About some specificities of embedded multiagent system design

Jean-Paul Jamont, Michel Occello

About some specificities of embedded multiagent systems design

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

Abstract

Multiagent systems (MAS) satisfy to design requirements for open physical complex systems. However, up to now, no method allows to build software/hardware hybrid multiagent systems : we introduce the DIAMOND method.

1 Introduction

Control systems, processing systems, communication systems and interactive systems can be seen as OPCS (open physical complex systems) because they are supported by new wireless technologies, they are more and more distributed and decentralized. They involve numerous software/hardware entities which enable logical/physical interactions between them and their shared environment. Our work deals with the modelling and the design of OPCS using MAS, arguing that two types of needs emerges in this specific context : needs concerning specific system architectures (our contribution is the MWAC model (Multi-Wireless-Agent Communication) based on our previous work on wireless sensor networks [4]) 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). We try to answer to some questions asked by this kind of applications all along the lifecycle and in the choice of formalisms.

2 The approach

A few works deal with embedded MAS, but new applications are strongly concerned by this domain as pervasive computing or industrial applications of MAS [8]. Even if we are at the beginning of the expansion of embedded MAS, we are sure that embedded MAS methods will be the continuation of traditional embedded system design lifecycle. Multiagent approaches focus on software parts and forget the hardware aspects. These aspects are generally taken into account only during the deployment step [3], and are limited to the choice of the plateform where the agents must be deployed. The hardware/software hybrid systems design is thus 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 lifecycle. We can thus settle that the choice of a specific lifecycle model which support a codesign approach [1] is required.

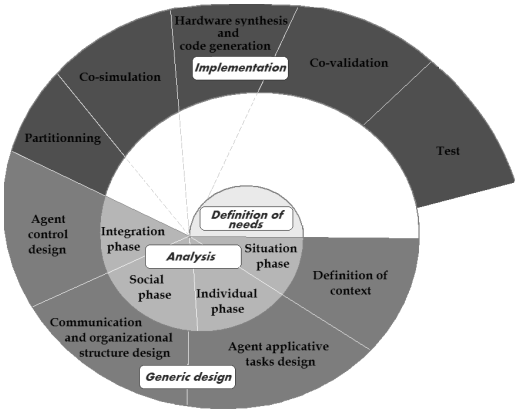

Figure 1. Our lifecyle

Because of the complex features of our systems, 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 1),

may be distinguished within DIAMOND. 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 MAS (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 code and hardware synthesis.

3 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 "running modes" and "stop modes". 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 entities? What must be the state of all the entities 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. 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 *running modes* which focus on the definition of the recognition states of normal running, test procedures etc. The *failure operations modes* which concentrate the security procedures (for example allowing to a human maintenance team to work on the system) or to specify rules for restricted running).

4 Multiagent-oriented analysis

The multiagent stage is handled in a concurrent way at two different levels. At the society level, the MAS 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. This 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 [7]. 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. A 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. A 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. A real environment is almost always continuous.

It is then necessary to identify active and passive entities which constitute the system. They 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 entity 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).

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 physically 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 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 MAS with some dynamics. According to the social analysis we must give to the agent the possibility to interact in order to choose its role.

5 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 MAS 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. Although 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 different 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 [5] the impact of the real time aspects on the design of the agents and shown that they must be taken into account for each ability of the agents and at each level of the design as reminded on figure 2.

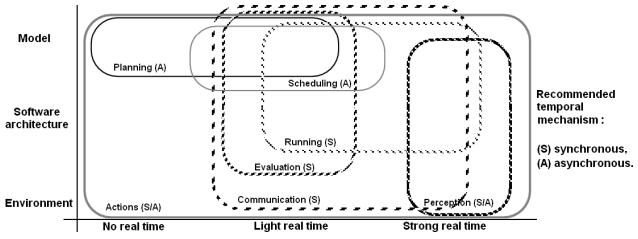

Figure 2. Distribution of the real time constraints processing

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 sensor and effector. It is time, here, to choose a 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 agents. 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 figure 3. At this level the designer has to build agents applicative tasks :

• Building of the external shell of agents : modules in charge of the acquisition of external information used to build the world representation (C_{i_x}) and modules ensuring the actions on the environment to change their state (C_{o_x}).

