12

A Multiagent Method to Design Open Embedded Complex Systems

Jamont Jean-Paul and Occello Michel
University of Grenoble, LCIS/INPG-UPMF Lab
France

1. Introduction

Open physical complex systems involve multiple interconnected software and hardwar entities which enable logical/physical interactions between them and their shared environment. They rise to many hierarchical level which exhibit common behaviours. These entities have their own goals but participate to the accomplishment of the global system.
There are different classes of open physical complex systems like control systems processing systems, communication systems and interactive systems. Because these systems take over new wireless technologies, they are more and more distributed, decentralized and often not completely described.
Through the use of multiagent system to model Open physical complex systems (OCPS) two types of requirements emerge: requirements in methods and in specific system architectures. Concerning the specific methods, our contribution is the DIAMOND method (Decentralized Iterative Approach for Multiagent Open Networks Design (Jamont & Occello, 2007)). Concerning the requirements in architecture, our contribution is the MWAC model (Multi-Wireless-Agent Communication) based on our previous work on wireless sensor networks (Jamont & Occello , 2006).
In this chapter, we focus on specificities of the methodological requirements. We try to answer to some questions asked by this type of applications in lifecycle terms, about the design step and the formalism.
A method consists in concepts, in an approach and in tools. So, in a first section, we focus on the main concept of our works: the multiagent paradigm. In a second part, we present the approach of the DIAMOND method. The third part describes the different steps and activities of our method. Before concluding, we propose a discussion of the method in comparison to other multiagent methods.

2. Multiagent systems

An agent is a software entity evolving in an environment that it can perceive and in which it acts. It is endowed with autonomous behaviours and has objectives. Autonomy is the main concept in the agent issue: it is the ability of agents to control their actions and their internal states. The autonomy of agents implies no centralized control (Wooldridge, 1999).
A multiagent system is a set of agents situated in a common environment, which interact and attempt to reach a set of goals. Through these interactions a global behaviour, more

intelligent than the sum of the local intelligence of multiagent system components, can emerge.
The emergence paradigm deals with the unprogrammed and irreversible sudden appearance of phenomena in a system confirming that "the whole is more than the sum of each part". It is one of the expressions of collective intelligence (Deguet et al., 2006).
The emergence process is a way to obtain dynamic results from cooperation that cannot be predicted in a deterministic way. There are three types of emerging features (Marcenac, 1996): emergence of structures at the origin of the self-organization process, behaviour and emergence of properties.
It is difficult to qualify the emergent characteristics of a phenomenon. Some fundamental elements have been settled by S. Forrest (Forrest, 1991),(Muller; 2004) proposes an interesting specialization in the multiagent context that has been recently discussed and completed in (Dessales & Phan, 2005).
It asserts that a phenomenon is emergent if:

- there is a set of agents interacting via an environment, whose state and dynamics cannot be expressed in terms of the emerging phenomenon to produce in a vocabulary or a theory D,
- the dynamic of the interacting agents produces a global phenomenon such as, for example, an execution trace or an invariant,
- the global phenomenon is observable either by the agent (strong sense) or by an external observer (weak sense) in different terms from the subjacent dynamics i.e. another vocabulary or another theory D '.

To give a system of agents a particular global functionality, the traditional method consists in carrying out a functional decomposition of the problem into a set of primitives which will be embodied by the agents. The alternative suggested by L. Steels (Steels, 1990) aims at making this functionality emerges from the interactions between the agents. The advantage of the "emergent functionality" approach is first of all a reinforcement of the robustness of the system becoming less sensitive to the changes of the environment.
The adaptation of the whole multiagent system is generally obtained through emergence. It exist a lot of multiagent methods. We give here some references to these different works and the result of an analysis of these methods through many criteria.

3. Approach

The lifecycle of traditional methods applied to design hardware/software hybrid systems (see fig.1) starts with a requirements analysis followed by a portioning step. During this partitioning step, the designer chooses the system parts which must become either hardware or software parts %: the requirements analysis which is derived in a hardware one and a software one. At this stage, the two different 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 partitioning!
This type of lifecycle doesn't allow to take into account some late modification of requirements and is thus not well adapted to OPCS which cannot, by definition, be completely a priori specified.

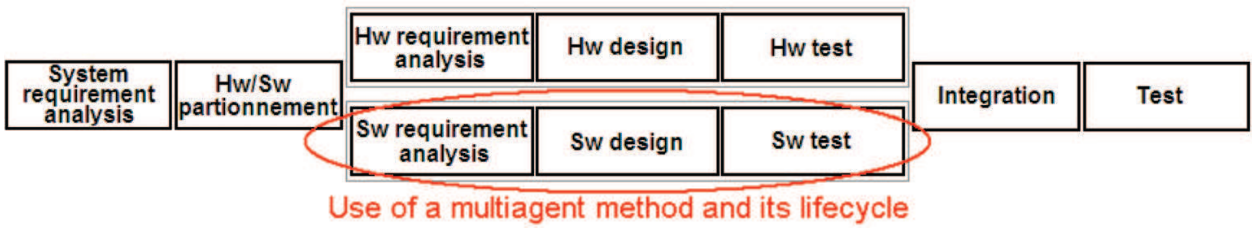


Fig. 1. Lifecycle of a traditional multiagent method

A few works deal with embedded multiagent systems, but new applications are strongl concerned by this domain (Pervasive computing (Carabelea et al., 2003), Ambiant computing (Maña & Rudolf, 2007)) and industrial applications of MAS (Parunak, 2000)). 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 generally taken into account only during the deployment step (Cossentino03 et al.), and are limited to the choice of the platform 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 supports 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 solution space 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 (Boehm, 1988).

