DESIGNING COOPERATIVE EMBEDDED SYSTEMS USING A MULTIAGENT APPROACH : THE DIAMOND METHOD

Jean-Paul and Michel Occello
Université Pierre-Mendès France, Laboratoire LCIS/INPG, F-26000 Valence, France

Abstract: Multiagent systems are well suited to specify requirements for open physical complex systems. However, up to now, no method allows to build software/hardware hybrid multiagent systems. This paper presents an original method for designing physical multiagent systems. It advocates a basic multiagent phase able to tackle functionnal and organizational issues, associated to a componential phase for the detailed design making easier the software/hardware partitionment.

Key words: Multiagent oriented method, embedded multiagent system, codesign.

1. INTRODUCTION

Complex artificial cooperative physical systems are involved in application domains as pervasive computing, intelligent distributed control or wireless computing. Physical systems have a physical reality which does not apply only to the entities but also to the environment in which they evolve. The system and its environment are strongly related. In this context, the system elements integrate generally a software part and a hardware part (electronic cards, sensors, effectors). The high dynamics, the great heterogeneity of elements and the openess make a multiagent approach highly profitable for these artificial complex systems. But the existing multiagent design lifecycles have to be modified to take into account software/hardware hybridation particularities.

This paper aims to present our approach called DIAMOND (Decentralized Iterative Multiagent Open Networks Design) for the design of open multiagent physical complex systems.

Our method can be qualified of codesign because it unifies the development of the hardware part and the software part : the partitioning step is pushed back at the end of the life cycle. A multiagent phase allows the management of collective features. A component phase is used to design the elementary entities of the system (the agents) and to facilitate the hardware/software partitioning.

2. OVERVIEW OF THE DIAMOND METHOD

The DIAMOND method is built to design physical multiagent system. Four main stages, distributed on a spiral cycle (fig. 1), may be distinguished within our physical multiagent design approach. The *definition of needs* defines 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 our method starts with a *generic design* which aims to build the multiagent system, once one knows what agents have to do without distinguishing hardware/software parts. Finaly, the *implementation* stage consists in partitioning the system in a hardware part and a software part to produce the code and the hardware synthesis.

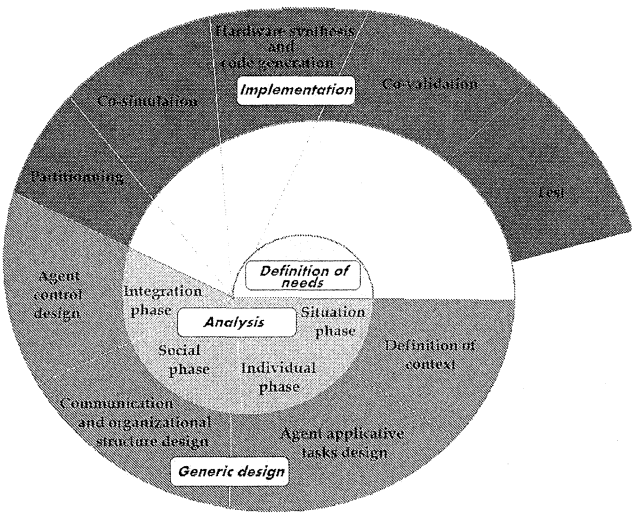

Figure 1. Our lifecyle

Most existing multiagent methods usually distinguish only analysis and design phases [2]. Very few methods deal with other phases. We can find for example a deployment phase in MASSIVE [8] or Vowels [10]. This deployment phase takes in our particular field a great importance since it includes the hardware/software partitioning. To cover the whole lifecycle, different formalisms are required to express different things at different levels [5], for this reason we adopt a cycle using four stages mixing different expressions using more or less formal paradigms and languages (agents, components, Finite State Machines, Hardware Definition Languages). The most current lifecycle used in multiagent methods is the classical cascade cycle. Even if some works attempt to introduce iterative cycle as Cassiopeia (W) [4] or Gaia [12], the proposal of a spiral life cycle is very original.

3. DEFINITION OF NEEDS

This preliminary stage begins by analysing 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), the services requirements (using UML sequence diagram) of these actors.

The second step consists in the study of the different modes of steps and stop. This activity is very significant because it will make it possible to structure the global running of the system. It is generally wishable that the system functions in autonomy. But working with physical systems imposes to know all the other possible behaviors precisely when the system starts, when it goes under maintenance, when we want to stop it.

