Meeting the challenges of decentralized embedded applications using multi-agent systems

Jean-Paul Jamont, Michel Occello

Meeting the challenges of decentralized embedded applications using multi-agent systems

Jean-Paul Jamont*

LCIS - Université Grenoble Alpes
50 rue Barthelemy de Laffemas
BP 54 - 26902 Valence Cedex 9
E-mail: jean-paul.jamont@lcis.grenoble-inp.fr
* Corresponding author

Michel Occello

LCIS - Université Grenoble Alpes
50 rue Barthelemy de Laffemas
BP 54 - 26902 Valence Cedex 9
E-mail: michel.occello@lcis.grenoble-inp.fr

Abstract: Today embedded applications become large scale and strongly constrained. They require a decentralized embedded intelligence generating challenges for embedded systems. A multi-agent approach is well suited to model and design decentralized embedded applications. It is naturally able to take up some of these challenges. But some specific points have to be introduced, enforced or improved in multi-agent approaches to reach all features and all requirements. In this article, we present a study of specific activities that can complement multi-agent paradigm in the ”embedded” context. We use our experience with the DIAMOND method to introduce and illustrate these features and activities.

Keywords: multi-agent systems; collective intelligence; embedded multi-agent systems; agent oriented analysis; agent design; collective cyber-physical systems; real world applications.

Reference Jean-Paul Jamont and Michel Occello (20xx) Meeting the challenges of decentralized embedded applications using multi-agent systems, *Int. J. of Agent-Oriented Software Engineering*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes:
Jean-Paul Jamont is an associate professor at the Université Grenoble Alpes, France, where he teaches computer science and network systems. He heads the Networks & Telecommunication department of the Intitute of Technology of Valence. His research focuses on embedded multiagent systems (methodology, models, and architectures) and, more generally, collective cyber-physical systems. Jamont has a PhD in computer science from the Grenoble Institute of Technology. He is in the board of the

French section of the IEEE Systems, Man, and Cybernetics (SMC) Society.

Michel Occello is a professor at the Université Grenoble Alpes in Valence (France). He obtained his PhD in Computer Science from the University of Nice-Sophia-Antipolis in 1993 for a work about the blackboard metaphor applied to control systems. After a position of research Assistant in I3S CNRS laboratory in this university, he joined the CNRS IMAG/Leibniz Laboratory in Grenoble as an Associate Professor. He obtained his ”Habilitation à Diriger les Recherches” degree from the Université Joseph Fourier in Grenoble in 2003 on the theme ”Methodology and architectures for multi-agent design”. In 2002, he contributed to launching the activity around multi-agent systems in the LCIS lab in Valence. He is now the leader of the Cooperative Complex Systems (COSY) team and Director of this laboratory. His main fields of interest are multi-agent systems methodology, adaptive self-organizations and recursive architectures, and the application of multi-agent systems to Collective Wireless Embedded Systems and New Software Technologies. He has published more than 70 papers for international conferences or journals focused on multi-agent methodologies and architectures, and their applications.

1 Introduction

For the last decades, embedded systems have been small-scale and standalone. New application domains have emerged from the new wireless technologies. They are called pervasive computing, ubiquitous computing or the web of things. They are a superset of a lot of specific domains as home automation, remote monitoring systems, collective robotics, wireless mobile sensor networks. These systems are composed of heterogeneous systems strongly related to their environment. Scalability and openness introduced by a massive deployment of these systems are significantly increasing their complexity.

A multi-agent approach can be highly profitable for designing these artificial complex systems. However, some features in existing multi-agent design life cycles and models have to be enhanced to take software/hardware hybridization particularities into account. Studying the design of embedded systems using multi-agent paradigms is a recent research field.

The application of multi-agent paradigm to embedded systems makes different types of requirement emerge. They concern the life cycle, specific activities at analysis and design levels, models and implementation. This paper presents a discussion about the specific aspects required to design decentralized embedded applications using decentralized artificial intelligence. These activities are involved in the DIAMOND method (Decentralized Iterative multi-agent Open Networks Design Jamont and Occello (2007)), we have developed for the design of embedded complex systems using multi-agent system (MAS).

The remainder of the paper is organized as follows. Section 2 introduces networked embedded systems and presents the challenges we have to face to develop them. Section 3 gives an insight into how multi-agent systems cope with these challenges. Section 4 presents the requirements that multi-agent methodologies must meet to ensure a satisfactory coverage of embedded multi-agent systems development. Each requirement is presented as a question. For each question, we present the problem, the related works in multi-agent systems and we provide our position, even if the paper is not dedicated to the presentation of DIAMOND. Finally Section 5 provides both a discussion about the relevance of MAS approach to design embedded system and an evaluation of DIAMOND. A conclusion ends the paper by making a synthesis of this study.

2 Networked embedded systems

In this section we aim at clearly defining the notion of networked embedded system in order to clarify the argumentation proposed in the paper about:

- the adequacy of multi-agent systems for modeling decentralized embedded applications,
- the particular aspects required for embedded multi-agent models and methodologies.

It is first of all necessary to precisely define what we call ”embedded systems” and ”networked embedded systems”. We then propose to expose a set of features that are specific to embedded and networked embedded systems and to come to the main challenges they imply. These definitions and challenges are the synthesis we made from some studies and reviews of the domain (Lee (2002); Elmenreich (2003); Henzinger and Sifakis (2006); Pottie and Kaiser (2009); Zurawski (2009)) and from our own experience.

2.1 Embedded systems and software

Embedded systems can be defined as dedicated systems built to handle one or a few pre-established tasks in interaction with the physical world. They are conceived using software built into or ’embedded’ within a device.

Embedded systems are used in cars, telephones, audio equipment, robots, appliances, toys, security systems, pacemakers, televisions or digital watches, and their architectures are often constrained by the use of inexpensive microprocessors and limited storage.

Embedded software is thus not just software on small computers. Due to the application size and constraints (interaction with the physical world or response time for example), the software can be partly ’built into’ the electronics. In this case, the program is written permanently into the system’s memory, rather than being loaded into RAM (Random Access Memory) like programs on a personal computer. Specialized architectures such as FPGA (Field-Programmable Gate Array) or SoC (System on Chip) can be used in this context.

Such systems must be endowed with some features, essential to carry out their main required functionalities.

(1) *Reactivity.* Embedded systems have to react continuously to their environment because of their strong relation to the physical word. Embedded systems must be able to decide to adapt themselves to changing conditions. Some of their services, resources or sensors can appear and disappear. To maintain its quality of service or even only its functional integrity, a system must be able to dynamically take decisions while it operates. The challenge is to react simultaneously to multiple stimuli and to change its behavior (or even its structure) without redesigning, recompiling or just stopping the system.
For example, when a robot executes a plan to close an opened door, it has *to react*, i.e. to interrupt its plan, when this door is closed by another robot.

(2) *Timeliness.* Being reactive is not enough for embedded systems. Physical processes with which they interact are evolving over time. To guarantee real-time features, timeliness has to be associated to reactivity. The challenge for embedded software practitioners is to offer sufficient time control abstractions to guarantee that the software will evolve at the speed of the environment. Dealing with time is critical for the global safety of the whole system.
Typically, as an example, an interception mission imposes to act in a certain time window.

(3) *Liveness.* Embedded computing may not accept premature termination of programs. Deadlocks have to be avoided at all costs. We cannot consider that correctness is obtained only if the right answer is given for a set of input data. The system must be able to adapt its behavior even by giving a not totally satisfying answer or a partial answer taking timeliness into account. It has to take into account the timeliness of a continuing stream of partial answers, as well as other non-functional properties.
A mobile robot has to generate an alternative path in time to avoid an obstacle even if this path is not the best one.

(4) *Power Autonomy.* Embedded systems are often deployed on mobile small-sized devices for which energy supply is not simple. To remain operational as long as possible, embedded software has to manage the tasks according to their importance and available energy resources. The challenge is to integrate the energy criteria as well as possible in the decision cycle, commuting from full operational to minimal security modes. Saving energy in the passive mode and searching for efficiency in the active mode determine the overall performance capability of the systems.
A grounded vehicle must find a good compromise between sending position at a high frequency - to ensure a moving research vehicle in the vicinity can detect it - and sending position with a more important interval to be detectable for a long time.

(5) *Safety management.* As they are strongly related to the physical devices, embedded systems can perform dangerous actions for humans or their environment. Leveson (2004) argues that accidents in complex systems arise due to poorly understood or dysfunctional interactions between humans and machines. The challenge is thus to take hazard and risks into account during both design and operational running.
A robotic arm carrying an important load must hold its position even if an

emergency stop is triggered, because a human operator can be blocked under the arm.

2.2 Networked Embedded systems

Today, using advances in networking technologies, sets of embedded systems can be used together to achieve complex tasks.

These networks are called networked embedded systems; they constitute the core of decentralized embedded applications. Building networked embedded systems (NES) makes some complementary features appear as requirements for the programming abstractions used for embedded software.

(6) *Complexity.* Networked embedded systems can become very sophisticated applications (like airplanes or process control systems) involving devices, instruments or large engineering structures/systems. Building a global model is often impossible for these systems. Making them more intelligible leads to try decentralized approaches.
In an A380 aircraft (Salzwedel, 2011), for example, more than 5000 electronic control units are networked and 50 calculators work concurrently. In the context of the fly by-wire technology (FBW) a decentralized architecture has been developed to reduce the load of the central control computer.

(7) *Concurrency.* Embedded systems interact with a lot of physical processes, controlling several actuators and interacting with humans. They must simultaneously react to stimulus from a variety of sensors. Multiple things can happen at once in the physical world and embedded systems must face all of them at the same time. Networked embedded systems are distributed among a network. Hence embedded software must also be able to manage task coordination between nodes. Supervising and controlling diverse devices implies that embedded software must be concurrent.
For example, to ensure redundancy in detection systems, several sensors must be scanned concurrently and sometimes give contradictory measures that the system must merge.

