12th IMEKO TC1 & TC7 Joint Symposium on
Man Science & Measurement
September, 3 – 5, 2008, Annecy, France

DESIGNING WIRELESS INTRUMENTATION SYSTEM WITH THE DIAMOND METHOD

Jean-paul Jamont , Michel Occello, André Lagrèze

University of Grenoble II, LCIS Labs, Valence, France, {jean-paul.jamont,michel.occello,andre.lagreze}@iut-valence.fr

Abstract: Designing wireless instrumentation systems requires to work with two levels: the individual level (the sensors) versus the social level (the whole system). We introduce the DIAMOND method which allows to design open embedded complex systems using a multiagent approach.

Keywords: Multiagent systems, Design of instrumentation system, co-design approach.

1. INTODUCTION

Wireless instrumentation systems (WIS) are sets of embedded systems (intelligent sensors) strongly related with their environment. These sensors integrate a software part and a hardware part (electronic cards, sensors, effectors, medium access control, routing algorithm, interaction protocols etc.).

WIS are based on a plurality of low power transceivers transmitting measurement data from a set of sensors to a central workstation. These systems are distributed but, more often now, acquire a decentralized intelligence. The adaptation feature comes from the cooperation between the entities of the whole system.

Such systems can be observed at two levels: an individual level and a social level. In fact, in an individual view an entity has its own capacity, its own knowledge and want reach its own objectives. In the social level, we focus our work on the whole system: the entities must cooperate to reach a global aim which can be contradictory with its own aim.

This paper aims to present our approach called DIAMOND (Decentralized Iterative Multiagent Open Networks Design) for the design of open embedded complex systems with multiagent system. More specifically, we focus on the integration of the individual part of such system with the social part.

2. OVERVIEW OF THE DIAMOND METHOD

A multiagent method At its origin, the DIAMOND method has been built to design embedded multiagent systems. This method addresses lot of application fields including manufacturing control and collective robotics. One of the first applications that we have designed with this method was a WIS to study an underground river system [1].
Multiagent systems are well suited for analyzing and designing complex systems such as networks of distributed autonomous entities behaving in an open environment. For example, the context of the instrumentation of an underground river system involves an open network of intelligent sensors whose cooperation must be monitored in order to insure the best organization.

Multiagent systems (MAS) are a collection of several agents. It is necessary to start by defining what we call an agent. Once this notion has been introduced, we shall approach the concept of MAS. For the past few years the resolution agents have undergone a strong and fast development of researches. The confrontation of several definitions [2][3] allows us to say that, in our context, “an agent is a software entity embedded in an environment which 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 can be defined as the agent’s ability to have control over their actions and their internal states. The autonomy of agents implies no centralized control.
The power of an agent decomposition is the decentralization of the intelligence, i.e. of the decision capabilities, and of entities’ knowledge. In MAS, agents are situated in a common environment. They interact and attempt to reach a set of goals. Through these interactions a global behaviour, more intelligent than the sum of local multiagent system components intelligence can emerge. The emergence process is a way to obtain dynamic results that cannot be predicted.

The DIAMOND method 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 (fig.1).

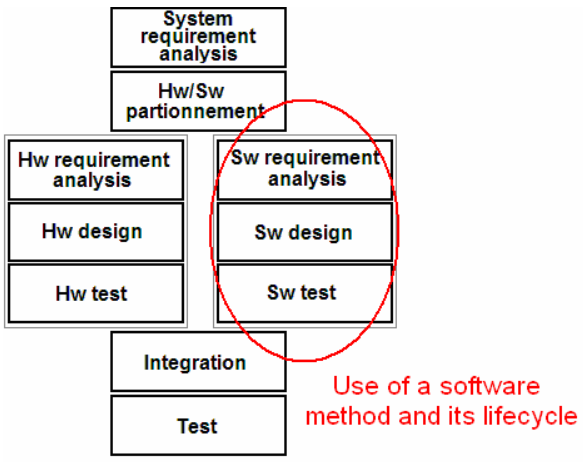


Fig. 1. Traditionnal software method approach

The DIAMOND method is a codesign method because it unifies the development of the hardware part and the software part. In fact, the partitioning step is pushed back at the end of the lifecycle to authorize modifications of the requirement, refinements and iterations.
Four main stages, distributed on a spiral shaped lifecycle [4] (fig 2), 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 into a multiagent solution. The third stage of our method starts with a *generic design* which aims to build the multiagent system 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.

