Design of embedded multiagent systems: discussion about some specificities

Jean-Paul Jamont and Michel Occello
University of Grenoble II
LCIS-INPG Laboratory
51 Rue Barthelemy de Laffemas, 26000 Valence, France
{jean-paul.jamont,michel.occello}@iut-valence.fr

Abstract

Multiagent systems satisfy to design requirements for open physical complex systems. However, up to now, no method allows to build software/hardware hybrid multiagent systems. This paper introduces the DIAMOND method. We present its spiral life cycle which associates a multiagent oriented analysis iteration and a component based design iteration. The agent architecture is used as a pattern to build agents using components. A partitioning phase is used to choose software/hardware components implementation.

Introduction

Our work deals with the modeling and the design of open physical complex systems. These systems involve numerous software/hardware entities which enable logical/physical interactions between them and their shared environment. These entities possess their own goals but participate to the accomplishment of the goals of the global system. A global behavior can emerge through their interactions . The emergence process is a way to obtain, from cooperation, dynamic results that cannot be predicted in a deterministic way. We have identified four classes of open physical complex systems: control systems, processing systems, communication systems and interactive systems. Because these systems are supported by new wireless technologies, they are more and more distributed and decentralized.

Through the use of multiagent systems (MAS) to model open physical complex systems emerges two types of needs : needs concerning specific system architectures (our contribution is the MWAC model (Multi-Wireless-Agent Communication) based on our previous work on wireless sensor networks [12]) and needs concerning methods. In this paper, we focus on methodological specificities and on our contribution, the DIAMOND method (Decentralized Iterative Approach for Multiagent Open Networks Design [10, 9]). We try to answer to some questions asked by this kind of applications all along the lifecycle and in the choice of formalisms.

In a first part we will talk about some specific features for the lifecycle : "what's the better lifecycle to design open physical complex systems?", "how to specify an hybrid hardware/software MAS?", "how to join, in this lifecycle, the requirements of both software and hardware parts ?". The necessity to introduce a particular requirements analysis phase to take into account the specificities of embedded MAS will be tackled in the second part and the third part. In a fourth part, we will focus on the impact of local constraints of the entities on the global functionalities of the system during the decomposition into a multiagent solution. In a fifth part, we will try to answer to the question "how to design software parts, hardware parts and how to integrate them in an operational whole system ?". In a sixth part we try talk about the specifities of embedded multiagent system simulation.

1 The approach

The lifecycle of traditional methods applied to design hardware/software hydrid systems (see fig. 1) starts with a requirements analysis followed by a partionning step. During this partitioning step, the designer chooses the system parts which must become either hardware or software parts. At this stage, the two differents parts are designed in parallel. At the end of the lifecycle, the two parts are integrated into a whole operational system. Through this integration step (and the following tests) some problems can emerge. These problems can question the software design, the hardware design or the both. Furthermore, it can be necessary to modify the whole result of the partitionning ! This type of lifecycle doesn't allow to take into account some late modification of requirements and is thus not well adapted to open physical complex systems which cannot, by definition, be completely a priori specified.

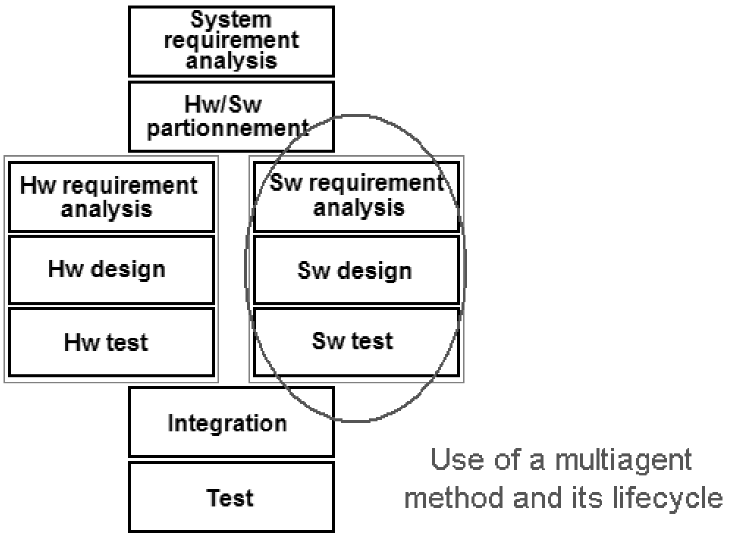


Figure 1. Traditional lifecycle

A few works deal with embedded multiagent systems, but new applications are strongly concerned by this domain (Pervasive computing [4] and industrial applications of MAS [18]). Even if we are at the beginning of the expansion of embedded multiagent systems, we are sure that embedded MAS methods will be the continuation of traditional embedded system design lifecycle (see fig. 1). Multiagent approaches focus on software parts and forget the hardware aspects. Hardware aspects are generaly taken into account only during the deployment step [5], and are limited to the choice of the plateform where the agents must be deployed. We can thus say that the hardware/software hybrid systems design is very partially covered by MAS methods. An alternative to this type of lifecycle is the codesign approach. A codesign method unifies the development of both hardware and software parts by the use of a unified formalism. The partitioning step is pushed back at the end of the life cycle. We can thus settle at this point of our study that the choice of a specific lifecycle model which support a codesign approach is required.