(8) *Heterogeneity aggregation.* Heterogeneity is an intrinsic part of computation in embedded systems since embedded systems are a mixture of hardware and software. Embedded software has to interact with specific hardware devices. Other kinds of heterogeneity can be found in embedded systems such as continuous/discrete time control techniques, real-time actions and longer-scale processing or synchronous/asynchronous event handling. Concerning NES, an objective is to ensure nodes interoperability and to characterize the behavior of the whole system. Due to the different kinds of heterogeneity, seeking a global model is often impossible. It’s necessary to combine multiple models to fit any given problem better. The challenge there is to talk about the properties of the aggregate.
Practically, a home automation system (with effectors and sensors) will have to be addressed as a whole by a user using a comfort service.

(9) *Interfaces integration.* Embedded Systems have to integrate or compose concurrent systems and services. As a consequence of the interoperability

requirements embedded software (and hardware) must provide frameworks that make the coordination of the components possible. To achieve this coordination sophisticated interactions are required and interaction mechanisms must be supported by these frameworks. Each node can be addressed through its interface specifying external available interactions. However, classical communication techniques are extremely weak for achieving elaborated interactions. The interaction models of classical programming paradigm like Remote Procedure Calls or even Object-Oriented Programming are imposed and fixed. The challenge for embedded software is to benefit from a component technology that includes enhanced flexible properties in interface definitions.
For example, interoperability bricks have to be embedded to allow interaction between a set of drones and a wireless ground sensor network.

(10) *Reconfiguration/self-organization.* Previous features lead to the fact that networked embedded systems are subject to frequently adapt their behavior according to their interactions and their local tasks in order to achieve the global objective of the system. The management of the decentralization is a challenge because, rather than using highly complex monolithic stand-alone tasks on powerful devices, NES uses numerous communication-centric small devices operating as a collective. Embedded software must thus be agile, self-organizing and critically resource adaptable (Culler et al., 2001).
As an example, we can take the manufacturing flexible cells able to be reconfigured to adapt a new process by introducing tool pro-activity.

(11) *Mobility.* In the case of networked embedded systems using wireless communication, embedded software must manage the mobility of the nodes. Nodes typically coordinate their behavior by agreeing on a set of actions or on a common view of their environment. With wireless (and possibly ad hoc) networks, communications are sometimes unreliable and the achievable performance greatly varies over time and location (Bouroche and Cahill, 2008). The challenge here is to allow coordination approaches to be applied despite of communication failures.
For example this property will be very important for the coalition of drones exploring areas for search and rescue.

(12) *Integrity.* Maintaining integrity of classical networked embedded systems essentially lies in ensuring functional integrity through fault tolerance mechanisms. How can the system continue to run in case of node failures? Advanced systems nodes, however, involve elaborate behaviors and manipulate a large amount of data which are vulnerable to both node and communication failures. The challenge lies in the confidence that each node can have in its partners and in how the system can resist external attacks (Ganeriwal et al., 2008). For example, to take right decisions nodes use data that must be completed by accuracy and confidence information their sensor must supply. In data collection systems, faults are indicators that sensor nodes are not providing useful information. In data fusion systems, the consequences are direr; the original outcome is easily affected by corrupted sensor measurements making fault detection less obvious.

This problem of confidence in data and behavior is crucial for example in military or surgical embedded devices.

In the following parts, we will assume that decentralized embedded applications can be developed using networked embedded systems by emphasizing their cooperation abilities. We will argue that MAS are a good solution provided that their embedded dimension should be considered to become embedded MAS.

3 Embedded multi-agent Systems

In this section we discuss reasons making a multi-agent approach interesting to design decentralized embedded applications. We introduce some real world embedded multi-agent applications.

3.1 Why are multi-agent systems well-suited to design such systems?

Networked Embedded Systems are constituted by numerous autonomous execution units achieving tasks of control, of communication, of data processing or acquisition (Challenge (7)). The units are related through linked or wireless networks. The global task of the system requires cooperation between units. Units could be viewed as artificial complex systems involving nodes autonomy for decision and energy management, for services or data exchanges, or to emphasize a collective activity to supply collective services. They are strongly related to their environment constituting cyber-physical systems.

The multi-agent paradigm offers a powerful mechanism for autonomous behavior, social organizational and cooperative exchanges needed for these kinds of artificial complex systems (Challenge (6)).

An agent is a hardware/software entity evolving in an environment that it can perceive and in which it acts. It is endowed with autonomous behaviors and has objectives.

Autonomy is one of the main concepts in the multi-agent issue: it is the ability of agents to control their actions and their internal states. Adaptation allows an agent to reason about the quality of its work according to constraints or incomplete data (Challenge (3)). The autonomy of agents implies no centralized control at a system level. An agent can be endowed with communication capabilities.

Other approaches like Autonomic Computing (AC) (Kephart and Chess, 2003) address the complexity and evolution problems in software system, based on autonomy. A software system that operates on its own or with a minimum of human interference according to a set of rules is called autonomic.

Autonomic systems are interactive collections of autonomic elements. Self-management as a whole is obtained by orchestrating a well-coordinated communication of self-manageable autonomic elements or components (Hassan et al., 2009).

The principles of AC are mainly inherited from system management while multi-agent principles are inherited from collective artificial intelligence. Autonomic systems can benefit from multi-agent properties (Tesauro et al., 2004; Huebscher and McCann, 2008).

Table 3.1 presents a comparison of the two approaches.

	Autonomic computing	Multi-Agent Systems	Networked Embedded Systems	Embedded Multi-Agent Systems
Applications	• system management and adaptation • integration	• system control • problem solving • simulation • integration	• system control	• system control • integration
Approach	to build and manage systems	to model and decompose problems	to design systems	to model and build systems
System goal	expressed global goal	emergence from local goals of a non necessarily expressed collective behavior	non necessarily expressed global goal obtained through communications	non necessarily expressed global goal for a cyber-physical society
System Management	• self-* without hypothesis about how they are achieved • functional integrity maintenance according to global policies	• usually not global policies • not designed for self-healing neither for self-protection, strategies can be defined in agents • very few mechanisms for robustness	• robustness above all • resilience techniques	• improved resilience and robustness • self-* non global policies
System Evolution	element updates	• agent learning, • social status changes	reconfiguration	• individual and social adaptation
System Environment	no physical environment except users	virtual or real environment	physical environment only	virtual and real physical environment
Element Architectures	user services or data managers	reasoning architecture	software/hardware	embedded collaborative decision
Element interactions	• data exchange • services call	• social multilateral interactions • variable interaction modes available	hardware defined communications	• enhanced hardware based interaction protocols • interaction modes flexibility
Element behaviors	data, services, resource managers	• reasoning about environment, other entity, themselves • cognitive interpretation of the environment	• control loops	enhanced embedded decision

Table 1 Comparison of Autonomic Computing and Multi-Agent Systems for NES

MAS aims to build collective intelligence systems. Nodes are not only autonomous but rather social entities. Agents are able to communicate, but also to interact adapting their interaction mode dynamically. The objective is not only to manage a system or to make a system adaptive but also to produce a collective emergent behavior.

The emergence paradigm of MAS (Muller, 2004) deals with the non explicitly programmed and irreversible sudden appearance of phenomena in a system. The emergence process is a way to obtain dynamic results, from cooperation, that cannot easily be predicted in a deterministic way. Emergent phenomena can be global structures or collective behaviors that can be observed by an external observer.

Designing a MAS leads to find a way to build local agent structures and behaviors to drive the system of agents that produces a particular global structure or a particular global functionality.

The traditional method consists in an individual centered decomposition of a problem. The multi-agent alternative aims at making this functionality emerges in a controlled way from the interactions between the agents.

MAS is traditionally used for problem solving, simulation, system integration or system control. It is obvious that the last two ones are concerned in an embedded context. Simulation can also be an advantage in this context to ensure the collective tuning. The objective is to agentify embedded nodes in order to realize the management and the supervision of an embedded networked system by a user, who can be considered as the observer of the phenomena. Considering that agents can be implemented as software/hardware hybrid entities from embedded software up to full physical units leads to define the notion of embedded multi-agent systems (*eMAS*).

Works are currently done to apply autonomic systems to embedded systems as it is done for multi-agent systems (Chun et al., 2010). The explicit modeling of a common environment for agents is an advantage of multi-agent systems in the context of cyber-physical systems like NES.

Finally, we will retain the advantages of this eMAS for NES:

- at design level:
  - simplification of the system design. Exploiting weak coupling between agents reduces the whole system complexity (Challenge (6)).
  - compliance with resource limitations and with challenges (6) and (8) underlying constraints. Giving a complete explicit model of the whole system to agents is no longer necessary.
- at running level:
  - reinforcement of the robustness of the system. The system becomes able to create new behaviors or new organizational structures to adapt itself to unexpected situations at the design time (Challenge (10)). The system becomes less sensitive to environment changes (Challenges (1) and (12)).
  - improved adaptation to mobility (Challenge (11)) by the inherent weak coupling of agents.

- at exploitation level:
    - monitoring by an external observer. An external explicit representation of interaction schemes and of organization structures of a system can be obtained (Challenge (9)).

3.2 Real world embedded multi-agent applications

Since the 2000s, multi-agent system technologies seem to be sufficiently mature for an industrial context (Parunak, 2000). Munroe et al. (2006) outline that only a small number of industrial sectors were impacted and they considered MAS industrial users as visionaries. Ten years later affected sectors are the same (Leitao et al., 2013; Müller and Fischer, 2014).

