Designing embedded collective systems: The DIAMOND multiagent method

Jean-Paul Jamont, Michel Occello

Designing embedded collective systems: The DIAMOND multiagent method

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

Abstract

Multiagent systems (MAS) are well suited to specify requirements for open physical complex systems. However, up to now, no method allows to actually build software/hardware hybrid MAS. This paper presents an original method for designing embedded MAS.

1 Introduction

Complex artificial cooperative physical systems are involved in application domains as pervasive computing, intelligent distributed control or wireless computing. Embedded 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 elements of the system integrate 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. Studying the design of embedded systems using multiagent paradigms is a recent research field. This paper aims to present our approach called DIAMOND (Decentralized Iterative Multiagent Open Networks Design) for the design of embedded complex systems with MAS.

2 Overview of the DIAMOND method

The DIAMOND method can be qualified of codesign because it unifies the development of the hardware part and the software part: the partitioning step is sent 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. In a traditional system design, the partitioning step takes place at the beginning of the cycle. In fact, a hardware requirement and a software requirement are created from the system requirements. In the case of a MAS, the software part of the system is built using a multiagent method and its associated lifecycle. In the case of DIAMOND, the partioning step is pushed back at the end of the lifecycle to authorize modifications of the requirement, reffinement and iteration.

Four main stages, distributed on a spiral shaped lifecycle (fig 1), may be distinguished within our embedded multiagent design approach. The *definition of needs* defines what the user needs and characterizes 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 MAS without distinguisting 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.

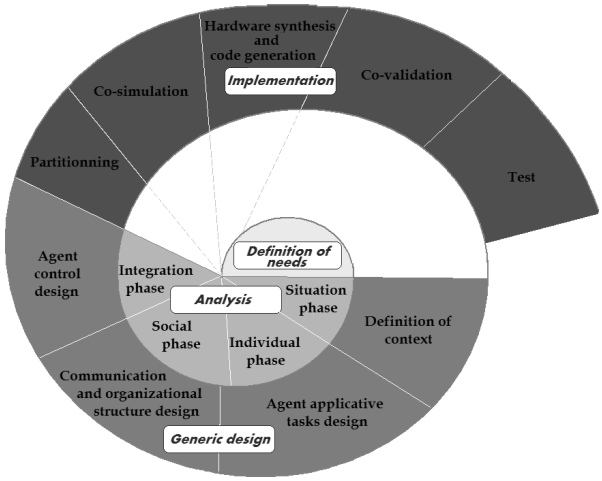

Figure 1. Lifecyle of the DIAMOND method

3 Definition of needs

This preliminary stage begins by analysing the physical context of the system (identifying workflow, main tasks, etc...). We then study the different actors and their participative user cases (using UML use case diagrams) as well as the services requirements (using UML sequence diagram) of these actors.

The second step consists in the study of the different modes of running and stops. This activity is very significant because it enable to structure the global running of the system. It is generally wishable that the system functions in autonomy. But working with embedded 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 restricted procedures for the system. It allows to specify the first elements necessary for a minimal fault-tolerance. Moreover, it enable to identify cooperative or non-cooperative situations [2] and to define recognition states in order to analyse, for example, the self-organizational process of an application. This activity allows to take into account the safety of the users plunged in the physical system as a human agent.

We have defined fifteen different modes that we regroup in three families. The *stops modes* which are related to the different procedures for stopping (partially or completely) and to define associate recognition states. The *running modes* which focuses on the definition of criteria or test procedures enabling to recognize states of normal functionning. The *failures operations modes* which concentrates the procedure allowing to a human maintenance team to work on the system or to specify rules for restricted mode.

4 Multiagent-oriented analysis

The multiagent stage is handled in a concurrent manner 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 MAS design procedure encompasses five main phases discussed in the following.

A *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 we proceed to the problem agentification.

We insist here on some elements of reflexion about the characteristics of the environment [10, 11]. We must identify here what is relevant to take into account from the environment, in the resulting application.

It is then necessary to identify 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.

In an *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. We specify the agent context with a context diagram (see 2).

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 physicaly decomposable. *Composed actions* are temporal ordered lists of primitives. *Situated actions* need to have a world representation to execute their tasks.

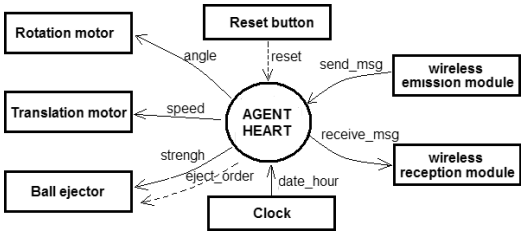

Figure 2. Context diagram

Interaction among agents are achieved via messages passing. Such exchange modes are formalized by means of interaction protocols in a *Society phase*. 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.

In an *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 through guard/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. We evaluated in [9] 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 with each level of the design.

5 The Generic Design

This stage is based on a component decomposition. We can define component as an elementary object, that performs a specific function that allows developpers to define reusable segments of code. It is designed in such a way to easily operate with other components to create an application. A component is a reusable program building blocks, 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 MAS built during the previous stage.