Because of the complex features of our system, the lifecycle model must enable late modification of specifications. Furthermore, it is necessary to come back on previous design steps (refinement) and to explore the space of solution of the hardware/software compromise. The design process must accept genericity (incremental criteria are in favour of the genericity). Finally, we must identify and keep a trace of all the parameters of the different retained solutions. The evaluation of different lifecycle models in respect with these previous criteria leads to adopt a spiral lifecycle [2].

Four main stages, distributed on a spiral cycle (fig 2), may be distinguished within DIAMOND, our physical multiagent design approach. The *requirements definition* precises what the user needs and characterizes the global functionalities. The second stage is a *multiagent-oriented analysis* which consists in decomposing a problem in a multiagent solution. The third stage of DIAMOND starts with a *generic design* which aims to build the multiagent system (once agents tasks have been defined) without distinguishing hardware/software parts. Finally, the *implementation* stage consists in partitioning the system in a hardware part and a software part to produce the code and the hardware synthesis.

A last and major difference between DIAMOND and other multiagents approach is, as said previously, that DIAMOND unifies the development of the hardware part and the software part. In fact, a hardware requirement and a software requirement are created from the system requirements. The software part of the system is built using a multiagent method and its associated lifecycle.

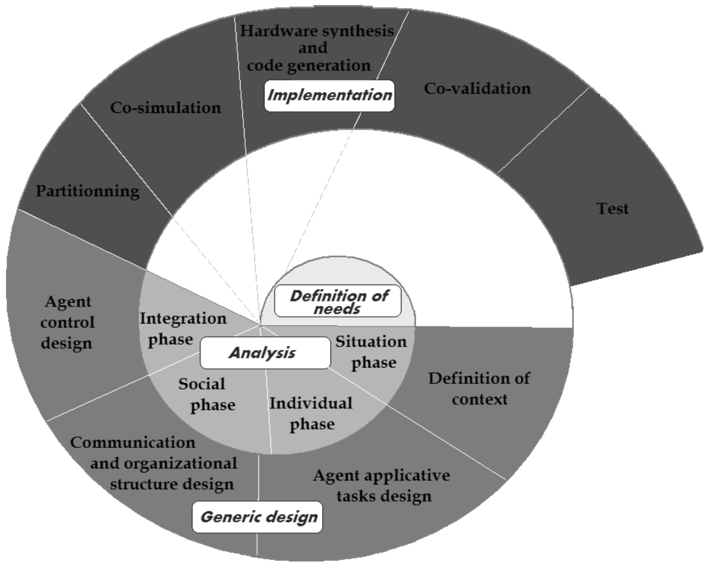


Figure 2. Our lifecyle

2 Requirements Definition

This preliminary stage starts by an analysis of the physical context of the system (identifying workflow, main tasks, etc...). Then, we study the different actors and their participative user cases (using UML use case diagrams), and the services requirements (using UML sequence diagrams) of these actors. The UML sequence diagram can include physical interaction.

The second step consists in an original contribution : the study of particular modes for a system that we call ”run mode” and ”stop mode”. It is generally wishable that the system works in autonomy. But working with physical systems requires to identify many particular possible behaviors : In which state must the system be when going under maintenance? how to calibrate the system components ? What must be the state of all the components when an emergency stop occurs (robot in safety area...)? Even in a decentralized intelligence context, the conditions defining these modes must remain easily understandable. The users of the system must respect laws and norms. These are very strong because the human safety can easily be altered in a physical context.

This activity puts forward a restricted running of the system. It allows to specify the first elements necessary for a minimal fault-tolerance. Moreover, this phase enables to identify cooperative (or not) situations and to define recognition states in order to analyze, for example, the self-organizational process of an application. This activity allows to take into account the safety of the physical integrity of the users possibly plunged in the physical system.