The lifecycle of traditional method applied to design an hardware/software hybrid system (see fig.1) begin by a requirement analysis followed by, very early, by a portioning step. During this partitioning step, the designer chooses the system part which must become hardware part or software part: the requirement analysis which is declined in a hardware one and a software one. After this step, these two different parts are designed in parallel. At the end of lifecycle, these two parts are integrated to become operational system. Through this integration step (and the following test) some problem can emerge. Theses problems can call into question the software design, the hardware design or the twice. More deeply, it can be necessary to modify the result of the partitioning!

The evaluation of the different lifecycle models in respect with these previous criteria carries out the spiral lifecycle (Boehm, 1988) as the best choice in our context.

The DIAMOND method is built to design physical multiagent systems. Four main stages, distributed on a spiral cycle (see fig.2), may be distinguished within our physical multiagent design approach. The definition of requirements defines what the user requirements are 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. 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.

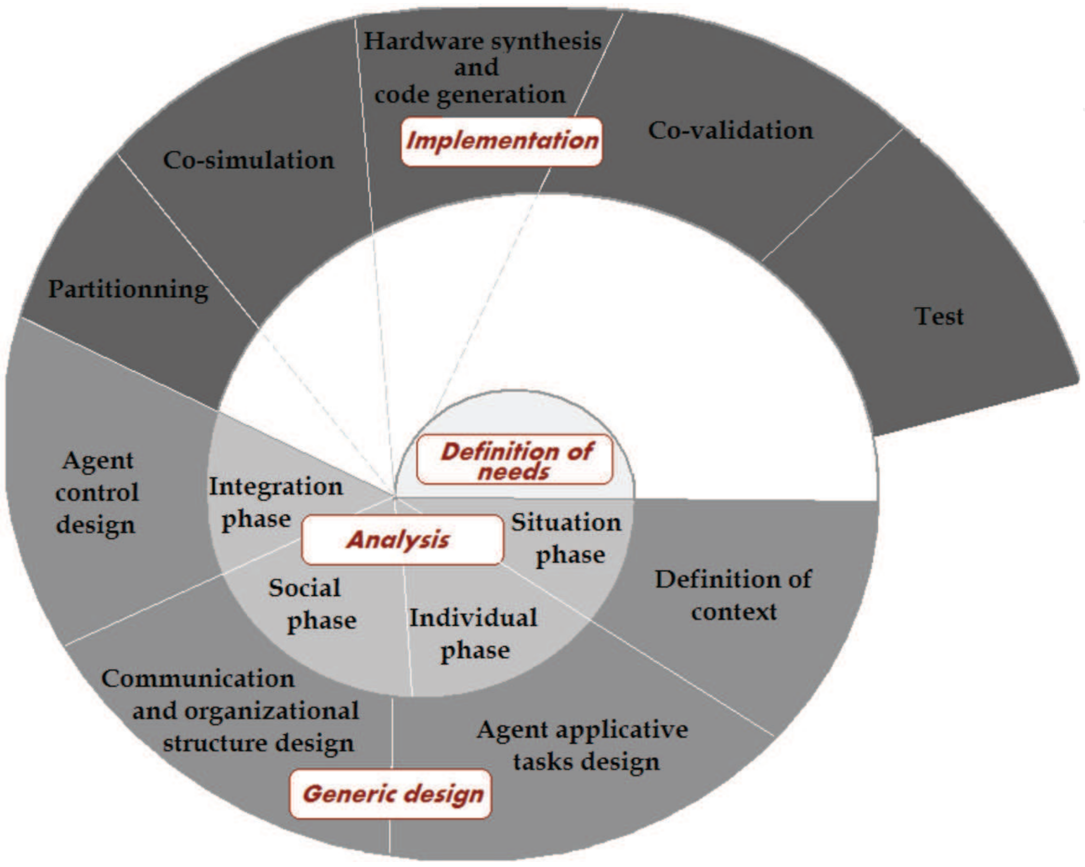


Fig. 2. Lifecycle of a traditional multiagent method

4. The DIAMOND method

4.1 Case study

To illustrate the various phases and activities of our method, we will use the robocup case study. To make the illustration easily understandable, we will adopt a simplified definition of requirements.

The experimental conditions are inspired by (Huang et al., 2001). Robots evolve on a football field (see fig. 3). A video recorder system makes possible to know the position of each robot as well as of the ball. These positions are periodically broadcasted to all robots. If the ball goes out of the limits of the field, a robot of the non faulty team recovers the ball and plays (the order is given by the referee). If a robot has no more battery or is dysfunctioning, the match is stopped (the order is given by the referee for human safety reasons) and the robot is withdrawn from the field: all robots must be then motionless. At the beginning of a match the robots must be located in their camp and the referee decides to give the guardian role to one robot of each team. So, the game is open and the team, which scores the higher number of goals in 90 minutes, wins.