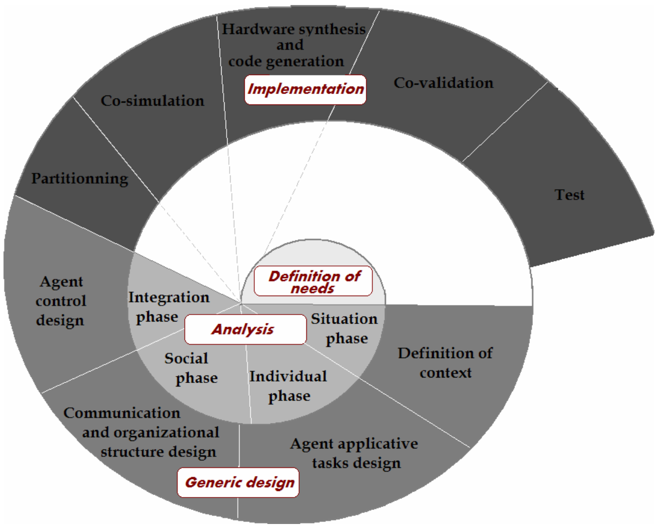


Fig. 2. The DIAMOND lifecycle

3. REQUIREMENTS DEFINITION

This preliminary stage starts by an analysis of the physical context of the system (identifying workflow, main tasks, etc...).

Study of actors. We study the different actors. An actor is a role played by something or someone who interact with the system. A person of the real work can play many roles.

Use case analysis The actors contribute in use cases. A use case is a description of a behaviour of the system as it responds to a request that originates from outside of that system. Study the use case enables to capture the functional requirements. So, an use case allow to describe how the different actor interact with the designed system to achieve a specific goal.

Service requirement analysis. We must study the services requirements of these actors. A sequence diagram shows different processes that live concurrently. It show the messages exchanged between these processes and the order in which they occur.
During this analysis, we use the UML use case diagram and the UML sequence diagram. They can include physical interactions.

Particular mode analysis The second step consists in the study of particular modes for a system that we call "running mode" and "stop mode". This step comes from the GEMMA methodology [5]. Working with physical systems requires to identify many particular possible behaviors directly linked with the embedded aspect of the system: In which state must the system be when going under maintenance? How to calibrate the system components? How must be the state of all the components when an emergency stop occurs (robot in safety area...)? Even in a decentralized intelligence context, the conditions defining these modes must remain easily understandable. The users of the system must respect laws and norms (the human safety can 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, this phase enables to identify cooperative (or not) situations and to define recognition states in order to analyze, for example, the self-organizational process of an application. This activity allows to take into account the safety of the physical integrity of the users possibly plunged in the physical system.
We have defined fifteen different modes grouped in three families. The *stop modes* are related to the different procedures to stop the system. They define the associate recognition states too. The *running modes* focus on the definition of the recognition states of normal running, test procedures etc. The *failure operations modes* concentrate the security procedures (for example allowing a human maintenance team to work on the system) or to specify rules for restricted running etc.

4. THE SOCIETY PHASE

The multiagent stage is handled in a concurrent manner at two different levels (Fig. 3.). 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.

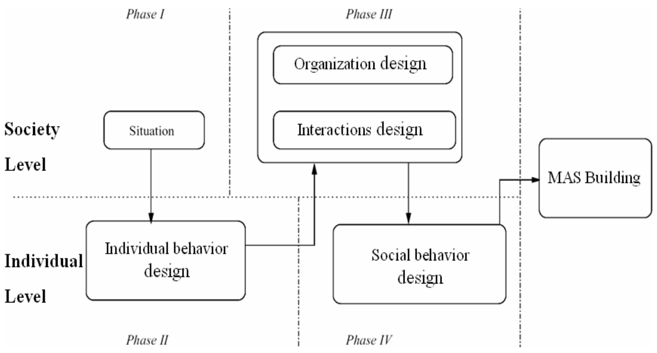


Fig. 3. The DIAMOND lifecycle

Situation phase The situation phase defines the overall setting, i.e., the environment, the agents, their roles and their contexts. This stems from the analysis stage. We first examine the environment boundaries, identify passive and active components and we proceed to the problem agentification.
We insist here on some elements of reflection about the characteristics of the environment. 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 e*pisodic* if its next state does not depend on the actions carried out by the agents. Some parts of a physical environment are generally episodic. 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 impacts 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 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 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 is 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.

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 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.

5. THE GENERIC DESIGN

This stage is based on 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. Components can be combined each other 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. 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 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, it will be necessary for them to be reactive using a stimuli/response paradigm to be most efficient.