We have defined fifteen differents modes grouped in three families. The *stop modes* which are related to the different procedures for stopping and to define the associate recognition states. The *run modes* which focuses on the definition of the recognition states of normal running, test procedures etc. The *failure operations modes* which concentrates the security procedures (for example allowing a human maintenance team to work on the system) or to specify rules for restricted running etc.

3 Multiagent-oriented analysis

The multiagent stage is handled in a concurrent way at two different levels. At the society level, the multiagent system is considered as a whole. At the individual level, the system’s agents are built. This integrated multiagent design procedure encompasses five main phases discussed in the following.

Situation phase. The situation phase defines the overall setting, i.e., the environment, the agents, their roles and their contexts. This stems from the analysis stage. We first examine the environment boundaries, identify passive and active components and proceed to the problem agentification.

We insist here on some elements of reflexion about the characteristics of the environment [19, 20]. We must identify here what is relevant to take into account from the environment, in the resulting application.

It's, first of all, necessary to determine the environment *accessibility* degree i.e. what can be perceived from it. We will deduce from these characteristics which are the primitives of perception needed by agents. Measurements make possible to measure parameters which enable to recognize the state of the environment. They thus will condition the decisional aspect of the agent.The environment can be qualified of *Determinist* if it is predictible by an agent, starting from the environment current state and from the agent actions. The physical environment is seldom deterministic. Examining allowed actions can influence the agent effectors definition. The environment is *Episodic* if its next state does not depend on the actions carried out by the agents. Some parts of a physical environment are generally episodical. This characteristic has a direct influence on agent goals which aim to monitor the environment. Real environment is almost always *dynamic* but the designer is the single one able to appreciate the level of dynamicity of the part of the environment in which he is interested. This dynamicity parameter as an impact on the agent architecture. Physical environments may require reactive or hydrid architectures. The environment is *discret* if the number of possible actions and states reached by the environment are finite. This criterion is left to the designer appreciation according to the application it considers. A real environment is almost always continuous.

It is then necessary to identify the active and passive entities which make the system. These entities can be in interaction or be presented more simply as the constraints which modulate these interactions. It is necessary to specify the role of each entities in the system. This phase allows to identify the main entities that will be used and will become agents.

Individual phase. Decomposing the development process of an agent refers to the distinction made between the agent's external and internal aspects. The external aspect deals with the definition of the media linking the agent to the external world, i.e., what and how the agent can perceive, what it can communicate and according to which type of interactions, and how it can make use of them.

The agent's internal aspect consists in defining what is proper to the agent, i.e. what it can do (a list of actions) and what it knows (its representation of the agents, the environment, interaction and organization elements [6]).

In most cases, the actions are carried out according to the available data about the agent's representation of the environment. Such a representation based on expressed needs has to be specified during specifications of actions. In order to guarantee that the data handled are real data, it is necessary to define the required perception capabilities. We have defined four types of actions. *Primitive actions* are tasks which are not physicaly decomposable. *Composed actions* are temporal ordered lists of primitives. *Situated actions* need to have a world representation to execute their tasks.

Society phase. Interaction among agents are achieved via messages passing. Such exchange modes are formalized by means of interaction protocols. Although these interaction protocols are common to all the agents, they are rather external to them. Conflict resolution is efficiently handled by taking into account the relationships between the agents, that is, by building an explicit organizational structure. Such an organization is naturally modelled through subordination relations that express the priority of one agent on an other.

Integration phase. We need to analyze the possible influences upon the previous levels. Those influences are integrated within the agents by means of their communication and perception assessment capabilities (given in each agent's model through guard and trigger rules). The decomposition masks the notion of agent's control, i.e., how it handles its focus of attention, its decisions, and it links its actions. This dual aspect is based on the two previous one. Through the integration of social influences within the agents, one will endow the multiagent system with some dynamics. According to the social analysis we must give to the agent the possibility to interact in order to choose its role.

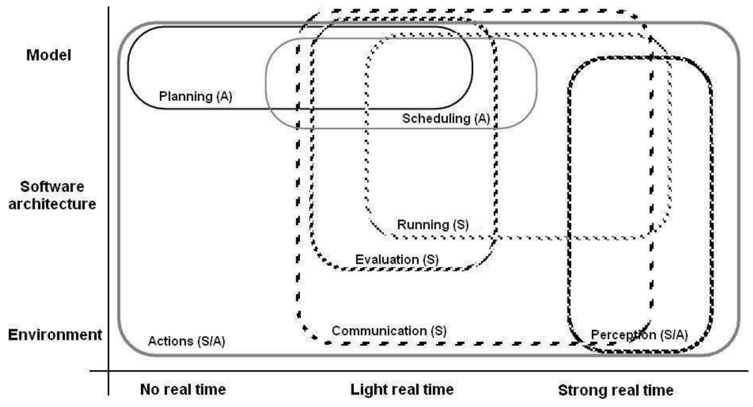
4 The Generic Design

This stage is based on an abstract component decomposition. We can define an abstract component as an elementary object, that performs a specific but reusable function. It is designed in such a way to easily operate with other components to create an application. Component can be combined with others to build more complex functions. This phase offers an efficient process leading to a component decomposition by starting from the informal description of the multiagent system built during the previous stage.