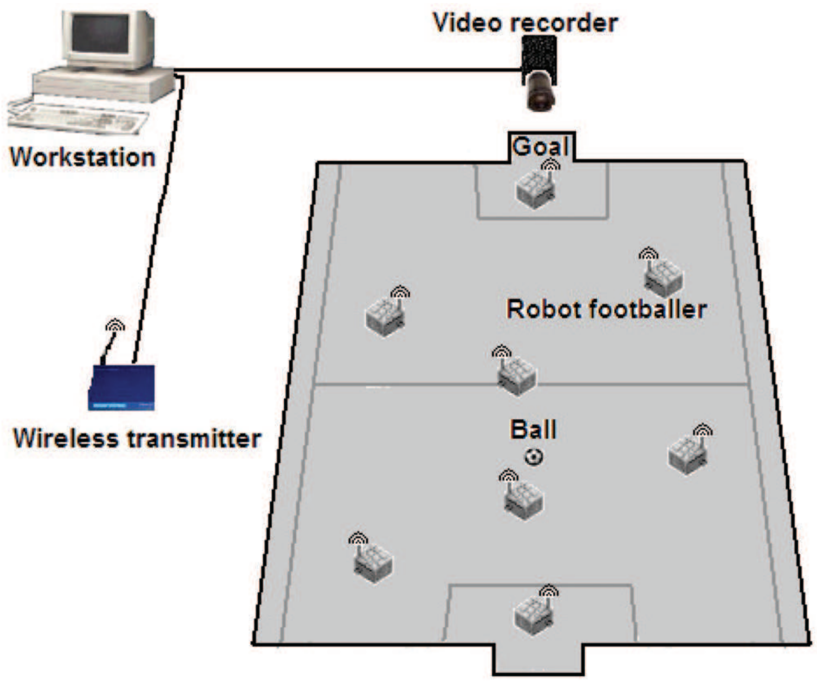


Fig. 3. Our case study

4.2 Definition of requirements

This preliminary stage begins by analyzing 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 UML sequence diagram can include physical interaction.

The second step consists in an original step: the study of the running mode and stop mode. This activity is very significant because it enables to structure the global running of the system. It is generally wishable that the system works in autonomy. But working with physical systems requires to identify many others possible behaviours: how must the system be before to stop it (robot in safety area...)? What must the system states be when it goes under maintenance? How must the system components be calibrated? What must the state of all the components be when an emergency stop occurs? Even if the problem is solved with a decentralized intelligence, this organization of these modes is easily understandable by the clients and the users. More of that, even if the system is approached with a decentralized intelligence, the system must respect laws and norms. They are very strong because the human safety can easily be altered.

This activity puts forward a restricted running of the system. It allows to specify the first elements necessary for a minimal fault-tolerance. Moreover, it 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 15 different modes regrouped in three families. The *stop modes* are relate to the different procedures for stopping the system. Moreover it allows to define the associate recognition states. The *running modes* focus on the definition of the recognition states of normal running, test procedures etc. The *failing operations modes* focus on the security

procedures (for example to allow a human maintenance team to work in the system) or to specify rules for restricted running etc.
Application to our case study. We find the following actors. The *referee (logical actor)* manages the match parameters: choose a goalkeeper and a camp for each team, verifies that robots respect the rules. It authorizes the human to withdraw a robot when all robots are motionless.
The *manager (physical actor)* withdraws robots when a problem occurs. The *ball (physical actor)* moves under the robot actions. The opposing team (*physical/logical actor*) shares the field with the studied one.
The *camera system* broadcasts the coordinates of each robot and of the ball.
There are two user cases. The *configuration* expresses that the referee chooses a field and a goalkeeper for each team. This user case triggers another one: the *games* opens the game (see fig.4).

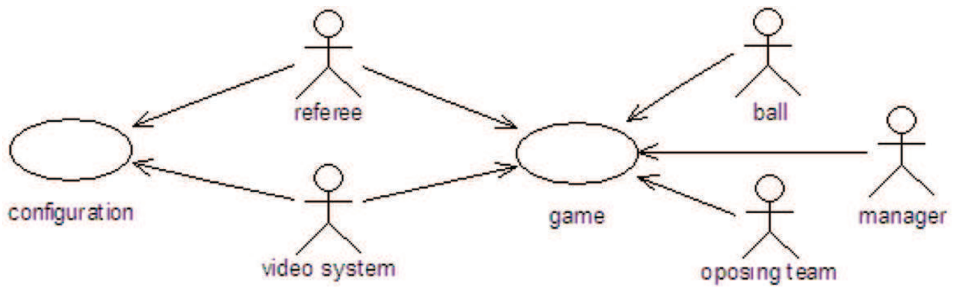


Fig. 4. Our case study

For our application, the identified modes are:

- Stops modes: Two modes of stops must be characterized: other modes are not exploited.
  - Idle: In a idle mode, the robots must be motionless.
  - Stops requested on normal mode: when a robot dysfunction occurs, the referee can decide to freeze the game.
- Running modes:
  - Normal mode: in this mode all the robots must answer to requests of the referee, there is no emergency stop.
  - Mode of preparation: during the phase of preparation, robots are positioned on the ground. Robots should neither then move nor use their actuators. This mode ends when the parameters setting period starts.
  - Mode of test: this mode will be used to calibrate the shooting power.
- Failure modes: only the management of the emergency stop is relevant in our application.
  - Mode of stop aiming to ensure the safety: If an emergency stop is activated, robots do not have any more the right to move or use effectors.