Concerning eMAS, as it is difficult to propose a classification of industrial applications, we will just quote some works in active areas where eMAS could be involved:

- Collective robotics. It is one of the most popular embedded applications that can be modeled with multi-agent systems (Huang et al., 2001). Agents realize decisional software parts of robots. They ensure the coordination of physical parts and collective strategies. Multi-agent systems can be found in mobile robotics (Le et al., 2009; Takimoto et al., 2007; Jamont and Occello, 2013) or the manufacturing field (Monostori et al., 2006). In addition to more classical properties, mobility (Challenge (11)) and safety (Challenge (5)) are the main aspects to develop specifically for these domains.

- Ad-hoc networks and wireless sensor networks. They are very successful applications of MAS where self-organization is especially emphasized. MAS can now improve the evolution of sensor networks by their integration (Challenge (9)) and heterogeneity aggregation (Challenge (8)) (capabilities for data merging, filtering, data access and intelligent routing) (Hla et al., 2010; Jamont et al., 2010).

- Transportation and logistics. Research in traffic simulation or logistical planning is a very active field for MAS but many works deal with simulation. Right now, sharing efforts with mobile robotics and ad-hoc network agent approaches can lead to very powerful real world deployments in this field (Chen and Cheng, 2010).

- Home automation. Comfort and security for home has up to now been a control theory problem. The growing number of devices and interactions in organizations brings theoretical limitations. The size and complexity of physical models continually increase, implying an exploitation of decentralized capabilities (Jamont and Occello, 2011). Home automation today requires heterogeneity integration (Challenges (9) and (8)), power management (Challenge (4)) and self-organization (Challenge (10)).

- Automated surveillance. The development of automated surveillance resides in both infrastructure management and scene analysis. Complexity of processing with real time constraint can impose a decision decentralized among nodes.

This field can benefit from MAS (Carrasco et al., 2010) particularly exploiting decentralized decision (Challenge (7)), integrity management (Challenge (12)) and data integration (Challenge (9)).

- RFID applications. Agents have recently been used to develop RFID applications as identity or location managers (Massawe et al., 2009). But RFIDs introduce the concept of remote behavior or service. Using RFIDs and agents can lead to giving reactive behaviors (Challenge (1)) to passive objects opening an innovative way of creating virtual societies associated to physical intelligent environments.

Most of the industrial applications require a wide range of functionalities. They need to purely cover software systems to perform planning, scheduling, simulation or other decision support like functionalities. They impose solutions that are closely related to some kind of hardware providing control, diagnosis or integration functionalities. This is especially true for manufacturing, collaborative robotics, or networking, where the arguments in favor of a deployment of decentralized computation solutions are stronger than for isolated software applications.

However researches in eMAS have not necessarily given rise to a deployment in the real world. Pechoucek and Marík (2008) studied the industrial deployment of multi-agent technologies. The authors identified the hardware integration of agents as a key obstacle for wider deployments. An abstract of their study is reported on Table 2 showing the crucial importance of the works about embedded multi-agent systems. They emphasize also in their work that agent technology can play an important role in embedded applications if it is given the ability to be closely linked to hardware devices.

Domain	Hardware integration
Manufacturing control	+++
Logistics	+
Production planning	++
Simulation	+
UAV control	+
Space exploration	++
Training	-
Automotive	++
Supply-chains	-

(-): Hardware integration is not a key requirement
(+,++,+++): Hardware integration is - one of the key requirements, an important key requirement, a crucial key requirement for - the considered area.

Table 2 Hardware Integration Importance for multi-agent Industrial Application Domains (from (Pechoucek and Marík, 2008))

4 Challenges for embedded multi-agent systems methodologies

Discussing methods for the implementation of intelligent solutions for embedded systems (Elmenreich, 2003) claims that multi-agent systems are able to implement intelligent functions for embedded systems but that many challenges occur according to dependability, real-time requirements and to constraints of cost, size, and power consumption. In part 2, we established some challenges relative to networked embedded systems. We have seen in the previous section that some of these challenges can be directly addressed through the nature of multi-agent systems. In this section we will attempt to show how multi-agent systems methodologies can be improved to address the remaining exposed challenges.

We draw our experience from the application of DIAMOND to numerous real world projects that validate it. They allow us to introduce specific activities in DIAMOND that we will use to illustrate the advances we propose for multi-agent methodologies.

Table 4 lists the different kinds of applications.

Id	Project name	Type of funding	Theme	References
(A)	PULSER	EU	Sensor networks	Occello et al. (2008)
(B)	ENVSYS	Industrial	Sensor networks	Jamont et al. (2010)
(C)	KURASU	Industrial	Robotics	Jez (2011)
(E)	SIET	Industrial	Home care	Raïevsky et al. (2014)
(F)	PALETTE	Industrial	Robotics	Jamont et al. (2014b)
(G)	VAICTEUR AIR2	Industrial	Home energy monitoring	Jamont et al. (2011)
(H)	ASAWOO	ANR	Web of things	Jamont et al. (2014a)

Table 3 List of real world applications of DIAMOND

DIAMOND was validated in several real world trial or industrial projects.

The VAICTEUR AIR2 project (G) (Jamont and Occello, 2011) proposes a building comfort multi-agent control with a generic physical model of house temperature evolution. In this model, a house is defined as a set of rooms. The room models are connected using an internal building equation-based model constituting a realistic simulated physical environment.

PALETTE (F) is a technology transfer project dealing with an application of collective robotics for palletization in a manufacturing process. The aim is to optimize the process through the use of collective robotics (Jamont et al., 2014b). The purpose of the industrial ENVironment SYStem project (B) (Jamont et al., 2010) is to monitor an underground river network. In an underground river system the installation of wire communication networks is difficult, especially because the structure of hydrographic systems is very often chaotic. In the case of a radio communication network, the underground aspect complicates wave propagation and for the moment the techniques that are used are not totally mastered. The general idea of the project is to propose a sensor network from the existing physical wireless layer. DIAMOND was used to build a node architecture, an intelligent routing and a functional integrity maintenance of the sensor network.

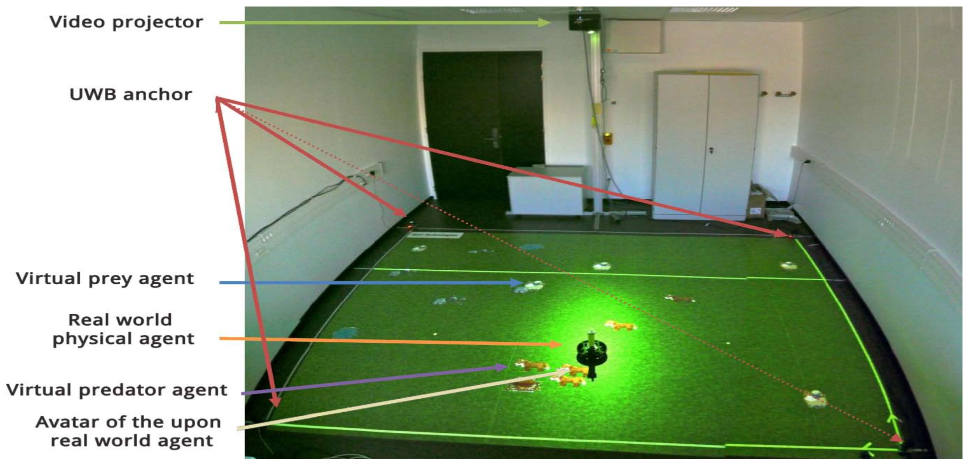


Figure 1 A prey/predator simulation mixing one real world predator agent and two virtual prey agents

A collaboration with the CEA-LETI Grenoble, a work for the PULSER Project (MEDEA+ 2A204 European Project) (A) was to develop a system allowing to follow the evolution of a rescue team, inside a building (making GPS unusable), using entities guaranteeing the lowest power consumption. PULSER was interested by the development of a generic architecture for wireless sensors based on low-cost silicon devices involving Ultra Wide Band (UWB) location. UWB is a technology able to ensure indoor location that uses less energy.DIAMOND was used to produce mobile node software architectures (Occello et al., 2008).

Another application has been made conjointly with the CEA-LETI/Grenoble laboratory in the Kurasu project (C) using an UWB location system(Jez, 2011). A physical robot interacts with virtual robots projected in its physical environment. The preys are virtual agents. The robot controller is built on a Virtex 4 FPGA chip (Xilinx XC4VFX20) which has a PPC405 microprocessor logical block. The position of the robot is measured with an experimental UWB location system.Figure 1 presents an experiment on a prey/predator hybrid simulation.

This principle is used in the ASAWOO (H) ANR project. It has for objective to enhance appliance integration into the Web. The project proposes an architecture to provide users with understandable functionalities under the form of WoT applications, while enabling collaboration between heterogeneous physical objects, from basic sensors to complex robots using MAS (Jamont et al., 2014a; Mrissa et al., 2015).

The SIET project (E) aims at evaluating the benefits that adapted tablet computers can bring to dependent people in specialized institutions or at home. The SIET project especially aims at improving elderly people's wellbeing by giving them means to communicate more easily with their caregivers (family and health workers) using information technologies. In order to improve dependent people's safety and social network dynamics, we added agents to these tablets (Raïevsky et al., 2014; Mercier et al., 2013).

4.1 What life cycle to support the design of such a system?

Problem. Designing embedded systems traditionally starts by a system requirement analysis followed by a partitioning step (Figure 2a): a hardware requirement and a software requirement are induced from the global system requirements. Generally, software is used for its flexibility, while hardware is used for its performance. Then, the hardware part and the software part are developed concurrently and are integrated at the end of the process.