The Problem Description Phase. This phase consists in identifying and delimiting the domain of the general problem, as well as identifying some specific aspects that should be taken into account. Althrough this phase is informal, it allows designers to clearly separate the various aspects embedded within the application. We must choose here the architecture of the differents agents.

The agents are built following hybrid architectures, i.e. a composition of some pure types of architecture. Indeed, the agents will be of a cognitive type in case of a configuration alteration, it will be necessary for them to communicate and to manipulate their knowledge in order to have an efficient collaboration. On the other hand, in a normal mode use, it will be necessary for them to be reactive using a stimuli/response paradigm to be most efficient.

We evaluated in [15] the impact of the aspects time real on the design of the agents and shown that they must be taken into account for each abilities of the agents and at each level of the design as reminded on figure 3.



Recommended temporal mechanism: synchronous (S), asynchronous (A).

Figure 3. Distribution of the real time constraints processing

Agent applicative tasks design phase. We must build the external shell of the agent i.e. elaborating the interface with the external world for each sensors and effectors. It is time, here, to choose technological solution for them and to complete the context diagram to specify all information about the signal. The next step is to design the internal shell of the agent. We begin by the elaborated actions according to the task tree.

It is necessary at this stage to arrange the components to build the application: the architecture of the agent will be used as a pattern, at a very high level, for the components decomposition. The components have an external and an internal description. The internal description can be an assembly of components, or a formatted description of a decisional algorithm.

Each task is associated to a component as showed on fig. 4. At this level the designer has to build agents applicative tasks :

- Building of the external shell of the agent :
  - modules in charge of the acquisition of external information used to build the world representation (components C_{i_x}),
  - modules ensuring the actions on the environment to change its state (components C_{o_x}). Each interfaces between sensors/actuators and the core of the agent is a component able to sample and code an information.
- Building of the internal shell of the agent :
  - action modules of the agent connected through or ports to the external shell or other components(components C_{om_x}),
  - modules used to interprete perceptions (components C_{im_x}).
- Building communication modules and organisational structure composants (components C_{COM_i} and C_{COM_o}). In our context, interaction protocols (specified using AUML) are translated into finite state automatas. As we manipulate abstract components the advantage of these automatas is that they can generate either software or hardware descriptions.
- Building of the agent control : elaboration of the behavior of the components/agents by evaluation and decision components. When the interpretation of a message is transmitted through their ports to decision components, they can change values on ports and thus change component states. The whole agent state will so be changed.

It is important to notice that we exploit here the potential of the spiral lifecycle. The enrichment of components is made through the derivation of the results of the MAS iteration.

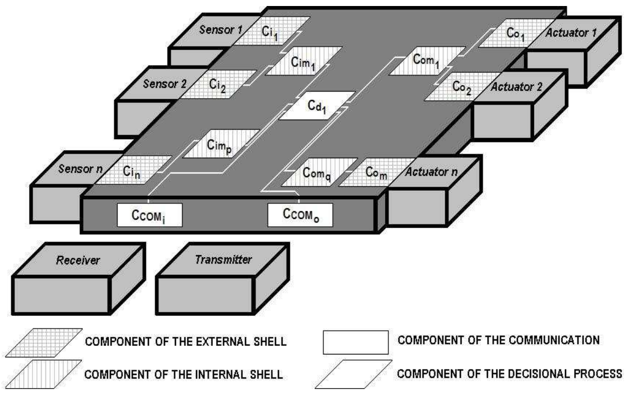

Sensor 1
Ci1
Cim1
Com1
Co1
Actuator 1
Sensor 2
Ci2
Co2
Actuator 2
Cd1
Cimp
Sensor n
Cin
Comq
Com
Actuator n
Ccomi
Ccomo
Receiver
Transmitter
COMPONENT OF THE EXTERNAL SHELL
COMPONENT OF THE INTERNAL SHELL
COMPONENT OF THE COMMUNICATION
COMPONENT OF THE DECISIONAL PROCESS

Figure 4. Building a composite agent

5 Implementation Stage

Partitioning Phase. The main use of codesign techniques appears in the software/hardware partitioning of the abstract components defined in the third level. Also it is essential to study the different partitioning criteria.