In this application, where the life period is short, importance of the other modes is not relevant.

4.3 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 built. This integrated multiagent design procedure encompasses five main phases discussed in the following.

Situation phase. The situation phase defines the overall settings, 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 agentification of the problem.
We insist here on some elements of reflexion about the characteristics of the environment (Russel & Norvig, 1995),(Wooldridge, 2000). 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 predictable 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 episodically. 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 has an impact on the agent architecture. Physical environments may require reactive or hydride architectures. The environment is *discrete* 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 will compose 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 entity in the system. This phase allows to identify the main entities that will be used and will become agents.
Application to our case study. The environment is not accessible. Each robot can know its geographical position, the position of the ball and of the other robots. Dimensions of the ground are known and the field of each team is communicated at the beginning of each part. The positions of each robot can be memorized at different dates to estimate displacements, directions of the robots and their trajectories. The trajectory of the ball obeys to physical laws. Agents can estimate this trajectory and act on it. Environment is rather not determinist. Even if agents cooperate and there is no dysfunction, an agent cannot know actions of other agents. However elements of the environment are not fully predictible like the trajectory of the ball. The possible actions on the environment are displacements (robots and ball). Environment is not episodical because we suppose that no intervention of the human is possible. The future evolutions depend only on the actions carried out by the robots. Environment is dynamic and continuous although the feasible actions are finite.
The active entities are the robot-players. The ball is a passive entity which obeys to agent actions (shootings) by a displacement according to the physical laws.
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 (Demazeau, 1995).
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.
Application to our case study. The agent world representation consists in a collection of triplets (id,x,y) and in the field dimension. In our application, robot players are modelled by agents. Their individual capabilities can be specified using a tree to show the different action levels (fig. 5).

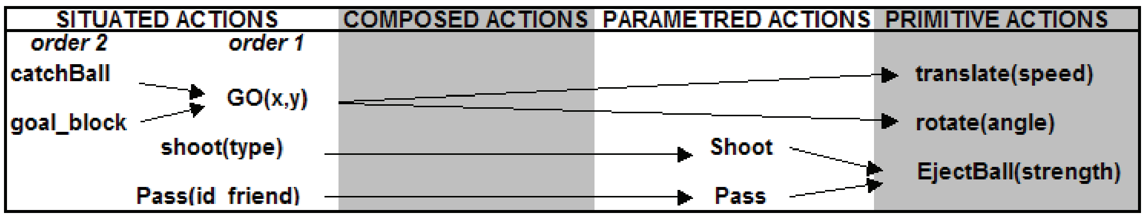


Fig. 5. Actions scheme

We specify the agent context with a context diagram (see fig 6).
After one iteration to take into account the society phase, individual behaviours are implemented using finite state machine. We can define an agent with the goalkeeper behaviour. Other agents can alternate two different behaviours (shooter or defender). For example, the goalkeeper behaviour defines that the agent must always be on a possible trajectory of shooting.

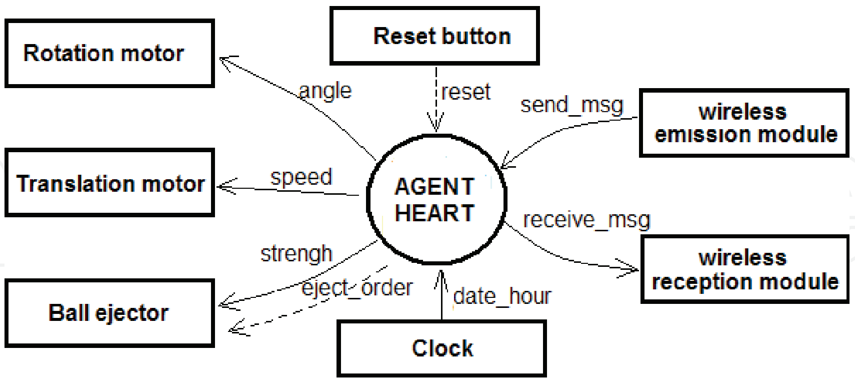


Fig. 6. Context diagram

Society phase. Interactions 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 another.

Application to our case study.
Representation of others: The positions of other players can be known by the capture of information from the video system (WIFI module). Their directions can be estimated if agents can memorize the previous positions. Friend's intentions can be announced.
Interactions: between the agents they are carried out by exchange of messages. An agent must be able to communicate with its team to diffuse its intention. It can use a peer-to-peer communication to solve a conflict or to choose a trajectory with a friend.
Collaborative actions can be instantiated: a player can request the ball when it has an occasion for shooting. It can ask somebody to change position to attract an opponent elsewhere.
Organization: A TEAM according to the requirement is composed of a goalkeeper and three other agents which can be SHOOTER or DEFENDER.
Collective behaviour can be implemented by finite state machines.
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 and trigger rules). The decomposition masks the notion of agent's control, i.e., how it handles its focus of attention, its decisions, and how 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.
Application to our case study. We illustrate this phase with two examples.
Influence: If an agent wants to move to a given point, somebody (a friend or not) can be on its trajectory. Correction: If the agent on the trajectory is a friend, the agent owning the ball has the priority.
Influence: Two agents request the ball for shooting. Correction: Agents use an election protocol (they exchange an estimation of their success probabilities).