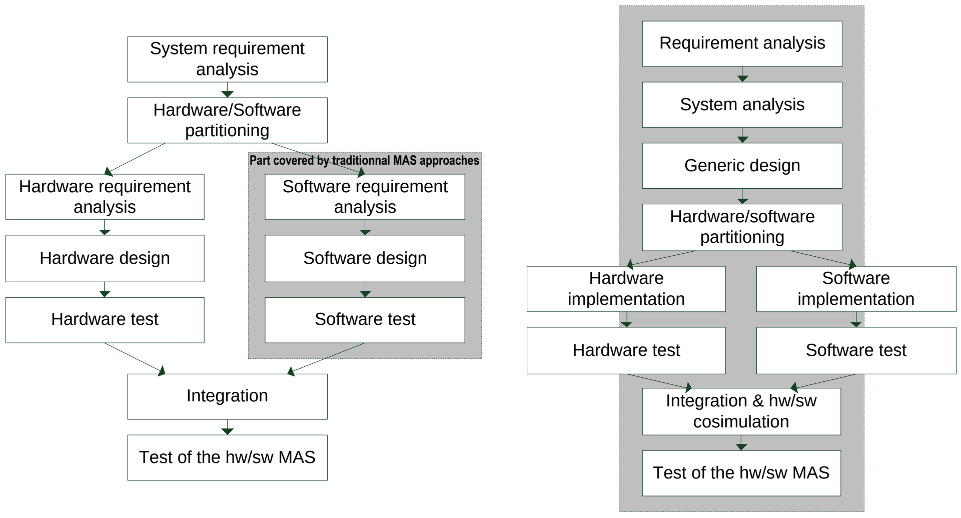
It is therefore necessary that the life cycle enables late specification changes. Furthermore, the criticality of the applications requires to find the best hardware/software (hw/sw) design trade-off (to decrease response time (Challenge (2)), to limit power consumption (Challenge (4)), to increase fault tolerance (Challenge (5)...) and require returning to previous design steps (refinement). The design process must accept genericity (incremental criteria are in favor of genericity). Finally, we must identify and keep a trace of all the parameters of the different chosen solutions.

MAS related works. Most existing multi-agent methods usually distinguish only analysis and design/implementation phases (as for example MASE (DeLoach et al., 2001) or Gaia (Wooldridge et al., 2000)). Very few methods deal with other phases. In particular very few methodological works focus on deployment i.e. integrating agents in an operational environment. In the context of classical (non-embedded) agent applications, the main objective of deployment is to place agents into facilities. In this context, we can find for example a deployment phase in MASSIVE (Lind, 2001). We can also quote the contribution of (Braubach et al., 2005) proposing a prototype for a deployment tool (ASCML). The deployment phase includes the specification of the physical architecture of the system and how software is to be deployed on it. In our particular field, it takes a great importance since it includes the hardware/software partitioning.

Concerning process, the most current life cycle used in multi-agent methods remains the classical cascade life cycle, even if the need of iterative and incremental process is recognized today (Cernuzzi et al., 2005). Methodologies that devote attention to deployment try to position the coding phase somehow late in the process in order to be as flexible as possible. Some of them even try to make the cycle able to dynamically take into account late requirement modifications by introducing agile capabilities like in Agile-PASSI (Chella et al., 2006).

Our proposal. We adopt a hardware/software co-design approach to build the MAS in order to meet embedded requirements. Hw/sw co-design approach is an alternative to a traditional embedded system development life cycle. A co-design method unifies the development of both hardware and software parts by the use of a unified formalism. The partitioning step, which refers to the mapping of functions using an instruction set architecture (software) and a logic block architecture (hardware), is pushed back at the end of the life cycle.

This approach is interesting because we can question the hw/sw partition (Figure 2) more easily than traditional hw/sw system design and development approaches. It enables a more efficient search space exploration for the partition and it allows to produce a design specification meeting both performance criteria and functional requirements.



a. Traditional embedded MAS approach b. DIAMOND hw/sw codesign approach

Figure 2 Hardware/Software co-design life cycle used in DIAMOND to design an embedded system

Our embedded multi-agent design approach uses five main stages. They are distributed on a spiral shaped life cycle (Figure 3). The *requirement analysis* defines what the user needs and characterizes global functionalities. The second stage is a *multi-agent-oriented analysis* which focuses on decomposing a problem in a multi-agent solution.

The third stage of our method starts with a *generic design* which aims at building the multi-agent without distinguishing hardware and software parts. The fourth stage enables to define the partitioning criteria and to define the simulation models and parameters which will be used to test the hardware/software implementations of the multi-agent system.

Finally, the *implementation stage* aims at partitioning the system in a hardware part and a software part to produce the code and the hardware synthesis.

4.2 How to define the requirements for designing such systems?

Problem. This section deals with special requirements that have to be taken into account to take some challenges up. As seen in the previous part a major specificity in this context is the consideration of deployment. The designer has some specific characteristics for their final system in mind when they specify it. Physical

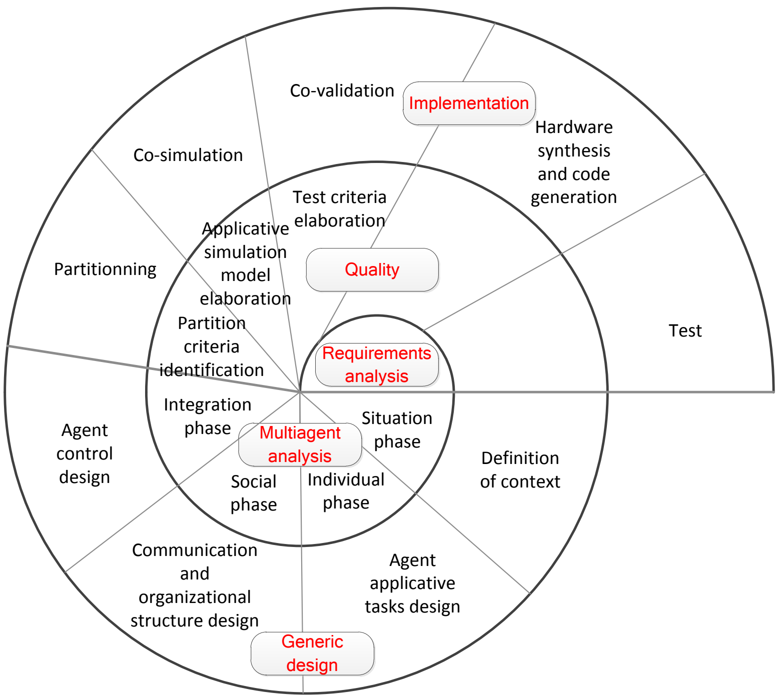


Figure 3 Life cycle of the DIAMOND method

architecture requirements must be expressed at a specification level under the form of expected qualities. Designing embedded multi-agent systems implies to deal with non-functional requirements (NFR) related to deployment. Other aspects to study, related to the physical context, are the requirements in terms of safety (Challenge (5)). By safety we mean the guarantee that the system will not be dangerous for the human user even in a degraded running mode.

MAS related works. Most of multi-agent methodologies only focus on functional requirements whereas designing embedded multi-agent systems imposes to handle NFR.

Dealing with the consideration of NFR and the way the designs are driven by these abstract requirements is very recent. Some of the multi-agent leading methodologies attempt to integrate some techniques in their latest extensions like (Harmon et al., 2009) with O-Mase or (Blanes et al., 2009) with RE-Gaia. Inspired by goal-oriented requirements works (Mylopoulos et al., 1999; Liu and Yu, 2004), Tropos (Bresciani et al., 2004) proposes to build an incrementally refined model of the system where NFR are seen as specific goals.

ADELFE (Picard and Gleizes, 2004) takes some NFR into account (Werneck et al., 2007) in its third activity called "Define Consensual Requirements" regarding storage of large data volumes, capacities of human-machine interaction, and system availability under the form of pre-requisites. Some running constraints are included in this requirement definition as distribution or multi-task capabilities.

One of the main problems is to express the requirements in order to simplify their specification, to make them reusable and to consider them all along the life cycle. Some tools have been proposed involving ontology-driven techniques (Lindoso and Girardi, 2006) or model-driven approaches (Naji et al., 2004).

Concerning safety, the problem is rarely addressed. Some works focus their efforts on the security, such as the inviolability of the systems (Bresciani et al., 2004). They do not address physical applications and their risks for users. One may question how safety can be considered in multi-agent systems. Some studies about hazard in complex systems claim that the approach must use a risk-based whole-system model but that the analysis can be achieved only by evaluating the exposure to risk, either suffered by or caused by each entity in the system (Alexander et al., 2008). Our analysis also leads us to distinguish a global level concerning requirement specifications for the whole system and a local level concerning how the requirements will be achieved in the decentralized system at the design level.

Some researchers attempt to adapt classical techniques widely used in manufacturing industries to multi-agent systems. The failure mode and effects analysis (FMEA) is a step-by-step approach introduced to enable hazard identification and qualitative risk investigation. It helps to identify potential failure modes based on past experience with other systems. The objective is to study the effect of these failures and how they can affect the user. Ebrahimipour et al. (2010) proposed an agent structure and used it in a multi-agent system to ensure safety engineering by the means of fault diagnosis diagram trying to solve limitation of FMEA in complex systems or with a process with numerous components. In a work driven by the philosophy of FMEA, a visual language to express self-management aspects that leads to self-protection and self-configuration aspects is shown in (Rodriguez-Fernández and Gómez-Sanz, 2010). Sterling and Taveter (2009) present safety as a quality attribute for a multi-agent system. They propose to improve a multi-agent methodology using Hazard an Operability (HAZOP) Studies. The HAZOP approach is a systematic procedure for determining the causes of process deviations from normal behavior and their consequences of those deviations. Several industrial applications using multi-agent systems have involved some HAZOP rules (Lakner et al., 2006; Johnson, 2005).