A first level relates to agent parts for which the partitioning question doesn't exist.Indeed some elements must be hardware as input/output periphericals such as for example the sensors and the actuators.

The second level relates to features for which there are several choices of implementation. We present below, those which can be considered to be relevant for the agents according to previous works we have made in this field [15, 11, 8] and codesign works like [1]:

The *cost* is present at all the stages of a system design life cycle. On very small series, we must decrease, as much as possible, the price of the software/hardware development and the hardware material. In the case of great series, we must reduce manufacturing costs.

The *performance* depends on the considered problem. A real-time application for which the robustness is a function of the occupation processor time is an example of system where this criterion is very important. A hardware partitioning is often privileged.

The *flexibility* plays in favor of the software. Software modifications have generally a less significant impact on the whole system than a hardware change. However, the flexibility of the EPLD (Electrical Programmable Logic Device) and other FPGA (Field Programmable Gate Array) increases quickly. For example, these architectures are reprogrammable in-situ : it is possible to modify their specifications without extracting them from the electronic chart.

From their nature, software systems are less *fault tolerant* than hardware components like EPLD. Indeed, microcontrolers use memories, stack structures with possible overflow etc. The *internal fault tolerance* will be thus a criterion which will play in favour of a hardware partitioning.

The *ergonomic contraints* gather all the system physical characteristics like weight, volume, power consumption, thermal release etc. Depending on the application, this criterion can be highly critical (case of the aeronautics embedded applications). One more time, the designer must appreciate correctly this criterion.

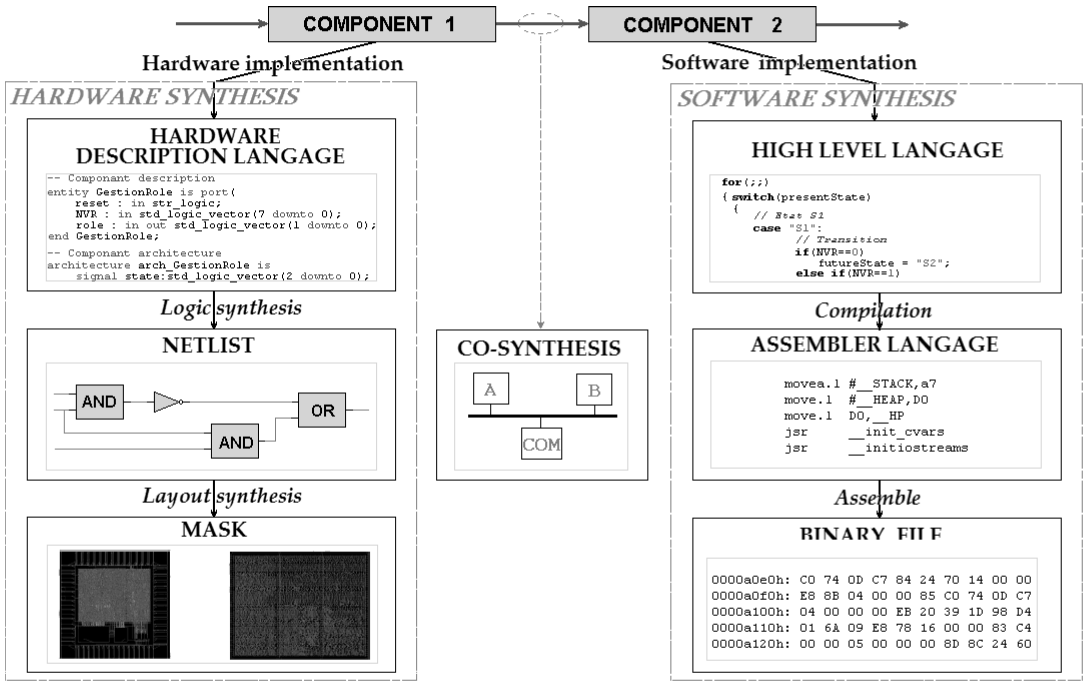
The Algorithmic complexity has a great importance for some applications. The software part will be more important if tasks are verz complex. In fact, it is very difficult to make hardware synthesis of highly cognitive features.

Co-simulation and co-validation Phases. This activity allows to simulate the collaboration between software part, hardware part and their interfaces. The type of tools used here come from the codesign communauty.

Implementation Phase. At this level, each components are completely specified with a common graphic specification formalism for the hardware part and the software part. For each component, the designer has already selected if he wishes a hardware or a software implementation.

This level must ensure the automatic generation of the code for the components for which an implementation software has been selected. The code is made in a portable language like Java or C++.