4.4 The generic design

This stage is based on component decomposition. We can define a component as an elementary object, which performs a specific function that allows developers to define reusable segments of code. It is designed in such a way to easily operate with other components to create an application. So, a component is a reusable program building block, which is an identifiable part of a larger program. Components can be combined with others to build more complex functions. This phase offers an efficient process leading to 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. 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.

Application to our case study At this level, the designer chooses technical solutions for each sensors/effectors. The context diagram (fig. 6) is detailed (see the table 1).
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 (Occello et al., 1998), especially adapted to a real time context.

Information	Specification
Reset	Active on high logical level (1bit)
Angle	Relative angle in [- 180, +180] coded whole signed on 10 bits
Speed	Two speeds are possible. Entirety coded on 2 bits. (00: stop / 01: slow speed / 10: fast speed)
Strengh	Two levels of possible forces. Level coded on 1 bit (0: pass/1: shooting)
Eject_ball	Transition to high level
Date_heure	Number of milliseconds run out since the powering (32 bits)
Send_msg	Specific protocol bit field (sender 1octet, receiver 1byte, data_lenght 1octet, data 1-25octets)
Receive_msg	Specific protocol bit field (same than Send_msg)

Table 1. Details of the context diagram

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.

4.5 Implementation stage

Partitioning Phase. 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 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 (Occello et al., 1998),(Jamont et al., 2002),(Luo et al., 2007) and codesigns work like (Adams & Thomas, 1996):

- 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 f*lexibility* plays in favour 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 fewer *faults tolerant* than hardware components like EPLD. Indeed, microcontrollers 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 constraints* 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.

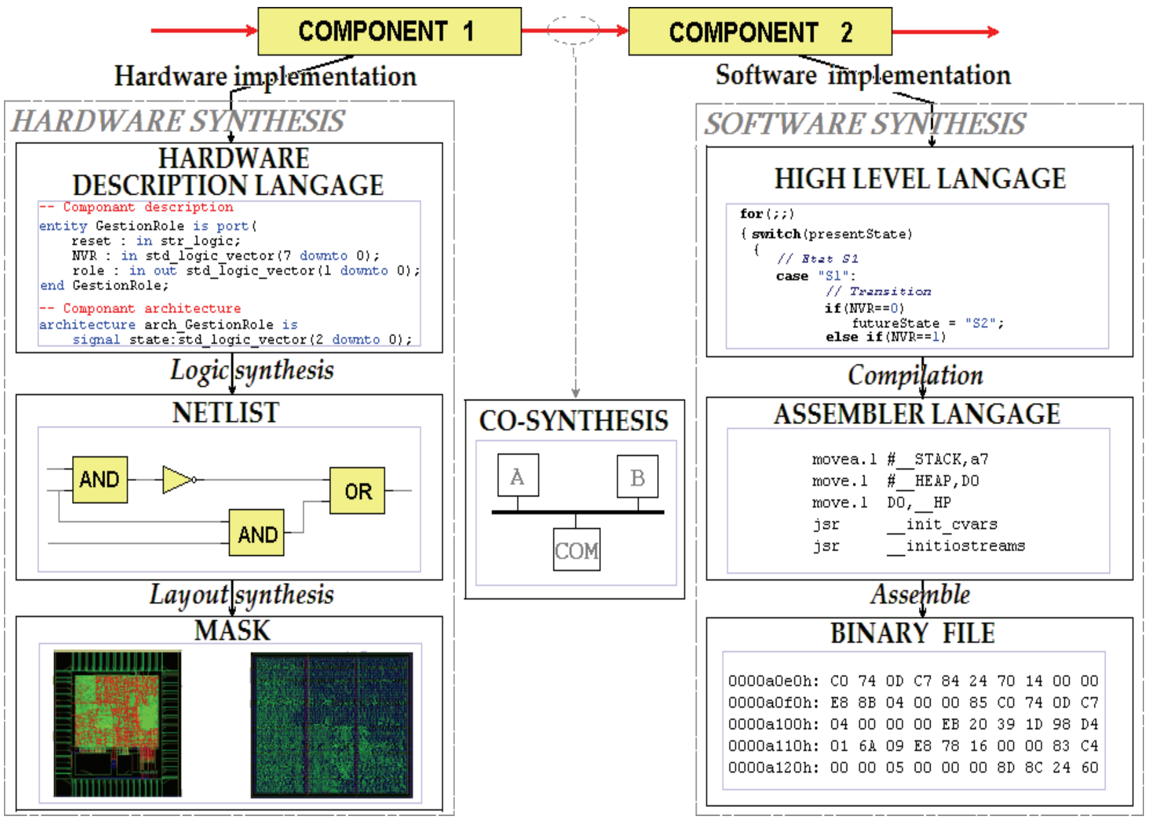
Co-simulation and co-validation Phases. This activity allows to simulate the collaboration between software part, hardware part and their interface.

Implementation Phase. At this level, each component is completely specified with 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 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 (Breuer et al. , 1999). The compilation of the code and the hardware synthesis of different specifications in VHDL are carried out like illustrated on figure 7.