Our proposal. The requirements definition should begin with an analysis of the physical context of the system. It includes activities such as main tasks identification and workflow characterization. DIAMOND uses UML (Jacobson et al., 1999) notations, well suited for studying user functional requirements.

The physical context in which a MAS is embedded requires identifying many particular possible behaviors: In what state should a component be when the system is under maintenance? How to calibrate the different physical system components as effectors and sensors? What should a component do when an emergency stop occurs? A substantial list of questions has to be addressed.

We introduce a new activity to take the embedded context into account and to structure the global running of the system. This activity (the *particular mode analysis*) allows to prompt design-oriented questions at the beginning of the project. It decreases the number of iterations in the analyzing phase by taking the possible human interventions from general functional requirements into account. Moreover this activity emphasizes a restricted running of the system. This activity allows to take the physical safety of the users, possibly plunged in the physical system, into account.

Starting from the GEMMA Guide (Adams and Paques, 1988) used by some GRAFCET designers, we have defined a tool which can be seen as a graphical checklist which allows the designer to define, from the real world context operations and their consequences for the designed multi-agent system.

We have defined sixteen different states grouped into three families called procedures (Figure 4). The *running procedures* (see Table 4) are related to the definition of the recognition states of normal start, normal running, tests while running procedures etc. The *stop procedures* (see Table 5) focus on the different procedures to stop the multi-agent system because of external reasons (such a lack of raw material in the case of manufacturing control). The *emergency procedures* (see Table 6) concentrate security procedures (for example allowing a human maintenance team to work on the system) or specific rules for restricted running.

The use of this tool during this early stage can seem to guide the designer towards a centralized resolution of the problem, but cooperative decision-making will be carried out to enable the detection of the transition conditions between states and to decentralize decision capabilities.

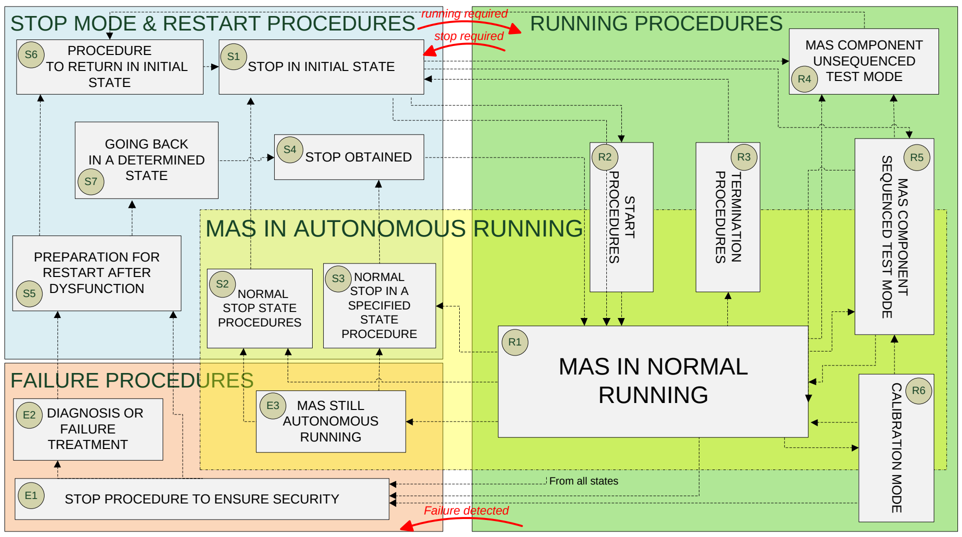

STOP MODE & RESTART PROCEDURES
running required
stop required
RUNNING PROCEDURES
S6 PROCEDURE TO RETURN IN INITIAL STATE
S1 STOP IN INITIAL STATE
R4 MAS COMPONENT UNSEQUENCED TEST MODE
GOING BACK IN A DETERMINED STATE S7
S4 STOP OBTAINED
R2 START PROCEDURES
R3 TERMINATION PROCEDURES
R5 MAS COMPONENT SEQUENCED TEST MODE
MAS IN AUTONOMOUS RUNNING
PREPARATION FOR RESTART AFTER DYSFUNCTION S5
S2 NORMAL STOP STATE PROCEDURES
S3 NORMAL STOP IN A SPECIFIED STATE PROCEDURE
R1 MAS IN NORMAL RUNNING
FAILURE PROCEDURES
E2 DIAGNOSIS OR FAILURE TREATMENT
E3 MAS STILL AUTONOMOUS RUNNING
R6 CALIBRATION MODE
From all states
E1 STOP PROCEDURE TO ENSURE SECURITY
Failure detected

Figure 4 Particular mode study in DIAMOND (from Jamont et al. (2014b))

As an example, in (F) , we consider a multi-agent solution that (1) gives manufacturing orders to workstations, (2) assigns operators to workstations depending on their qualification, (3) controls both robots that carry containers between workstations in the same workshop and from a workshop to another and (4) does not control machines/tools at workstations. Here we are interested in identifying different types of NFR like the safety, the availability and the recoverability. The synthesis of this capture is shown in Figure 5.

State	Description
R1	*MAS IN NORMAL RUNNING* This is the normal state of the MAS.
R2	*START PROCEDURES* This state concerns the operations which must be done before the MAS can start its autonomous running.
R3	*TERMINATION PROCEDURES* This state concerns the operations which must be done before the MAS can be considered as shutdown.
R4	*MAS COMPONENT UNSEQUENCED TEST MODE* This state allows to specify operations that some agents can do to be checked locally without following the MAS autonomous functioning.
R5	*MAS COMPONENT SEQUENCED TEST MODE* This state allows a more complex step-by-step test (in comparison with tests included in R4). Agents use their coordination capabilities here.
R6	*CALIBRATION MODE* This mode allows the agent actuators and agent sensors to be calibrated and adjusted.

Table 4 Description of running procedures

State	Description
S1	*STOP IN INITIAL STATE* In this state, agents are energized but their autonomous behavior is not switched on.
S2	*NORMAL STOP STATE PROCEDURES* In this state, MAS has been asked to stop during a time period when there is not global aim to achieve.
S3	*NORMAL STOP IN A SPECIFIED STATE PROCEDURE* The MAS has been asked to stop in a specific state recognition. As long as this state is not reached, the MAS continues its autonomous execution.
S4	*STOP OBTAINED* The MAS is stopped (not an emergency stop)
S5	*PREPARATION FOR RESTART AFTER DYSFUNCTION* In this state, all requested operations before a restart are carried out.
S6	*PROCEDURE TO RETURN TO INITIAL STATE* During this state we can manually control components of the MAS to set the MAS to a specific state.
S7	*GOING BACK TO A DETERMINED STATE* In this state, the MAS is manually or automatically set back to a position ready for resumption of autonomous running.

Table 5 Description of stop and restart procedures

The behaviors of the different types of agent are modified to take the NFR into account. As an instance, in the state $R1$, robot agents meet the functional requirements: their behaviors are not changed. In the $R2$ context, robot $r1$ goes to the docking station $d1$ and robot $r2$ goes to the docking station $d2$. In the $R4$ context, robots wait for commands from an operator. They perform tasks requested via their communication interfaces. In the $F1$ context, robot agents stay still and maintain their actuators in position. Robots have a battery dedicated to this task. Indeed, it is important to keep the manipulated containers in place (in case of a fall, contents can be deteriorated and operators may be injured).

An associated table lists all the devices that give safety-related information to the MAS and which allow it to detect transitions. Some of these devices are external to the MAS. As an instance, $b_{ES}(i)$ is a mushroom button with key release ($i = 0$:

State	Description
E1	*STOP PROCEDURE TO ENSURE SECURITY* This state concerns all the special sequences or actions which have to be taken in any emergency condition. This state includes stops, but also special movements to limit the consequences of the emergency or the failure conditions.
E2	*DIAGNOSIS OR FAILURE TREATMENT* In this state the MAS is examined after the failure and actions taken to allow a restart.
E3	*MAS STILL AUTONOMOUS RUNNING* Under certain circumstances, it is necessary to continue a partially autonomous behavior to reach goals. Some components of the MAS can be stopped to enable human interventions.

Table 6 Description of emergency procedures

emergency stop general button, $i \in [1, 8]$: workstations emergency stop buttons. $b_{ES}(i) = true$ when button i is locked). The $b_{net}(i)$ pulse button enables an operator to report that workstation i has been cleaned.

This graph highlights particular operating modes for the multi-agent system. As an instance, the loop $S1 \to R2 \to R1 \to S2 \to S1$ corresponds to a normal running cycle of the MAS. The loop $S1 \to R5 \to R4 \to S1$ corresponds to a regular test and fix sequence.

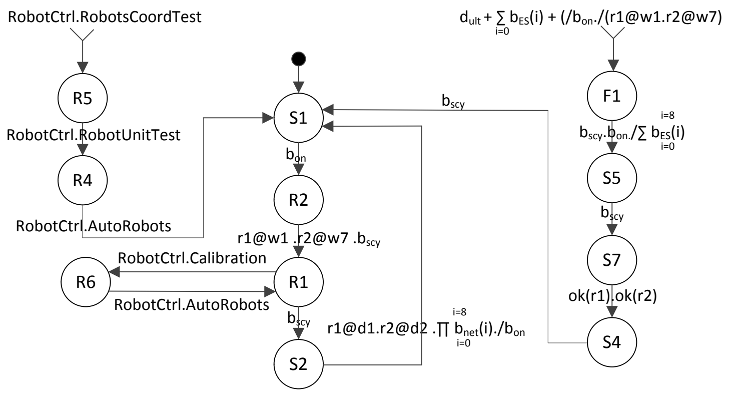

RobotCtrl.RobotsCoordTest
R5
RobotCtrl.RobotUnitTest
R4
RobotCtrl.AutoRobots
S1
b_{on}
R2
r1@w1 .r2@w7 .b_{scy}
RobotCtrl.Calibration
R6
R1
RobotCtrl.AutoRobots
b_{scy}
S2
r1@d1.r2@d2 .$\prod_{i=0}^{i=8} b_{net}(i)$./$b_{on}$
b_{scy}
$d_{ult} + \sum_{i=0} b_{ES}(i)$ + (/b_{on}./(r1@w1.r2@w7)
F1
b_{scy}.b_{on}./$\sum_{i=0}^{i=8} b_{ES}(i)$
S5
b_{scy}
S7
ok(r1).ok(r2)
S4

Figure 5 Synthesis of the NFR capture (from (Jamont et al., 2014a))

4.3 What specificities for such systems analysis ?

Problem. Embedded multi-agent systems present some important specificities that must be taken into account during analysis and specification.