• Building of the internal shell of agents : action modules of agents connected through ports to the external shell or to other components (C_{om_x}), and modules used to interprete perceptions (C_{im_x}).

• Building communication modules and organisational structure composants (C_{COM_i} and C_{COM_o}). In our context, interaction protocols are translated into FSM (they can easily generate either software or hardware descriptions).

• Building of agents'control : elaboration of the behavior of the components/agents by evaluation and decision components. Interpretation of a messages are transmitted

through ports to decision components. Values of ports can then change and thus change component states. Whole agents'states will so be changed.

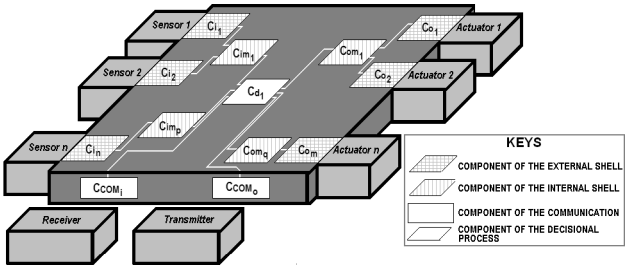

Figure 3. Building a composite agent

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.

6 Implementation Stage

The main use of codesign techniques appears in the software/hardware **Partitioning Phase** 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's parts for which the partitioning question doesn't exist. Indeed some elements must be hardware as input/output peripherals like the sensors and the actuators. The second level relates to features for which there are several choices of implementation. Following criteria can be considered to be relevant for the agents according to our previous works in this field [5, 4] and codesign works like [1]: the cost, the performance, the flexibility, the fault tolerance, the ergonomic contraints and the algorithmic complexity.

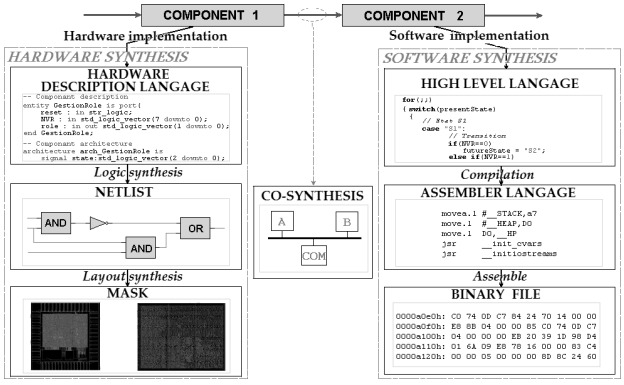

Figure 4. Software component synthesis and hardware component synthesis

Co-simulation and co-validation Phases are then necessary to simulate the collaboration between software part, hardware part and their interface.

At the **Implementation Phase**, each component is 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 a symbolic description of a hardware circuit and it interconnections. In our method hardware components are specified in VHDL [6]. The compilation of the code and the hardware synthesis of different specifications in VHDL are carried out like illustrated on fig. 4.

7 Conclusion

The DIAMOND Method has been validated by the design of several applications as the EnvSys project (a sensor network for the instrumentation of an underground hydrographic system) [4] or the PALETTE Project (application of collective robotics to paletization in manufacturing).

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.

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] M. Cossentino et al. A possible approach to the development of robotic multi-agent systems. In *Conference on Intelligent Agent Technology*, pages 539–544, Halifax, 2003.

[4] 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 4031, pages 101–110. Springer, 2006.

[5] M. Occello et al. Designing organized agents for cooperation in a real time context. In *Collective Robotics*, volume LNAI 1456, pages 25–73. Springer, March 1998.

[6] V. A. Pedroni. *Circuit Design With VHDL*. MIT Press, 2004.

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

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