Application to our case study. Today, the agents are embedded on autonomous processor cards. These cards are equipped with communication modules and with measuring modules to carry out agent tasks relative to the instrumentation. These cards supply a real time kernel. The KR-51(the kernel's name) allows multi-task software engineering for C515C microcontroller. We can produce one task for one capability. We can then quite easily implement the parallelism inherent to agents and satisfy the real-time constraints.



```
-- 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
```

Fig. 7. Software component synthesis and hardware component synthesis

5. Discussion about the DIAMOND method

5.1 Lifecycle and phases

Most existing multiagent methods usually distinguish only analysis and design phases (Deloach et al., 2001). Very few methods deal with other phases. We can find for example a deployment phase in MASSIVE or Vowels. This deployment phase takes in our particular field a great importance since it includes the hardware/software partitioning. A last and major difference between DIAMOND and other multiagent approach 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 requirements.
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 (Herlea et al., 1999), for this reason we adopt a lifecycle 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 lifecycle. Even if some works attempt to introduce iterative cycles as Cassiopeia (W) or Gaia, the proposal of a spiral lifecycle is very original.
In the definition of requirements 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 (Lind, 2001),(Brazier et al., 2002). These components are used to simplify the work of the designer through visual programming, to manage the complexity through a functional decomposition, to increase the genericity through reusability, to simplify the partitioning because the analogy between soft components and chips enables the hardware tools and the software tools to share a unified vision.
Table 2 comes from the work of G. Picard (Picard, 2004). It gives an insight of the different methods and the qualitative results of the comparison between them.

	Model of lifecycle	Requierements	Analysis	Design	Implementation	Test	Deploiement	Maintenance	Délivrables	Quality managment	Project managment
ADELFE *(Bernon et al., 2002)*	V	+	++	++	+	+	±	±	++	+	++
AAII *(Kinny et al., 1996)*	Waterfall	-	++	+	--	--	--	--	+	--	--
Aalaadin *(Ferber & Gutknecht, 1998)*	Waterfall	-	++	+	++	-	+	-	-	--	--
Cassiopée *(Drogoul & Collinot, 1998)*	Iterative	--	++	+	--	--	--	--	±	--	--
DESIRE *(Brazier et al., 2002)*	Waterfall	-	±	++	±	++	--	--	-	--	--
Gaia *(Wooldridge et al., 2000)*	Iterative	-	++	++	--	-	--	--	++	--	--
MaSE *(DeLoach et al., 2001)*	Waterfall	--	++	++	+	±	--	--	++	--	--
MASSIVE *(Lind, 2004)*	Incremental	+	++	++	+	+	++	±	+	-	-
MESSAGE *(Lind, 2001)*	Iterative	+	++	++	+	±	±	±	++	+	+
PASSI *(Chella et al., 2006)*	Incremental	+	++	++	+	±	++	±	++	--	--
Prométheus (Padgham et al., 2007)	Waterfall	-	++	++	+	-	--	--	+	--	--
Tropos *(Castor et al., 2004)*	Incremental	++	++	+	+	±	--	--	-	--	--
Voyelles *(Ricordel & Demazeau, 2000)*	Waterfall	-	++	++	+	+	+	--	--	--	--
DIAMOND	Spiral	+	++	++	++	(+)	++	+	+	(?)	(?)

++ : Properties are fully and explicitly supported
+ : Properties are taken care of in an indirect way
± : Properties are potentially Supported
--: Properties are not explicitly taken into charges
- : Properties are not supported

Table 2. Comparison synthesis of the multiagent methods

The criteria used in table 2 are:

- Requirements: Is the requirements gathering taken into account?
- Analysis: Is the analysis stage taken into account?
- Design: Is the design stage taken into account?
- Implementation: Is the implementation stage taken into account?
- Test: Is the testing process taken into account?
- Deployment: Is the deployment stage taken into account?
- Maintenance: Is the maintenance stage taken into account?
- Deliverables:
- Do the deliverables are clearly identified and associated with specific steps?
- Quality Management: Is the quality management taken into account?
- Project Management: Are the guidelines of conduct project are clear?

5.2 Models and notations

Multiagent method generally use notations and models from only one origin (Bernon et al., 2002) like UML (Mase , AAII, MESSAGE, PASSI). Other methods use many notation like TROPOS (notation i* coming from the knowledge engineering, A-UML (Koning et al., 2001) 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 (Herlea et al., 1999) 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 requirements. The interpretation of our use case diagrams is slightly different than their common use (as in (Bernon et al., 2002)) 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 decisional 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 finite state machines or a components set to describe the internal running. These formalisms enable to generate software code or hardware specifications in VHDL.

In this section, we compare our method with other multiagent methods (ADELPH, PASSI, MASE, GAIA, DESIRE, MASSIVE, MAMOSACCO etc.) In a first subsection we talk about lifecycle and stages. In the second subsection we focus on models and notations.

The methods multi-agents operating adopt mostly notations and models of a single origin (see table 3).