We evaluated in [6] the impact of the time real aspects on the design of the agents and showed that they must be taken into account for each ability of the agents and at each level of the design as reminded on figure 4.

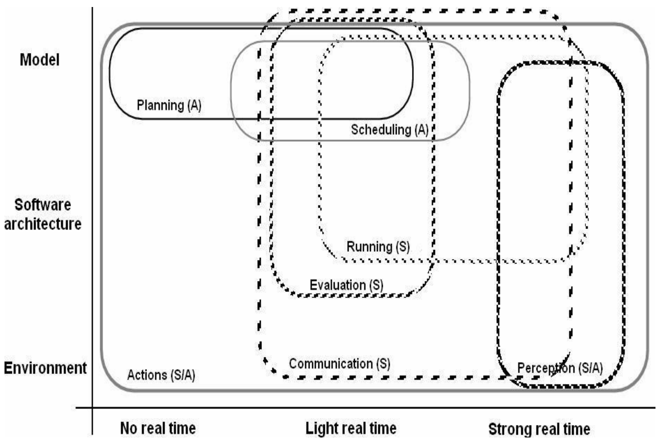


Fig. 4. Distribution of real-time constraints (recommended temporal mechanism (S)ynchronous or (A)synchronous)

Agent applicative tasks design phase. We must build the external shell of the agent i.e. elaborate the interface with the external world for each sensors and effectors. 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 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. At this level the designer has to build agents applicative tasks.

6. IMPLEMENTATION PHASE

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

A first level relates to agent parts for which the partitioning question doesn't exist. Indeed some elements must be hardware as input/output device 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 and codesign works like [7]: cost, performance, flexibility, fault-tolerance, ergonomic constraints and the algorithmic complexity.

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, microcontrollers use memories, stack structures with possible overflow etc. The *internal fault tolerance* will be thus a criterion which will play in favor 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 specifications for the hardware part and the software
part. For each component, the designer has already selected if he wishes hardware or 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 [8].

7. CASE OF STUDY

We illustrate the requirements definition and the society phase of our method with the EnvSys project [1]. It is a simplified version of the real analysis.

6.1 Requirement definition

Preliminary Approach This project (Fig. 5.) aims the wireless instrumentation of underground river systems. More precisely, our work on this project focuses on the design of a communication management of this system. The measures must reach to a workstation located at the exit of the underground river system.

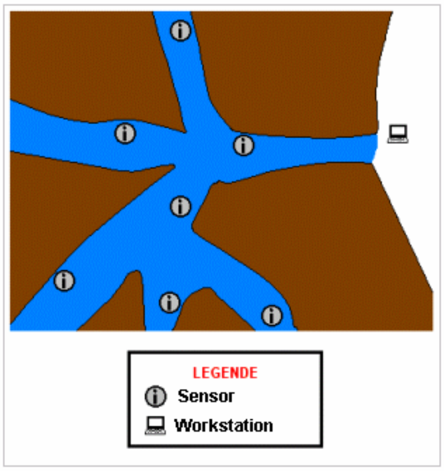


Fig. 5. Illustration of the EnvSys project

The `workstation` is a computer which collects the measures coming from the various sensors. They are stored in a text file. These records are saved with this format: <id_{sensor}><$id_{measure\_1}$><$data_1$><TAB>...<TAB><$id_{measure\_n}$><$data_n$><RC>.
id_{sensor} is the identifier of the sensor which sends the measures (2 bytes), $id_{measure\_1}$ is the identifier of the type of measure (1 byte - 1: temperature, 2:pressure, 3:ph), $data_n$ is the measure.
These measures are used by an `analyst` (viewable on site or remotely) The workstation also enables to know the IDs of the reached sensors. It enables speleologists to verify that their deployed sensors are always present and reachable.
The `sensors` are immersed in the underground river system. They are autonomous from an energy point of view (batteries). They can ensure several functions:

- A measure function: it is the first role of a sensor. It interacts with its environment to carry out an information.
- A relay function: Each sensor has a limited transmission range. It is therefore necessary that each sensor help others to send their measurements to the workstation.

In the complete analysis we must characterize the *wireless communication* and the *fault tolerance requirement*.

Study of actors The system consists of a set of sensors and a workstation. The actors acting on this system are:

- The speleologists: he put the sensors in the cave. He checks periodically that the sensors transmit their measures or if they need to be replaced.
- The analyst: he analyzes the collected data.