Some global aspects (like safety) that we listed in the previous section have to be specified in global early requirements. However, a large number of challenges can be considered through models or modeling tools used in the analysis phase.

MAS related works. The safety (Challenge (5)) constraints settled as requirements must be translated into analysis models. Bresciani et al. (2004) in

Tropos take it into account under the form of interaction constraints modeled through actor use case specifications. Risks can be considered as the occurrence of unwanted negative consequences of an event and then be integrated in the behavior of agents. Raja et al. (2009) introduce the notion of conservative design which is the ability of an individual agent to evaluate its overall behavior from its actions and interactions even if they are not totally predictable. Asnar et al. (2011) handle safety as a framework for Tropos extending Tropos Goals to consider risks.

The integrity (Challenge (12)) of embedded multi-agent systems can advantageously take benefit from the notion of reputation and trust as shown in (Ganeriwal et al., 2008), that proposes reputation-based framework for sensor networks with high integrity.

To satisfy mobility (Challenge (11)) the main problem is to maintain the connectivity between agents. Some contributions in the field of embedded robot decision software try to solve this problem by subordinating the agent decision to the organization model (Le et al., 2009; Bouroche and Cahill, 2008).

The challenge (2) of timeliness is a hard one. Elmenreich (2003) argues that real-time capabilities are not surely guaranteed, due to the loose coupling of asynchronous agents in the MAS. It leads to temporal unpredictability and deadlock situations that drive to use MAS only for non-time constrained embedded applications. However, several works deal with real-time MAS adopting specific formal time models for MAS (Hutzler et al., 2005). More specifically, RT-Message (Julian and Botti, 2004) extension of the MESSAGE methodology introduces time in behaviors or interaction models, enriches goals and tasks ontologies, defines temporal information on the environment and design the MAS according to the constraints.

Recently there has been an increase in the number of works about power management (Challenge (4)). This challenge is addressed through behaviors or organizational structures. Ambuhl et al. (2004) proposes to use selfish agents and rewarding mechanisms to manage energy consumption in radio networks. Takimoto et al. (2007) presents a hierarchical organization of both classical and mobile agents to save energy consumption for multi-robots. Shakshuki and Malik (2007) minimizes energy consumption in wireless sensor networks using clusters.

Finally the heterogeneity (Challenge (8)) is mostly addressed through interaction (Elmenreich, 2003), different embedded systems having to deal with differing data representations and semantics.

Concerning formalisms, multi-agent methods 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 (Castor et al., 2004) (notation i* coming from the knowledge engineering, A-UML for interaction protocols and plan) or DESIRE (graph-based notation for knowledge modeling and specific hierarchical notation for tasks description).

Our proposal. A multi-agent analysis is the core of this stage of the DIAMOND methodology. This analysis is handled in a concurrent manner at two different levels (Figure 6): the society level in which the multi-agent system is considered as a whole and the individual level in which the agents of the MAS are built. This integrated iterative multi-agent design process involves four phases discussed below.

We share the view of Herlea et al. (1999) that several formalisms are necessary for the different levels of abstraction to cover all the phases of a life cycle. DIAMOND does not propose any particular formalism but an approach that aims at organizing the abstraction cycle using four stages. All kinds of more or less formal paradigms and languages can be employed from specific ones (Final State Machine, Hardware Definition Languages) to unified ones like UML. Even though they have not yet been used in DIAMOND, specific formal tools could be used like OMEGA UML, an UML profile supplying diagrams and tools to develop and verify real-time embedded systems. Another example could be SysML which represents a subset of UML 2 with extensions needed to satisfy the requirements of the UML for Systems Engineering. It proposes an interesting requirement diagram usable to visualize relationships between requirements.

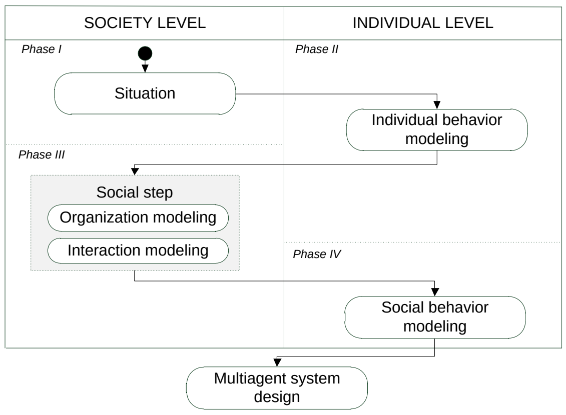


Figure 6 multi-agent analysis scheme

Phase I. From requirements analysis, the *Situation phase* aims to specify the modeled system boundaries and to characterize the agents, their roles and their contexts and then then environment. We first examine the environment boundaries, identify passive and active components. We then proceed to the problem agentification. Consideration is given to the characteristics of the environment (Russell and Norvig, 1995) to identify what is relevant to be taken into account in the resulting application. These features have an impact on the worldview of any agent and will be used by the designer to choose their future architectures.

At this point the designer can identify active and passive entities involved in the system. These entities can be in interaction or can be presented more simply as some constraints that 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.

Phase II. In order to create the agent individual behavior, the *Individual phase* focuses on the external and internal aspects of agents. The external aspect deals with the definition of the media linking the agent to the external world, what is perceived by an agent, by what means can an agent perceive something, what information can be obtained from other agents and how it can be used. We here use the context diagram borrowed from the SART notations (Ward and Mellor, 1989) to specify the context of each type of agent.

The internal aspect of an agent defines its own properties, i.e. what it can do (a list of actions) and what it knows (its representation of the agents, of the environment, of the interaction and of the organization elements). In most cases, the actions are carried out according to the available data about the representation of the environment by an agent. Such a representation based on expressed needs has to be defined 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 defined four types of actions. Primitive actions are tasks which are not physically decomposable. Nominal plans are temporal ordered lists of primitives. Situated actions need to own a partial world representation to execute their tasks.

Phase III. In order to design a collective behavior, the *Social phase* focuses on interactions among agents and organizations. The use of interaction protocols allows to achieve exchanges of data or tasks settled from the individual behavior needs. Although these interaction protocol descriptions are common to all the agents, they are rather external to them. Conflict resolution is efficiently handled by taking the relationships between the agents into account, that is, by building an explicit organizational structure. Such an organization is naturally modeled through subordination relations (Baeijs, 1998) or dependence relations (Sichman et al., 1994) that express the priority of one agent on another.

Phase IV. In order to achieve the socialization of individuals (the agents), in the *Integration phase*, social influences should be integrated into each agent individual behavior. It is therefore to analyze the possible leverage upon the two previous phases. These influences can be integrated within the agents either by modifying the nominal activity, or through communication and perception assessment capabilities. The decomposition hides the notion of agent control, i.e., how it handles its focus of attention, its decisions, and how it links them to its actions. Agent control features are addressed by models of agent. Based on required knowledge representation modes, required interaction modes, required decision techniques the choice of an agent architecture must be achieved at this step. The integration of social influences within the agents will lead to create an interesting dynamics within the MAS.

In Figure 7, we show the UWB agent context diagram we use in the analysis phase. This diagram enables to easily show all possible perceptions and possible actions of agent. Another advantage is that it allows to look at control flows between the physical part of an agent and its decisional part. In a word, context diagrams allow to specify the external shell of the agents. Table 7 illustrates the classification of the decisions we used in phases III and IV. The Observation column contains values observed for world representation (a given state of the world). The Perception column contains received messages and contents. The Decision parameters column presents parameters of decision functions. The Plans column refers to actions (or action sequences) able to be triggered after the validation of evaluation conditions. The Emergency column deals with orders of priority of the decision (immediately with preemption, after the current task, etc...). The Decision or cost function decides of the revelance of plans (actions) to trigger and of the emergency of the activation function of the state of the world.