The *Problem Description 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. This informal phase allows designers to clearly separate the various aspects embedded within the application. We must choose here the different agents architecture. 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.
Using a hybrid architecture for the agents enables to combine the strong features of each of reactive and cognitive capabilities seen before. We use our ASTRO hybrid architecture [9], especially adapted to a real time context.

In an *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.

6 Implementation Stage

The main use of codesign techniques 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.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 can say (according to previous works we have made in this field [9, 7] and codesigns work like [1]) that criteria as *cost*, *performance*, *ergonomic contrainsts*, *algorithmic complexity* and *flexibility* can be considered to be relevant for the agents . *Co-simulation and co-validation Phases* allows then to simulate the collaboration between software part, hardware part and their interface.

Finally, at the *Implementation Phase*, 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. The compilation of the code and the hardware synthesis of different specifications in VHDL are carried out.

7 Discussion about DIAMOND

Lifecycle and phases. Most existing multiagent methods usually distinguish only analysis and design phases [5]. Very few methods deal with other phases. We can find for example a deployment phase in MASSIVE [8]. This deployment phase takes in our particular field a great importance since it includes the hardware/software partitioning. A major difference between DIAMOND and other multiagent approaches is, as said previously, that DIAMOND unifies the development of the hardware part and the software part. In a traditional system design, the partitioning step stands at the beginning. In fact, a hardware requirement and a software requirement are created from the system requirement. The software part of the system is built using a multiagent method and its associated lifecycle.

To cover the whole lifecycle, different formalisms are required to express different things at different levels [6]. For this reason we adopt a lifecycle using four stages mixing different expressions using more or less formal paradigms and languages (agents, components, FSM, Hardware Definition Languages). The most current lifecycle used in multiagent methods is the classical cascade lifecycle. Even if some works attempt to introduce iterative cycles as Gaia [11], the proposal of a spiral lifecycle is very original.

In the definition of needs phase, we introduce a study of the modes of running and stops to structure the global running of the system. In the generic design phase, the design allows an abstraction of the software design and the hardware design. We use components to build the agents as few multiagent methods introducing an actual componential dimension [8, 3]. These components are used to simplify the work of the designer through visual programming, to manage the complexity through a functionnal decomposition, to increase the genericity through reusability, to simplify the partitioning because the analogy beetween soft components and chips enables the hardware tools and the software tools to share a unified vision.

Models and notations. Multiagent methods generaly use notations and models from only one origin [2] like UML (Mase, AAII, MESSAGE, PASSI). Other methods use many notation like TROPOS [4] (notation i* coming from the knowledge enginering, A-UML for interaction protocols and plan) or DESIRE (graph-based notation for knowledge modelling and specific hierarchical notation for tasks description). To cover all the phases of a lifecycle, we think like in [6] that several formalisms are necessary for the different levels of abstraction.

DIAMOND begins by using UML use cases because they proved reliable for the definition of needs. The interpretation of our use case diagrams are slightly different than their common use (as in [2]) because actors are necessarily outdoor to the system or its entities. Moreover, an actor can not be in the interaction diagram (this would be amazing in a traditional use of UML use cases) in the case of physical interactions. These differences come from the usual software nature of applications.

In the analysis phase, we use context diagrams. These diagrams enable to see easily all the possible perception and the possible action of the agents. Another advantage is that they allow to see control flow between the physical part of an agent and its decisonnal part. In a word, context diagram allow to specify the external shell of the agents.

In the generic design phase, DIAMOND uses component as operational units as seen previously. In these components, we use FSM or a components set to describe the internal running. These formalisms enable to generate software code or hardware specifications in VHDL.

8 Conclusion

Our method has been validated on several real world projects as for an underground river instrumentation [7], an application of collective robotics to paletization in a manufacturing process or to build the software infrastructure for UWB sensor localization.

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 hardware/software MAS.

References

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

[2] C. Bernon, M.-P. Gleizes, S. Peyruqueou, and G. Picard. Adelfe: A methodology for adaptive multi-agent systems engineering. In *Third International Workshop on Engineering Societies in the Agents World*, volume LNCS N2577, pages 156–169, Spain, September 2002. Springer.

[3] F. M. T. Brazier, C. M. Jonker, and J. Treur. Principles of component-based design of intelligent agents. *Data Knowledge Engineering*, 41(1):1–27, 2002.

[4] A. Castor, R. Pinto, C. T. L. L. Silva, and J. Castro. Towards requirement traceability in tropos. In *Workshop em Engenharia de Requisitos*, pages 189–200, 2004.

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

[6] D. Herlea, C. Jonker, J. Niek, and J. E. Wijngaards. Specification of bahavioural requirements within compositional mas design. In *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume LNCS n1647, pages 8–27. Springer, 1999.

[7] J. Jamont and M. Occello. Using organizational structures emergence for maintaining functional integrity in embedded systems networks. In *IFIP Conf. on Artificial Intelligence Applications and Innovations*, pages 197–210. KAP, 2004.

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

[9] M. Occello, Y. Demazeau, and C. Baeijs. Designing organized agents for cooperation with real time constaints. In *Collective Robotics, First International Workshop*, volume 1456, pages 25–37. LNCS, Springer, 1998.

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

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