We use a Hardware Description Language which provides a formal or symbolic description of a component or of a hardware circuit and it interconnections. In our method the hardware components are specified in VHDL [3]. The compilation of the code and the hardware synthesis of different specifications in VHDL are carried out like illustrated on figure 5.



```
-- Componant description
entity GestionRole is port(
    reset : in str_logic;
    NVR : in std_logic_vector(7 downto 0);
    role : in out std_logic_vector(1 downto 0);
end GestionRole;
-- Componant architecture
architecture arch_GestionRole is
    signal state:std_logic_vector(2 downto 0);
```

```
for(;;)
{ switch(presentState)
  {
     // Etat S1
     case "S1":
           // Transition
           if(NVR==0)
              futureState = "S2";
           else if(NVR==1)
```

```
movea.l  #__STACK,a7
move.l   #__HEAP,D0
move.l   D0,__HP
jsr      __init_cvars
jsr      __initiostreams
```

```
0000a0e0h: C0 74 0D C7 84 24 70 14 00 00
0000a0f0h: E8 8B 04 00 00 85 C0 74 0D C7
0000a100h: 04 00 00 00 EB 20 39 1D 98 D4
0000a110h: 01 6A 09 E8 78 16 00 00 83 C4
0000a120h: 00 00 05 00 00 00 8D 8C 24 60
```

Figure 5. Software component synthesis and hardware component synthesis

Hardware/Software agent society simulation.

To decrease difficulties of the deployment of embedded multiagent systems, a new kind of simulation are required: the hw/sw hybrid simulation. Traditional software simulation approach focuses only on the MAS design and not on its implementation on a specific platform of the real world. Hw/Sw ybrid simulation allows verifying that there is no deviation between the behavior of the real embedded implementation of the MAS and the designed software part behavior of the MAS.

We discuss about our tool called MASH in the next section.

6 Hardware/Software MAS simulation : the MASH simulator

6.1 Overview

Our hw/sw simulator of several simulated/real agent nodes which interacting with an environment component (SimNetwork) via an abstraction layer (SimAgent). The basic architecture of our simulator and its major simulation models are described on the figure 6. It allows to simulate the MAS in three different ways: the software simulation, the hardware simulation and the hw/sw hybrid simulation.

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... A 2-dimensional grid models it. The *wave propagation model* is implemented like circular wave propagation through the 2-dimensional grid. It estimates the received power measured by a receiver agent Pr

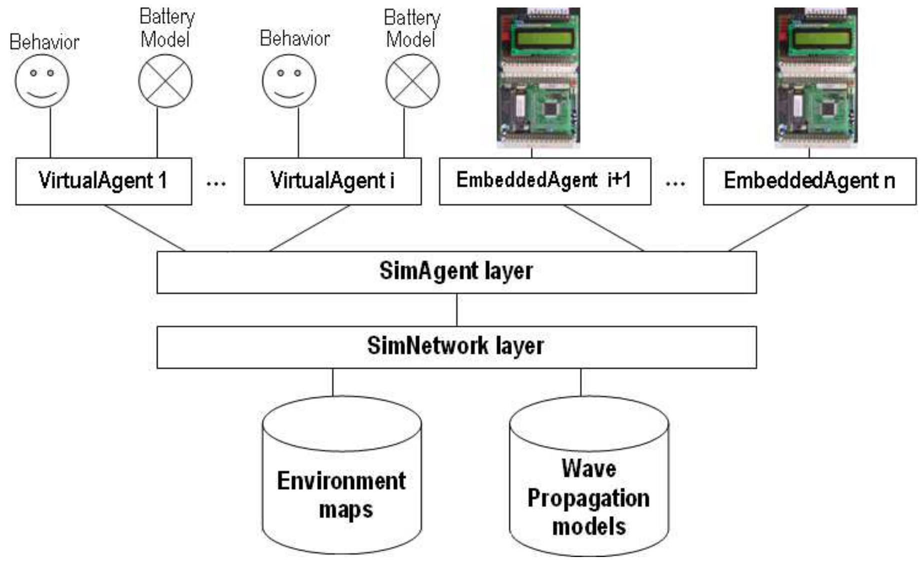


Figure 6. The MASH simulator architecture

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 Pr 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...) during its propagation. Our tools implement the Friis free space transmission equation.

SimAgent layer This layer enables the simulation of the hw/sw 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.