When he put a sensor in the cave, the speleologist must know how many other sensors are in its range. When he finished this operation (and when he wants to verify that the system works i.e. that the data necessary to the analyst are collected by sensors) he may launch a test. The analyst can view the data which arrives in the workstation, store them or destroy them.

Use case analysis We can identify the different use cases:

- The use case *configuration* that allows the creation of the network and the addition of sensors.
- The use case *measure* that allows to collect all the measures on the workstation. During this use case, the user can:
  - manage the data (analyst),
  - check the topology i.e. sensors that are still in working condition and participate in the collecting data task (speleologist).

The next figure (Fig. 6.) illustrates these cases of use and their organization.

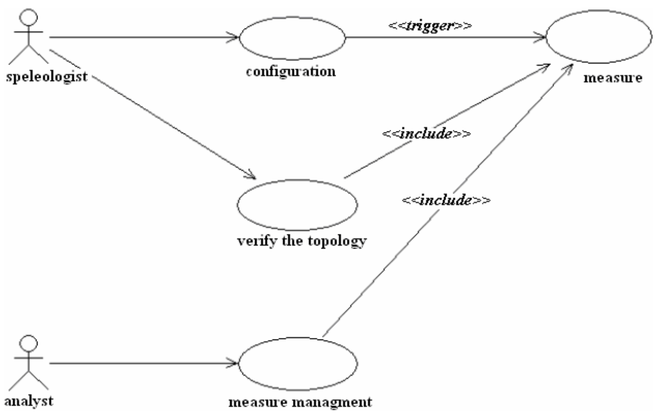


Fig. 6. EnvSys use case diagram

Service requirement analysis We give here an abstract of the different sequence diagrams. The following diagram (figure 7) introduces the services by the actors "speleologist" and "analyst" when the system is used to instrumente the underground river system.

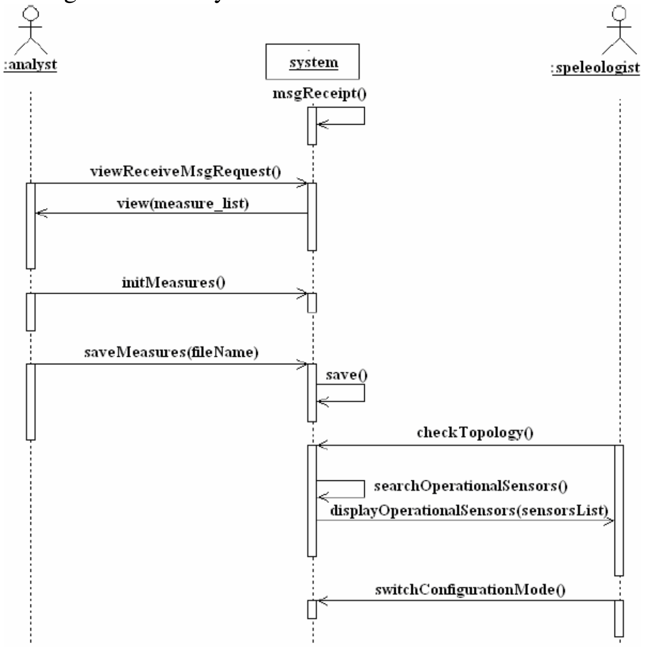


Fig. 7. An example of sequence diagram

Particular mode analysis. In normal measurement mode, the data arrive at the workstation. If f_i is the sending frequency used by sensor i to transmit its measures, then if during the period $Tmin_i = 1/f\_i + t_{delivery}$ there is no reveived message from i, we have been rejected for the normal mode. It may be necessary to add sensors. The analyst must determine whether the missing data can distort the analysis of results.
A mode of preparation has been identified. It is linked to the use case "configuration". In this use case, the speleologist must to know the neighbourhood of the deployed sensor.

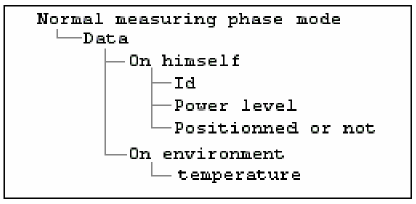
6.2 Society phase

Situation phase The environment is the underground river system. The environment consists in the information measurable by the sensors: in our case we will consider, as a first step, only the temperature.
Sensors are located but may not acquire their geographical position. The environment, as the majority of physical environments, is deterministic, not episodic, dynamic and continuous.
The components of our system are:

- The workstation: this station receives measures from the sensors. It allows the user to manage data or discover the network topology.
    - ⇒ This entity is not autonomous. The workstation will not become an agent.
- The sensors: As seen above, the sensor must interact with the environment to extract data. It must then send these data to a workstation. They have a limited amount of energy. This amount of energy should enable the measurements process and message delivery tasks. Sensors must cooperate because they involve a multihop communication. There are numerous paths to enable a communication. A sensor must decide himself whether he can still help other sensors to communicate
  ⇒ They are active entities. They are autonomous and involve the cooperation.
  ⇒ They become agents.

