realistic development tool, bringin closer MAS and physical systems.

Currently MASH supports simulation of systems incorporating populations of 600 to 900 agents (depending on the agent architecture complexity). So the next main improvement of MASH consists in enabling the society manager to distribute individual agent manager on clusters of machines.

## References

1. Abras, S., Ploix, S., Pesty, S., Jacomino, M.: A multi-agent home automation system for power management. In: Proc. of the 3rd Int. Conf. on Informatics in Control, Automation and Robotics, Intelligent Control Systems and Optimization, pp. 3–8. INSTICC Press (2006)

2. Begg, R., Hassan, R.: Artificial neural networks in smart homes. In: Augusto, J.C., Nugent, C.D. (eds.) Designing Smart Homes. LNCS (LNAI), vol. 4008, pp. 146–164. Springer, Heidelberg (2006)
3. De Carolis, B., Cozzolongo, G., Pizzutilo, S., Plantamura, V.L.: Agent-based home simulation and control. In: Hacid, M.-S., Murray, N.V., Raś, Z.W., Tsumoto, S. (eds.) ISMIS 2005. LNCS (LNAI), vol. 3488, pp. 404–412. Springer, Heidelberg (2005)
4. Conte, G., Scaradozzi, D., Perdon, A., Morganti, G.: Mas theory for resource management in home automation systems. Journal of Physical Agents 3, 15–19 (2009)
5. Fairweather, I., Brumfield, A.: LabVIEW: A Developer's Guide to Real World Integration. Taylor and Francis (2011)
6. Galton, A.: Causal reasoning for alert generation in smart homes. In: Augusto, J.C., Nugent, C.D. (eds.) Designing Smart Homes. LNCS (LNAI), vol. 4008, pp. 57–70. Springer, Heidelberg (2006)
7. Girod, L., Elson, J., Cerpa, A., Stathopoulos, T., Ramanathan, N., Estrin, D.: Emstar: A software environment for developing and deploying wireless sensor networks. In: Proc. of the USENIX Annual Tech. Conf., pp. 283–296. USENIX (2004)
8. Handy, M., Timmermann, D.: Simulation of mobile wsn with accurate modelling of non-linear battery effects. In: Applied Simulation and Modelling. Acta Press (2003)
9. Hassaine, F., Moulton, R., Fink, C.: Composing a high fidelity hla federation for littoral operations. In: Symp. on Applied Computing, pp. 2087–2092. ACM (2009)
10. Jamont, J.P., Mendes, E., Occello, M.: A framework to simulate and support the design of distributed automation and decentralized control systems: application to control of indoor building comfort. In: Proc. of the IEEE Symp. on Computational Intelligence in Control and Automation, pp. 80–87. IEEE (2011)
11. Jamont, J.P., Occello, M.: A multiagent method to design hardware/software collaborative systems. In: Proceedings of the 12th International Conference on Computer Supported Cooperative Work in Design, pp. 361–366. IEEE (2008)
12. Jamont, J.-P., Occello, M.: Using mash in the context of the design of embedded multiagent system. In: Demazeau, Y., Ishida, T., Corchado, J.M., Bajo, J. (eds.) PAAMS 2013. LNCS, vol. 7879, pp. 283–286. Springer, Heidelberg (2013)
13. Kim, I., Park, H., Noh, B., Lee, Y., Lee, S., Lee, H.: Design and implementation of context-awareness simulation toolkit for context learning. In: IEEE Conf. on Sensor Networks, Ubiquitous, and Trustworthy Computing, pp. 96–103 (2006)
14. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire tinyos application. In: Proc. of the Int. Conf. on Embedded Networked Sensor Systems, pp. 126–137. ACM (2003)
15. Mendes, M., Santos, B., da Costa, J.S.: A matlab/simulink multi-agent toolkit for distributed networked fault tolerant control systems. In: Proc. of the 7th IFAC Symp. on Fault Detection, Supervision and Safety of Technical Processes (2010)
16. Moore, H.: MATLAB for engineers. ESource–the Prentice Hall engineering source. Prentice Hall (2001)
17. Nguyen, T.V., Nguyen, H.A., Choi, D.: Development of a context aware virtual smart home simulator. CoRR 1007.1274 (2010)
18. O'Neill, E., Klepal, M., Lewis, D., O'Donnell, T., O'Sullivan, D., Pesch, D.: A testbed for evaluating human interaction with ubiquitous computing environments. In: Proc. of the 1st Int. Conf. on Testbeds, Research Infrastructures for the Development of Networks and Communities, pp. 60–69. IEEE Computer Society (2005)
19. Park, S., Savvides, A., Srivastava, M.B.: Simulating networks of wireless sensors. In: Proc. of the 2001 Winter Simulation Conference, pp. 1330–1338. ACM (2001)

20. Ponci, F., Deshmukh, A., Monti, A., Cristaldi, L., Ottoboni, R.: Interface for multiagent platform systems. In: Proceedings of the IEEE Instrumentation and Measurement Technology Conference, pp. 2226–2230. IEEE (2005)
21. Robinson, C.R., Mendham, P., Clarke, T.: A multiagent approach to manage communication in wis. Journal of Physical Agents 4(3), 489–503 (2010)
22. Sobieh, A., Hou, J.: A simulation framework for sensor networks in j-sim (2003)
23. Titzer, B., Lee, D.K., Palsberg, J.: Avrora: scalable sensor network simulation with precise timing. In: Proceedings of the 4th Int. Symp. on Information Processing in Sensor Networks, pp. 477–482. IEEE (2005)
24. Weyns, D., Schelfthout, K., Holvoet, T., Lefever, T.: Decentralized control of e'gv transportation systems. In: 4th Int. Joint Conf. on Autonomous Agents and Multiagent Systems - Industrial Applications, pp. 67–74. ACM (2005)