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

We have defined fifteen different modes that we regroup in three families. The *stop modes* which are related to the different procedures for stopping (partially or completely) and to define associate recognition states. The *steps modes* which focus on the definition of the recognition states of normal functionning, tests procedure etc. The *failing operations modes* which concentrate to the procedure allowing to a human maintenance team to work in the system or to specify rules for degraded running.

4. MULTIAGENT-ORIENTED ANALYSIS

The multiagent stage is handled in a concurrent manner 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 build. 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 component and proceed to the agentification of the problem.

We insist here on some elements of reflexion about the characteristics of the environment [11,12]. 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 which are the primitives of perception needed by agents. Measurements make possible to recognize states of the environment to interprete the state of the environment. They thus will condition the agent decisional aspect.

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 hydride architectures.

The environment is *discret* if the number of possible actions and states reached by the environment are finished. 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. For each entities it is

necessary to specify the role of these entities in the system. This phase makes it possible to identify the key components that we will use. Some of these active entities 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. Its 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, the interaction and the organization elements [3].

In most cases, the actions are carried out according to the avaiable 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 actual ones, it is necessary to define the useful perception capabilities.

We can take to illustrate these phases the the EnvSys application described in [6]. Capabilities can be specified using a tree to show the different nested levels (see fig. 2). We specify the agent context with a context diagram (see fig. 2}.

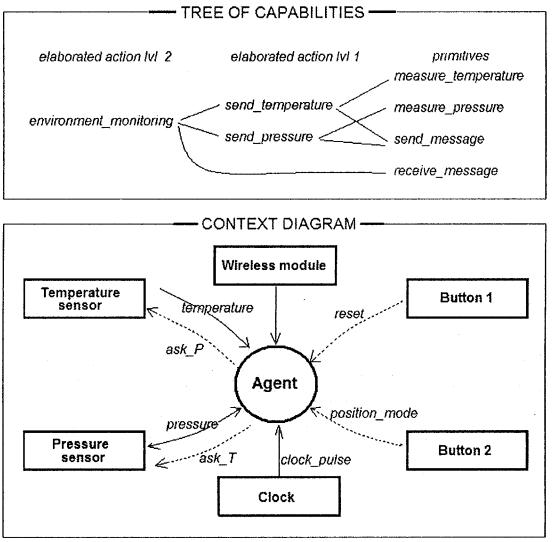

Figure 2. Primitives and actions scheme / Context diagram

Society phase. Interaction among agents are achieved via message passing. Such exchange modes are formalized by means of interaction protocols based on speech acts. Although common to all the agents, these interaction protocols are rather external to them.

Conflict resolution is efficiently handled by taking into account the relationships between the agents, that is, by constructing an explicit organizational structure.

Such an organization is naturally modeled through subordination relations that express the priority of one agent on another.

The main property of our organization is that it will be dynamic. In this kind of application no one can control a priori the organization. Relations between agents are going to emerge from the evolution of the agents'states and from their interactions. We are only going to fix the organization parameters, i.e. agents'tasks, agents'roles.

Integration phase. We need to analyse 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). The decomposition masks the notion of agent's control, i.e., how it handles its focus of attention, its decisions, and link its actions. This dual aspect is based on the two previous ones. Via 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.

5. THE GENERIC DESIGN

This stage is based on a component decomposition. We can easyly define component models as an architecture and an API that allows developers to define reusable segments of code that can be combined to create an application. So, a component is a reusable program building block, which is an identifiable part of a larger program. 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 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 normal use it will be necessary for them to be reactive (stimuli/response paradigm) to be most efficient.

Agent applicative tasks design phase. We must construct the external shell of the agent i.e. elaborate the interface with the external world for each sensor and effector. It is time, here, to choose technological solution for them and 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 for 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 component decomposition.

The components have an external and an internal description. Internal description can be an assembly of components or a formalism to describe a decisional.

6. IMPLEMENTATION STAGE

Partitionning Phase. The main use of the codesign technique appears in the software/hardware partitioning of the 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 i.e. sensors and effectors. Indeed some elements must be hardware as input/output peripherals 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 work we have made in this field [9,7] and codesign work:

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 contrainsts* 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 very 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 there interface.

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 (see fig. 3). The code is made in a portable language like Java or C++.

We use a Hardware Description Language which provides a formal or symbolic the components of a hardware circuit and it interconnections. In our method the hardware components is specified in VHDL [1]. The compilation of the codes and the hardware synthesis of the different VHDL specifications are carried out like illustrated on figure 3.