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

	Requierement	Analysis	Design
ADELFE	UML diagrams (use case, sequence, collaboration)	UML diagrams (sequence, class), A-UML protocols	UML diagrams (class, paquetage, stéréotypes)
AAII		UML diagrams (collaboration, class)	UML object diagrams
Aalaadin		AGR organization diagram	A-UML diagrams
Cassiopée		FSM/dependency	
DESIRE		entity relationship diagram, FSM	Components
Gaia		Array, logic langage	
MaSE		UML sequence diagram	UML class diagrams
MAMOSACO	Arrays	UML class diagram, parametred Petri network, SADT actigram, OSSAD processing model	UML class diagrams, parametreed Petri networks
MASSIVE	UML use case diagrams	UML activity diagram	UML class diagrams
MESSAGE	UML use case diagrams	UML diagrams (class and activity), A-UML diagrams (collaboration)	UML class diagrams
PASSI	UML diagrams (use case, sequence), UML like packetage diagram	UML diagrams (sequence, class)	UML deployment diagrams
Prométheus		UML diagrams and A-UML diagrams	UML and A-UML diagrams
Tropos	i*	State diagram, A-UML protocols	

DIAMOND	UML diagrams (use case, sequence), textual specifications for the modes study, glossary	UML diagrams (sequence), A-UML protocols, context diagram (SART), entity relationship diagram (organisation)	FSM, components VHDL

Table 3. Notation used by these different methods

Our future work will be to improve the MASC tool (MultiAgent System Codesign) associated with the DIAMOND method. The agent design with components and the code generation in Java and C languages are operational. The VDHL specification generation is partially developed.

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 hardware/software multiagent systems.

7. References

Adams, J.; Thomas, D. (1996) The design of mixed hardware/software systems, *Proceedings of the 33st Conference on Design Automation*, pp 515-520, ISBN 0-89791-779-0, USA, June 1996, ACM Press.

Bernon, C.; Gleizes, M.-P.; Peyruqueou, S. & Picard, G. (2003), ADELPH: A methodology for adaptive multi-agent systems engineering., In: *Engineering Societies in the Agents World III*, page numbers 156-169, Springer Verlag, ISBN 3-540-14009-3, 2002,Spain.

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

Brazier, F. M. T. ; Jonker, C. M. & Treur, J. (2002). Principles of component-based design of intelligent agents. *Data Knowledge Engineering,* Vol. 41, No. 1, April 2002, Elsevier, page numbers 1-27, ISSN 0169-023X.

Breuer, P. T.; Madrid, N.M.; Bowen, J. P.; France, R. B.; Larrondo-Petrie, M.M. & Kloos, C. D. (1999) Reasoning about vhdl and vhdl-ams using denotational semantics, *Proceedings of the Design, Automation and Test in Europe*, pp 346–352, ISBN 0-7695-0078-1, Germany, March 1999, IEEE Computer Society, Munich.

Carabelea, C.; Boissier, O. & Ramparany, F. (2003) Benefits and requirements of using multi-agent systems on smart devices, *Proocedings of 9th International Euro-Par Conference*, pp 1091-1098, ISBN 3-540-40788-X, Austria, August 2003, Springer Verlag, Klagenfurt.

Castor, A.; Pinto, R.; Silva, C. T. L. L. & Castro, J. (2004). Towards requirement traceability in tropos, *Proceedings of the Workshop em Engenharia de Requisitos*, pp 189–200, ISBN 950-658-147-9, Argentina, Dec. 2004, WER, Tandil.

Chella, A.; Cossentino, M., Sabatucci, L. & Seidita, V. (2006). Agile PASSI: An agile process for designing agents. *Computer Systems: Science & Engineering,* Vol. 21, No. 2, March 2006, In press, ISSN 0267-6192.

Antonio Chella, Massimo Cossentino, Luca Sabatucci, Valeria Seidita: Agile PASSI: An agile process for designing agents. Comput. Syst. Sci. Eng. 21(2): (2006)

Cossentino, M.; Sabatucci, L. & Chella, A. (2003). A possible approach to the development of robotic multi-agent systems, *Proceedings of the IEEE/ACM/WIC Conference on Intelligent Agent Technology*, pp 539–544, ISBN ISBN 0-7695-1931-8, Canada, 2003, Halifax.

Deguet, J.; Demazeau, Y. & Magnin, L. (2006). Elements about the emergence issue: A survey of emergence definitions, *Complexus*, Vol. 3, No. 1-3, 2006, pp. 24-31, ISSN 1424-8492.

DeLoach, S. A.; Wood, M. F. & Sparkman, C. H. (2001). Multiagent systems engineering. *International Journal of Software engineering and Knowledge Engineering,* Vol. 11, No. 3, June 2001, page numbers 231-258, ISSN 0218-1940

Dessalles, J.L.; Phan, D. (2005). Emergence in multi-agent systems: cognitive hierarchy, detection, and complexity reduction, *Proceedings of the 11th annual meeting of the Society of Computational Economics*, June 2005, Society of Computational Economics, University of Washington.

Demazeau, Y. (1995). From interactions to collective behavior in agent-based systems, *Proceedings of European Conference on Cognitive Science*, pp. 14-17, ISBN, France, Avril 1995, Saint-Malo

Drogoul, A & Collinot, A. (1998). Applying an agent oriented methodology to the design of artificial organizations: A case study in robotic soccer. *Journal on Agents and Multi-Agent Systems,* Vol. 1, No. 1, 1998, Kluwer Academic Press, page numbers 113-129, ISSN 1387-2532

Ferber, J. & Gutknecht, O. (1998). A Meta-Model for the analysis and design of organizations in multi-agent systems, *Proceedings of the 1998 International Conference on Multi-Agent Systems*, pp. 128-135, ISBN 0-8186-8500-X, France, July 1998, IEEE Computer Society, Paris.

Forrest, S. (1991). *Emergent computation,* The MIT Press, ISBN 978-0262560573, England.