To address the communication problem, we will use the MWAC model specially designed for this type of problem [9].

Individual phase *Tree of capabilities.* The agent must perform measure primitive tasks and send these measures to the workstation. An agent is unable to send its measure to the collection station at this part of the analysis.
Building representation of itself and of the environment. An agent has an identifier and knows its energy level. However, the agent does not require to have a representation of its environment. Indeed, relevant information would be its geographical position, but it can not acquire it.
The required data on the environment which will be acquired, lead to define the agent sensors. In our case we consider only the temperature.



```
Normal measuring phase mode
 └─Data
    ├─On himself
    │  ├─Id
    │  ├─Power level
    │  └─Positionned or not
    └─On environment
       └─temperature
```

Fig. 8. Tree of capabilities and building representation of itself and the environment (individual phase)

Context diagram. When a speleologist places a sensor, he must know how many other sensors are in its range: it is necessary to have a screen. He requires a sensor to carry out the temperature.
Establish the behaviour of agents. The behaviour of the agent is actually very simple. It will periodically call upon its sensors to establish measures. He requires to send the measure to the workstation.

Social phase *Defining the knowledge about others.* We require the help of others agents to deliver messages to the workstation. It is impossible to know its geographical position. We will use the MWAC model to enable a reliable communication between the agents. The knowledge of others is taken into account with this model. Thus knowing another sensor is to know his triplet `<id,role,group>`. `Role` is `REPRESENTANT` or `SIMPLE MEMBER` or `CONNECTION`. A group is identified by the representant identifier.
This analysis is in fact the result of several iterations. In a first time the organization is not defined: a neighbour is identified only by its id. Once the model MWAC chosen and therefore the organization defined, an agent will be identified by the triplet <id ,role, group>.
During the next iteration of this process, we will introduce additional data on others (list of neighbours, adjacent list of groups and list of route fragments) and the addition of data on itself (the role). A tree consolidated with the results of all these iterations can be found on figure 7.
Defining the communication primitives. The communication primitive are the one proposed by the MWAC model (sending function, receiving messages).
Defining collaborative actions/perceptions. No more collaborative actions/perceptions are required than defined in the MWAC model.
Organizational aspect. An organization is inherited from the MWAC model.
When the system is being instrumented, there is no special requirement in organizations and in interactions.
However, in a configuration phase, it is necessary to define a protocol for the communication between an agent and its neighbours. Indeed, this mode requires that an agent knows how many other agents sensors are reachable. We define a communication protocol (Fig. 9).

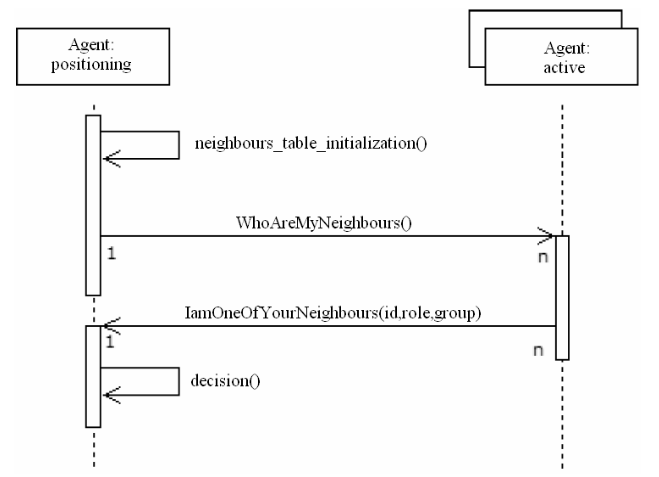

Agent:
positioning
Agent:
active
neighbours_table_initialization()
WhoAreMyNeighbours()
1
n
IamOneOfYourNeighbours(id,role,group)
1
n
decision()

Fig. 9. A simplified communication protocol

At this stage it is therefore necessary to start a new iteration of the previous process:

- Adding to the agent of a button because the speleologist needs to define the mode (positioning mode or not). The complete context diagram is shown on figure 10.
- Adding to the agent the data to memorize the mode or not (the complete data/capabilities tree is visible in figure 11). A privitive/action scheme is shown on figure 12.