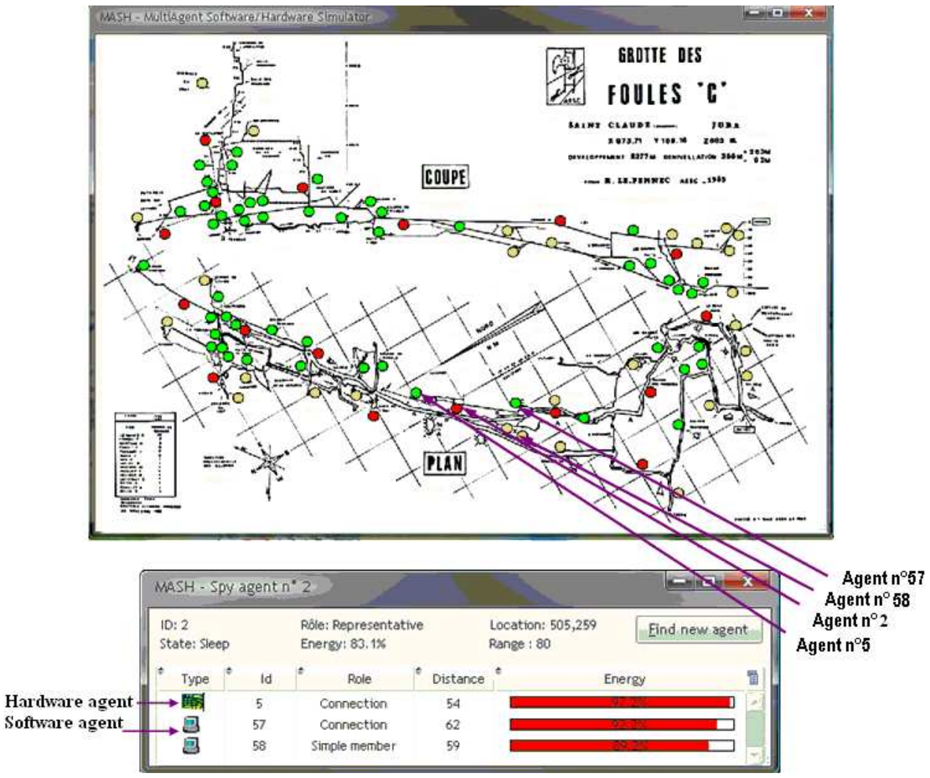
Behavior component The behavior component is the applicative component. It simulates the execution of software on a single sensor node. It receives messages from the other agent. The SimNetwork answers physical component requests like reading sensor values or controlling actuators. The simulator user 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 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. 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 models of battery: the linear model. Other models are described in [17, 7]. In fact, in the case of our eMAS simulation, we do not need the same precision that the one required for the hw/sw partitioning for example, where the aim is to find the best hw/sw 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 dependent model, relaxation model...). The model that we have implemented in our simulator defines the battery as a linear storage of current. 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.

6.2 Discussion about an experience

We discuss in this part about the three different types of simulation in evolved in the MASH simulator. Our simulator allows to evaluate eMAS by comparing our multiagent approach with others solutions because it provides a scenario editor/player. The figure 7 shows two windows of a hw/sw joint simulation that evolves 95 agents: 91 software agents and 4 hardware agents.



ID: 2
State: Sleep
Rôle: Representative
Energy: 83.1%
Location: 505,259
Range : 80

Type	Id	Role	Distance	Energy
	5	Connection	54	[illegible]
	57	Connection	62	[illegible]
	58	Simple member	59	[illegible]

Figure 7. Screen copy of the simulator

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. 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 MAS. However, a major problem of these software simulations is that the quality of the simulation depends on the quality of the different evolved models. The embedded multiagent simulation integrates many models that come from non-multiagent community. These models concern the environment [14], the wireless channels [13], the battery devices [7] ...

Hybrid software/hardware simulation. One of the hw/sw 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...). Another advantage of the hw/sw 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 programs 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.

In figure 8, we can see 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 have chosen an unexpected role (simple member) because the time delay to analyze all the received messages.

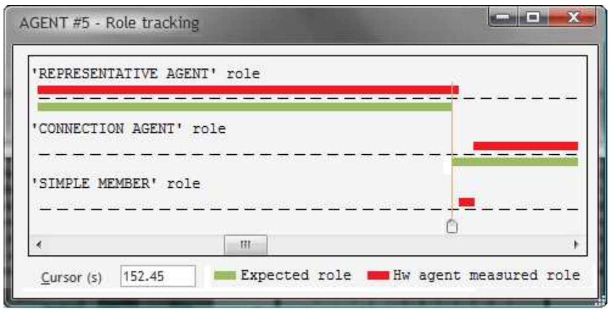


Figure 8. Role tracking of an hardware agent

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... In a debugging phase, it is possible to debug the agent with a serial debugging backchannel (independently from MASH). 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, many devices are required. The associated financial cost is important.