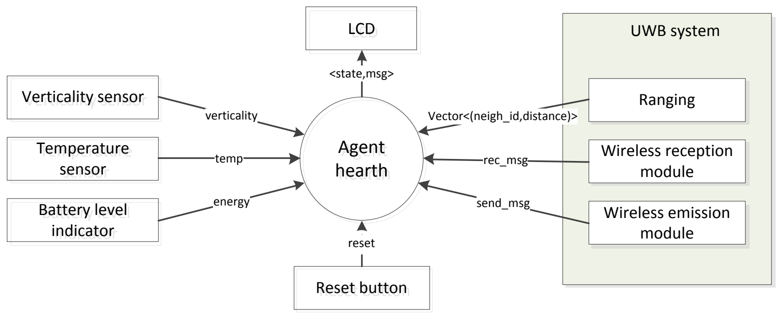


Figure 7 Context diagram of an agent

Observation	Perception	Parameters	Plans	Emergency	Cost
	Start (IHM)	Energetic Quota verified	Agent Initialization (data base, modules)	none	Energy
	Operational Agent	Energetic Quota verified	Network searching	none	Energy
	Detected network	Energetic and Traffic Quotas verified	Associating to the network	none	Energy, traffic
	No network	Energetic and Traffic Quotas verified	Starting a new network	none	Energy, traffic
	Associated to a network or New network started	Energetic Quota verified	Locating itself (initialization periodical interruption)	none	Energy, location
	Request of association from an agent	Energetic and Traffic Quotas verified	Associating and answering to the agent	none	Energy, traffic
Locate itself (periodical interruption)		Location frequency, Energetic and Traffic Quotas verified	Position request and/or *ranging (+confidence)*	none	Energy, traffic
	position answer and/or *ranging (+confidence)*	Energetic Quota verified	Estimating its position and Updating the knowledge base	none	Energy
	Position request	Energetic and Traffic Quotas verified	Returning its position and the associated confidence index	answering immediately (application)	Energy, traffic
Standard standby (periodical interruption)			standard standby	none	none
Inactivity		Inactivity	Extended standby	none	none
	Software update request	Energetic and Traffic Quotas verified	Update procedure launching	none	Energy, traffic
	Faulty organization detection	Energetic and Traffic Quotas verified	Election, negotiation or nomination procedure launching	Urgent	Energy, traffic
Traffic, energy, position confidence, mobile		Traffic, energy, position confidence, mobile	Changing frequency of location and/or tasks priority	none	Energy, traffic, location

Table 7 Agent's control rules

4.4 How to design the multi-agent system?

Problem. Embedded systems involve monolithic and platform-dependent software (Möller et al., 2004). They are difficult to maintain, to upgrade and to specialize to another application. Exporting software from one specific platform to another is a heavy work. We can define a component as an elementary object that performs a specific function which allows developers to define reusable segments of code. It is designed in such a way as to easily operate with other components to create an application. 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 stage offers an efficient process leading to an abstract component decomposition by starting from the informal description of the MAS built during the previous stage.

An important amount of work has been done to adapt component models to embedded systems. Some models focusing on software appear in the literature like PBO (Stewart et al., 1997), Koala (van Ommering et al., 2000), ReFlex (Bard, 2003), PECOS (Jawawi et al., 2006), Rubus (Hanninen et al., 2008). Few works try to unify hardware and software components (Bunse and Groß, 2006), the most detailed ones are co-design works like (Cesário et al., 2004).

MAS related works. In the multi-agent domain one issue is to bridge the gap between analysis and design (Dinkloh and Nimis, 2003). An important interrogation raised about the way multi-agent principles can be incorporated in combined Hardware/Software based reconfigurable Systems (Challenge (10)). Some approaches present a vertical partitioning to constitute hybrid societies of both pure hardware agents and pure software agents (Naji et al., 2004). Other studies recommend decomposing agents horizontally including both physical and software parts. Meng (2005) argues that the agent model (BDI) can be seen as a unified structure allowing to partition the systems into software agents and hardware agents based on heuristic approaches and providing more flexibility, intelligence, autonomy, and scalability than existing module-based approaches.

Components can be used to build the agents. Few multi-agent methods introduce an actual component 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 re-usability, 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.

Some works aim at building the agents using traditional control theory components like Proportional/Integral/Derivative controller (like in (van Breemen and Vries, 2000)) or Labview components (Polaków and Metzger, 2009) to design distributed multi-controller systems.

Our proposal. We propose to base the design phase on an abstract component decomposition. An abstract component is an elementary object that performs a specific but reusable function. It is designed in such a way as to easily operate with other components to create an application. Components can be combined to 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 multi-agent system built during the previous stage.

The Problem Description phase enables the analyst to identify and to delimit the application domain of the problem, and specific domain aspects that should be taken into account. At this point the architectures for agents must be adopted. The architectures following hybrid architectures combining reactive and cognitive capabilities are well suited in our context. Operating in a normal mode, they will be reactive using a stimuli/response paradigm to be the most efficient. The agents will activate deliberative capabilities in case of a configuration alteration.

In the *Agent applicative tasks design* phase, we must build the external shell of the agent i.e. elaborating the interface with the external world for each sensor and effector. It is time to choose technological solutions for them and to complete the context diagram to specify all information about the signal (as illustrated in Table 8 for the previous SART context diagram). The next step is to design the internal shell of the agent. We start with the elaborated actions according to the task tree. At this stage it is necessary 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.

Information	Description
Reset	Active on high logic level (1b)
Verticality	Vertical angle (float IEEE 754, 32b)
Temperature	External temperature in celsius (float IEEE 754, 32b)
Energy	Battery level expressed as a percentage of the battery initial capacity (float IEEE 754, 32b)
<state,msg>	Visualization of the node state (8b) and specific alert messages (null terminated string)
Vector<neigh_id, distance>	A vector given for distance to each neighboor i.e. agent in the range of the UWB module. (neigh_id: uint 16b, distance: in meters, float IEEE 754 32b)
Rec_msg	Bit sequence $(sender_{uint16b}, receiver_{uint16b}, data\_sizesender_{ushort8b}, data_{0-2048b})$
Send_msg	Bit sequence $(sender_{uint16b}, receiver_{uint16b}, data\_sizesender_{ushort8b}, data_{0-2048b})$

Table 8 SART context diagram interface specification

Here we are using the potential of the spiral life cycle. The enrichment of components is made through the derivation of the results of the MAS building iterations.

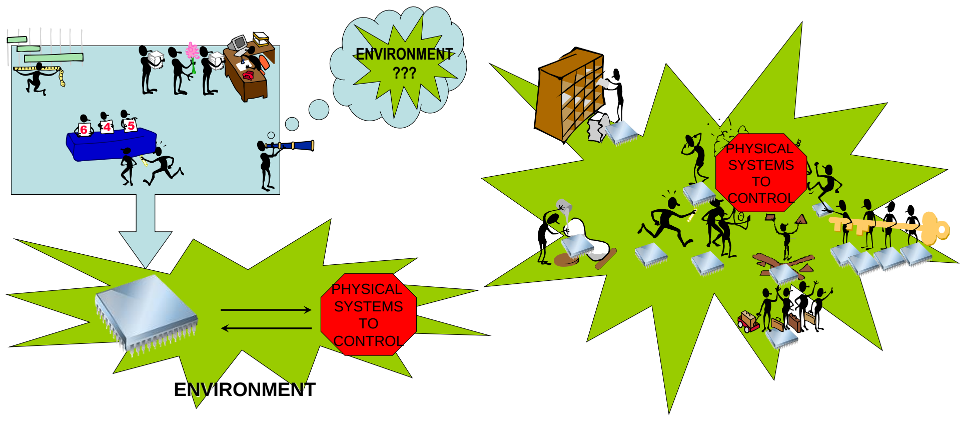
In the generic design phase, DIAMOND uses components as operational units, as seen previously. In these components, we use a final state machine or a components set to describe the internal running. These formalisms enable to generate software code or hardware specifications in VHDL (Very High Speed Integrated Circuit Hardware Description Language).

4.5 How to implement this system?

Problem. Once designed, we can find different kinds of possible implementation for an embedded multi-agent solution that can be implemented following centralized or pseudo-parallel or truly distributed ways.

Non distributed implementation: Non-really distributed solutions are appreciated by industry because they can overrule the psychological barrier of decentralization, i.e. the delegation of authority to a set of autonomous systems. Indeed, in this case, the system is clearly located in one place and the maintenance activity seems to be easier. The system can be controlled by a multi-agent system that replaces the traditional command law coming from automation. In this sense it is not a distributed embedded system. In this type of target, the agents of the multi-agent system are on a single platform connected to all the sensors and all the effectors. Each agent has its own role and its own representation of the real world. Agents get involved on this platform and proceed to a collective resolution of the problem. If the multi-agent is deployed on a Multiprocessor System-on-Chip (MPSoC), we can introduce the concept of multi-agent System on Chip (Figure 8a, MASoC).The multi-agent system can also be implemented using a real-time executive system achieving a pseudo-parallelism but conserving a centralized control (Julian and Botti, 2004). An example of area using this type of architecture is the supply chain management.

Distributed architecture: In case of distribution, agents are deployed on the different units composing the architecture. Agents dynamically interact to solve the problem according to their partial representation of the environment. Each agent can be deployed as a whole or as a set of components using Services Oriented Architectures (Spanoudakis and Moraitis, 2007) or component based environment as OSGi (Wu et al., 2007; Jaszczyk and Król, 2010).
If one agent is deployed on one platform, the concept of Agent on Chip can be introduced (Figure 8b) (Meng, 2005). An example of area involving this type of architecture is the ad-hoc network based multi-agent systems.



(a) Non distributed implementation - MASoC (b) Distributed implementation - AoC

Figure 8 Decentralized intelligence implemented on a distributed and a non distributed architecture

Whatever solution is chosen, agents are serious candidates that offer good integration properties (Challenge (9)) because of the loose coupling between agents and the non-necessity to give a complete explicit model of the whole system.

Bringing the agent implementation closer to the hardware is a way to ensure improved reactivity (Challenge (1)) but leads to the problem of giving autonomous behavior to a physical object.

There are two ways to embed autonomous behaviors into a physical object depending on the computation capabilities available on the object.

The first one consists in embedding the behavior into the physical agent. The physical object is seen as monitored by the agent. It is the case for the sensors of a WSN. Each sensor embeds the agent behavior (in the code).

The second way lies in deploying the behavior on the Internet or on a local server. This server hosts an avatar of the physical object. For example, if you want to give an intelligent behavior to an RFID marked object, like a yogurt pot, you cannot embed the code into the physical object.