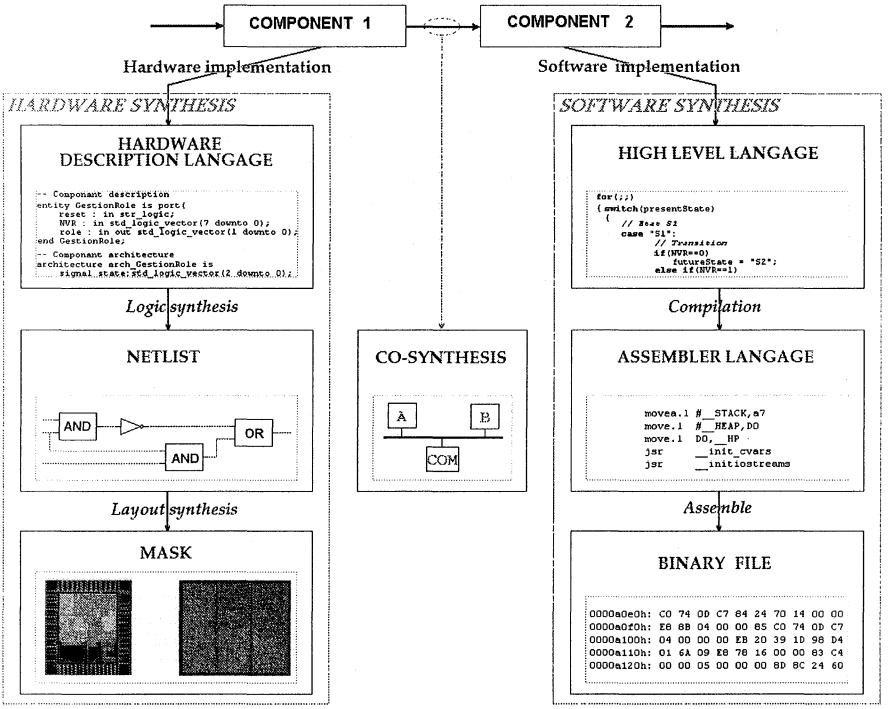

Figure 3. Software component synthesis and hardware component synthesis

7. CONCLUSION

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 creation with components, manual partitioning and automatic generation of code are operationnal.

This work proposes some innovative contributions in term of hybrid software/physical multiagent life cycle. 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. Components are thus used in this approach as units of implementation but further as unit of design allowing the assembly.

REFERENCES

[1] Breuer, P. T., Madrid, N. M., Bowen, J. P., France, R. B., Larrondo-Petrie, M. M., and Kloos,. D. (1999). Reasoning about vhdl and vhdl-ams using denotational semantics. In DATE, pages 346-352.

[2] DeLoach, S. A., Wood, M. F., and Sparkman, C. H. (2001). Multiagent systems engineering. International Journal of Software Engineering and Knowledge Engineering, 11(3):231-258.

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

[4] Drogoul, A. and Collinot, A. (1998). Applying an agent oriented methodology to the design of articial organizations: A case study in robotic soccer. Autonomous Agents and Multi-Agent Systems, 1(1):113-129.

[5] Herlea, D. E., Jonker, C. M., Treur, J., and Wijngaards, N. J. E. (1999). Specication of bahavioural requirements within compositional multi-agent system design. In MAAMAW, pages 8-27.

[6] J.P. Jamont and M. Occello (2004). Using organisational structures emergence for maintaining functional integrity in embedded systems networks. In Proceedings of IFIP Working Conference on Artificial Intelligence Applications and Innovations, pages 197-210, Kluwer Academic Publisher.

[7] J.P. Jamont and Occello, M. and A. Lagreze (2002). 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, AK, USA.

[8] Lind, J. (2001). Interative Software Engineering for multiagent systems: The MASSIVE Method, volume 1994 of LNCS/LNAI. Springer Verlag.

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

[10] Ricordel, P.-M. and Demazeau, Y. (2000). From analysis to deployment: Amulti-agent platform survey. In Proceedings of the First International Workshop on Engineering Societies in the Agent World, pages 93-105, London, UK. Springer-Verlag.

[11] Russel, S. and Norvig, P. (1995). Articial Intelligence : a Modern Approach. Prantice-Hall.

[12] Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems, 3(3):285-312.