Conclusion

The DIAMOND Method has been validated by the design of several applications as the EnvSys project (embedded sensor network for the instrumentation of an underground hydrographic system) [12], a UWB positionning system [16], the PALETTE Projet (Application of Collective Robotics to paletization in manufacturing). This project funded by the Rhone-Alpes regional council program for technological transfert.

We work currently on the tool associated with the method that we propose. It is created using the Java language. The part which relates to the creation of agents with components, manual partitioning and automatic generation of code are operationnal.

Our future work concerns the MASC tools (MultiAgent System Codesign) associated with the DIAMOND method. The agent design with components and the code generation in Java and C langages are operational. The VDHL specification generation is partially developped.

Very few works are addressing the problem of the analysis of self-organized embedded systems. This work proposes some innovative contributions in term of hybrid software/hardware multiagent lifecycle. It integrates in particular all the phases of the development from the analysis to the implementation. It introduces a multi-paradigm spiral lifecycle. It proposes components used as tools for integration, allowing software or hardware derivation. They enable a unified approach for all kinds of hybrid harware/software multiagent systems.

References

[1] J. Adams and D. Thomas. The design of mixed hardware/software systems. Las Vegas, USA, june 1996. ACM.

[2] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72, 1988.

[3] P. T. Breuer, N. M. Madrid, J. P. Bowen, R. B. France, M. M. Larrondo-Petrie, and C. D. Kloos. Reasoning about vhdl and vhdl-ams using denotational semantics. pages 346–352, Munich, Germany, March 1999. ACM.

[4] C. Carabelea, O. Boissier, and F. Ramparany. Benefits and requirements of using multi-agent systems on smart devices. In H. Kosch, L. Böszörményi, and H. Hellwagner, editors, *Euro-Par*, volume LNCS n2790, pages 1091–1098. Springer, 2003.

[5] M. Cossentino, L. Sabatucci, and A. Chella. A possible approach to the development of robotic multi-agent systems. In *IEEE/WIC Conf. on Intelligent Agent Technology (IAT'03)*, pages 539–544, Halifax (Canada), 2003.

[6] Y. Demazeau. From interactions to collective behavior in agent-based systems. In *European Conference on Cognitive Science*, France, 1995.

[7] 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, Sep. 2003.

[8] J.-P. Jamont and M. Occello. A self-organized energetic constraints based approach for modelling communication in wireless systems. In *Advances in Applied Artificial Intelligence*, volume LNAI N4031, pages 101–110. Springer-Verlag, 2006.

[9] J.-P. Jamont and M. Occello. About some specificities of embedded multiagent systems design. In *Prooceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 51–54. IEEE Computer Society, 2007.

[10] J.-P. Jamont and M. Occello. Designing embedded collective systems: The diamond multiagent method. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 2. IEEE Computer society, 2007.

[11] J.-P. Jamont, M. Occello, and A. Lagreze. A multiagent system for the instrumentation of an underground hydrographic system. In *Proceedings of IEEE International Symposium on Virtual and Intelligent Measurement Systems*, Mt Alyeska Resort, USA, May 2002. IEEE Measurement and Instrumentation Society.

[12] J.-P. Jamont, M. Occello, and A. Lagrze. A multiagent approach to manage communication in wireless instrumentation systems. *Measurement*, 43(4):489–503, 2010.

[13] G. Judd and P. Steenkiste. A software architecture for physical layerwireless network emulation. In *Proocedings of the First ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and CHaracterization*, pages 97–107. ACM, September 2006.

[14] 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.

[15] M. Occello, Y. Demazeau, and C. Baeijs. Designing organized agents for cooperation in a real time context. In A. Drogoul, M. Tambe, and J. Singh, editors, *Collective Robotics*, volume LNAI 1456, pages 25–73. Springer-Verlag, March 1998.

[16] M. Occello, J.-P. Jamont, R. Guillermin, and M. Pezzin. A multiagent approach for an uwb location embedded software architecture. In *Proceedings of the 5th International Conference on Soft Computing as Transdisciplinary Science and Technology*, pages 279–285. ACM, 2008.

[17] 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.

[18] H. V. D. Parunak. A practitioners? review of industrial agent applications. *Autonomous Agents and Multi-Agent Systems*, 3(4):389–407, 2000.

[19] S. Russel and P. Norvig. *Artificial Intelligence : a Modern Approach*. Prantice-Hall, 1995.

[20] M. Wooldridge et al. The gaia methodology for agent-oriented analysis and design. In *Journal of Autonomous Agents and Multi-Agent Systems*, volume 3. Kluwer Academic Publishers, 2000.