Herlea, D. E.; Jonker, C. M.; Treur, J. & Wijngaards, N. J. E. (1999) Specification of Bahavioural Requirements within Compositional Multi-agent System Design, *Proceedings of 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pp 8–27, ISBN 3-540-66281-2, Spain, June 1999, Springer, Valencia.

Huang, H.-P.; Liang, C.-C & Lin C.-W. (2001) Construction and soccer dynamics analysis for an integrated multi-agent soccer robot system, *Natl. Sci. Counc. ROC(A)*, Vol. 25, No. 2, 2001, pp. 84-93.

Jamont, J.-P; Occello, M. (2007), Designing Embedded Collective Systems: The DIAMOND Multiagent Method, *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, pp. 91-94, ISBN 0-7605-3015-X, Greece, October 2007, IEEE Computer Society.

Jamont, J.-P; Occello, M. (2006), A Self-organized Energetic Constraints Based Approach for Modelling Communication in Wireless Systems, In: *Advances in Applied Artificial Intelligence*, page numbers 101-110, Springer Verlag, ISBN 3-540-35453-0, 2006, France.

Jamont, J.-P.; Occello, M. & Lagreze A. (2002). A multiagent system for the instrumentation of an underground hydrographic system, *Proceedings of IEEE International Symposium on Virtual and Intelligent Measurement Systems*, pp. 20-25, ISBN 0-7803-7344-8, USA, May 2002, IEEE Measurement and Instrumentation Society, Mt Alyeska Resort

Kinny, D.; Georgeff, M. & Rao, A. (1996). A methodology and modelling technique for systems of BDI agents, Agents Breaking Away: *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-AgentWorld*, pp. 56-71, ISBN 3-540-60852-4, The Netherlands, January 1996, Springer-Verlag

Koning, J.-L.; Huget, M.-P.; Wie, J. & Wang, X. (2001). Extended Modeling Languages for Interaction Protocol Design, *Proceedings of the Second International Workshop on*

Agent-Oriented Software Engineering, pp. 68-83, ISBN 3-540-43282-5, Canada, May 2001, Springer, Montreal

Lind, J. (2004). *Interative Software Engineering for multiagent systems: The MASSIVE Method, Springer Verlag*, ISBN 3-540-42166-1, Berlin

Luo, J.; Xu, L.; Jamont, J.-P.; Zeng, L. & Shi Z. (2007). Flood decision support system on agent grid: method and implementation. *Enterprise Information Systems*, Vol. 1, No. 1, (November 2007) , Taylor and Francis, page numbers (1751-1757), ISSN 1751-1757.

Maña, A. & Rudolf, C.(2007). *Developing Ambient Intelligence,* Springer, ISBN 978-2-287-78543-6, Paris

Marcenac, P. (1996). Emergence of behaviors in natural phenomena agent-simulation. *Complexity International*, Vol. 3, 1996, ISSN 1320-0682.

Muller, J.-P. (2003), Emergence of collective behaviour and problem solving, In: *Engineering Societies in the Agents World IV*, page numbers 1-21, Springer, ISBN SBN 3-540-22231-6, 2003, England.

Occello, M. ; Demazeau, Y. & Baeijs C. (1998). Designing organized agents for cooperation in a real time context, *Proceedings of the first International Workshop of Collective Robotics*, pp. 25-73, ISBN 3-540-64768-6, France, March 1998, Springer-Verlag, Paris

Padgham, L.; Thangarajah, J. & WinikoffParunak, M., (2007). AUML protocols and code generation in the Prometheus design tool, *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp.270-271, ISBN 978-81-904262-7-5, Hawaii, May 2007, IFAAMAS.

Parunak, H. V. D. (2000). A practitioners? review of industrial agent applications. Autonomous Agents and Multi-Agent Systems, Vol. 3, No. 4, (2000) page numbers 389-407, ISSN 1387-2532

Picard, G. (2004). *Methodology for developping adaptive multi-agent systems and designing software with emergent functionality (PHD thesis),* Institut de Recherche en Informatique de Toulouse, France.

Ricordel, P.-M. & Demazeau, Y. (2000). From analysis to deployment: A multi-agent platform survey, *Proceedings of 1st International Workshop on Engineering Societies in the Agent World*, pp. 93-105, ISBN 3-540-41477-0, Germany, 2000, Springer-Verlag, Berlin

Russel, S. & Norvig P. (2002) *Artificial Intelligence : a Modern Approach – 2nd edition*. Prantice-Hall, ISBN 978-0137903955.

Steels, L. (1990). Cooperation between distributed agents through self-organisation, *Proceedings of IEEE Workshop on Intelligent Robots and Systems,* pp 8-14, ISBN 0-7803-8464-4, Japan, Jul. 1990, IEEE Robotics and Automation Society.

Wooldridge, M.; Jennings, N. R. & Kinny, D. (2000). The GAIA methodology for agent oriented analysis and design. *Journal on Agents and Multi-Agent Systems,* Kluwer Academic Publishers, Vol. 3, No. 3, September 2000, page numbers 285-312, ISSN 1387-2532

Wooldridge, M.-J. (1999). Intelligent agents. In: *Multiagent systems: A modern approach to Distributed Artificial Intelligence*, G. Weiss (Ed.), page numbers 27-79, MIT Press, ISBN 0-262-73131-2, 1999, England.