Integration phase

Identifing social influence. In a positioning mode, agents must refuse to join a group. Indeed, it follows many re-organizations. These re-organizations lead to unnecessary energy losses.

Defining the social behaviour of the agent. During the positioning mode, we require to degrade the interaction protocols inherited from the MWAC model. We just have to correct this problem, to modify the decision module of the agent he chooses for any role. It aborts the smooth running of the MWAC inherited protocol.

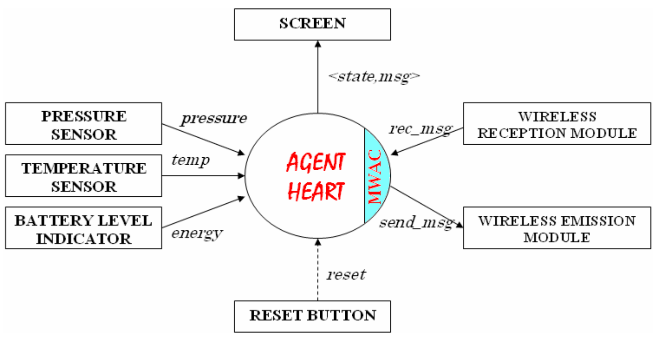


Fig. 10. Context diagram of an agent

```
Normal measuring phase mode
  ├─Data
  │   ├─On himself
  │   │   ├─Id
  │   │   ├─Power level
  │   │   ├─Distance from work station
  │   │   └─Positionned or not
  │   ├─On neighbour
  │   │   └─Neighbour table
  │   └─On environment
  │       ├─Pollutant rate
  │       ├─Humidity rate
  │       └─Etc...
  └─Task
      ├─Communication
      │   ├─Frame emission
      │   └─Frame reception
      ├─Decision
      │   ├─Connection quality emission
      │   ├─Neighbour informations update
      │   └─Energy level alert and correction
      └─Perception
```

Fig. 11. Consolidate tree of capabilities and building representation of self and the environment (iteration between the individual phase and the social phase)

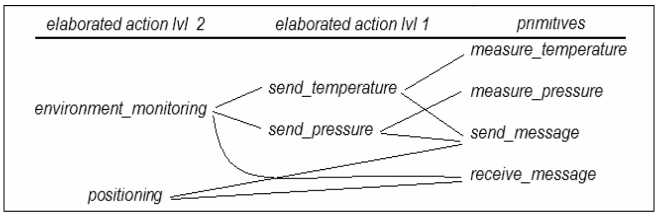


Fig. 12. Primitives and actions scheme

7. CONCLUSION

We have proposed in this paper a method to approach domains where collaborative entities are connected to physical devices they have to control or to supervise like in wireless instrumentation systems. We examined how a traditional multiagent design cycle must be enriched in this context.

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

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 languages are operational. The VDHL specification generation is partially developed.

Model wireless instrumentation systems are a recent field. Very few works address the problem of the analysis of self-organized instrumentation systems. This work proposes some innovative contributions in term of hybrid software/hardware multiagent lifecycle.

REFERENCES

[1] J.-P. Jamont et al., A Multi-agent System for the instrumentation of an underground hydrographic system, 2002 IEEE International Symposium on Virtual and Intelligent Measurement Systems - VIMS'2002, IEEE Instumentation and Measurement Society, 2002.

[2] Y. Demazeau, "Steps towards multi-agent oriented-programming," 1st International Workshop on Distributed Artificial Intelligence and Multiagent Systems, 1995.

[3] M.J. Wooldridge, "Intelligent agents," in Multiagent systems: A modern approach to Distributed Artificial Intelligence, Gerhard Weiss Ed., MIT Press, 1999.

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

[5] S. Moreno and E. Peulot, Le GEMMA, Casteilla, ISBN 978-2713517525, 1997.

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

[7] J. Adams et al. The design of mixed hardware/software systems. Las Vegas, USA, june 1996. ACM.

[8] M. Zwolinski, Digital System Design With Vhdl, Prentice Hall, ISBN 978-0130399854, 2003

[9] J.-P. Jamont, M. Occello and A. Lagrèze. Management of communication in wireless instrumentation systems: a solution based on a multiagent approach, Proceedings of the 12th IMEKO TC1-TC7 joint Symposium on Man, Science & Measurement, IMEKO.