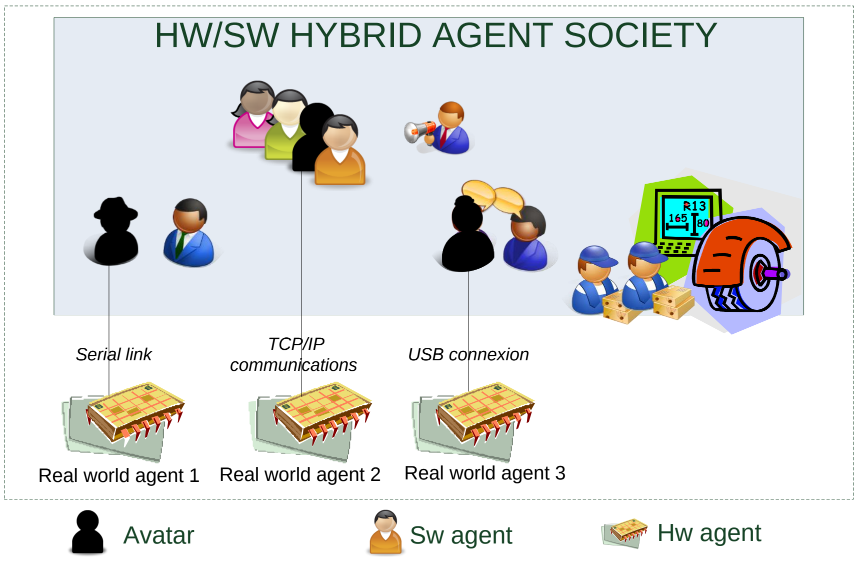


Figure 9 Real world agent interacting with software agent through avatars (from Jamont and Occello (2011))

MAS related works. The more the agents are implemented as pure physical agents, the more difficult it is to ensure the capacities of reconfiguration (Challenge (10)) because the problem becomes a full hardware state commutation (Meng, 2005).

The implementation step is also one of the place to take Time Constraint Processing into account (Challenge (2)). At this level the choice can be made to implant hardware agents respecting hard time constraints cohabiting with less efficient software agents (Naji et al., 2004) or to deal with time constraints communications (Meng, 2005).

Our proposal. The implementation level we propose in DIAMOND begins with the *Partitioning phase*. Its objective is to achieve the software/hardware partition of the components defined during the *generic design* (Figure 3). Through our previous works in this field and existing co-design works like (Adams and Thomas, 1996), we have identified some major criteria which can be considered as relevant for the agents: cost, performance, flexibility, physical constraints and algorithm complexity.

We then propose a *co-simulation phase* and a *co-validation phase* to simulate the collaboration between software parts, hardware parts and their interfaces.

Finally, each component is completely specified according to common graphic specification formalism. For each of them the designer has already manually specified partitioning into hardware and software implementation. The graphic specifications are converted into source code (a portable language like Java or C++ is then used) for the components for which an implementation software has been selected. The specifications in Hardware Description Language (HDL) are generated for hardware ones. HDL enables to produce formal descriptions of digital circuit and their interconnections. In our method dedicated tool, the compilation of the source code and the hardware synthesis of different specifications in VHDL (Pedroni, 2004) are carried out as illustrated on Figure 10.

To illustrate the hardware implementation of components, we consider the graph given in Figure 11 that models the MWAC agent role attribution (a simplification of algorithm described in Jamont et al. (2010)) according to the number of representative neighbor agents (NRN).

The VHDL description of this agent decision component is divided into two parts:

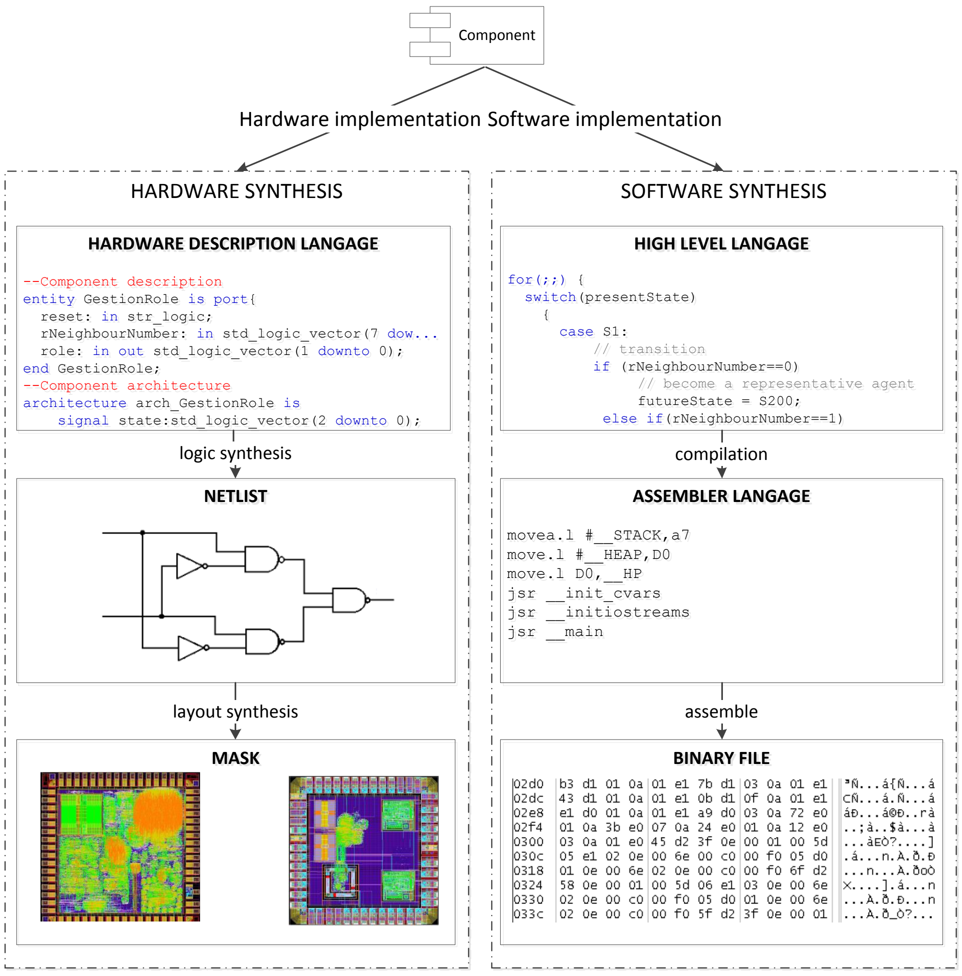
1. The component is described according to an external perspective.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY MWAC_role_attribution IS
     PORT (
          -- Description of the inputs and outputs of the structure by
          -- explaining, for each, the name,  direction, and type.
          clk: in std_logic;
          reset: in std_logic;
          NRN: in std_logic_vector (7 downto 0);
          role: inout std_logic_vector (2 downto 0);
           );
END ENTITY MWAC_role_attribution ;
```

2. We specify the internal architecture of the component.

```
ARCHITECTURE core OF MWAC_role_attribution IS
  constant NOTHING: std_logic_vector(2 downto 0):= "00";
  constant SIMPLE_MEMBER: std_logic_vector(2 downto 0):= "01";
  constant LINK: std_logic_vector(2 downto 0):= "10";
  constant REPRESENTATIVE: std_logic_vector(2 downto 0):= "11";
   TYPE STATE_TYPE IS ( S1, S2, S3, S4, S5 );
   SIGNAL current_state: STATE_TYPE;
  SIGNAL next_state: STATE_TYPE;

BEGIN
  -- Logical structure description.
   computeRole: PROCESS (clk, reset)
       BEGIN
           IF reset = '1' THEN
               role <= NOTHING;
           ELSE
               IF clk'event AND clk = '1' THEN
                   CASE current_state IS
```



```
--Component description
entity GestionRole is port{
  reset: in str_logic;
  rNeighbourNumber: in std_logic_vector(7 dow...
  role: in out std_logic_vector(1 downto 0);
end GestionRole;
--Component architecture
architecture arch_GestionRole is
    signal state:std_logic_vector(2 downto 0);
```

```
for(;;) {
  switch(presentState)
    {
      case S1:
          // transition
          if (rNeighbourNumber==0)
               // become a representative agent
               futureState = S200;
           else if(rNeighbourNumber==1)
```

```
movea.l #__STACK,a7
move.l #__HEAP,D0
move.l D0,__HP
jsr __init_cvars
jsr __initiostreams
jsr __main
```

```
02d0  b3 d1 01 0a  01 e1 7b d1  03 0a 01 e1  ªÑ...á{Ñ...á
02dc  43 d1 01 0a  01 e1 0b d1  0f 0a 01 e1  CÑ...á.Ñ...á
02e8  e1 d0 01 0a  01 e1 a9 d0  03 0a 72 e0  áÐ...á©Ð..rà
02f4  01 0a 3b e0  07 0a 24 e0  01 0a 12 e0  ..;à..$à...à
0300  03 0a 01 e0  45 d2 3f 0e  00 01 00 5d  ...àEÒ?....]
030c  05 e1 02 0e  00 6e 00 c0  00 f0 05 d0  .á...n.À.ð.Ð
0318  01 0e 00 6e  02 0e 00 c0  00 f0 6f d2  ...n...À.ðoÒ
0324  58 0e 00 01  00 5d 06 e1  03 0e 00 6e  X....].á...n
0330  02 0e 00 c0  00 f0 05 d0  01 0e 00 6e  ...À.ð.Ð...n
033c  02 0e 00 c0  00 f0 5f d2  3f 0e 00 01  ...À.ð_Ò?...
```

Figure 10 Hardware/software synthesis

```
                    WHEN S1 =>
                     role <= NOTHING;
                     IF NRN="00000000" THEN
                        next_state <= S4;
                      ELSIF NRN="00000001" THEN
                       next_state <= S2;
                      ELSE
                        next_state <= S3;

           ...

                   END CASE;
                   current_case <= next_case;

               END IF;
           END IF;
     END computeRole;
END core ;
```

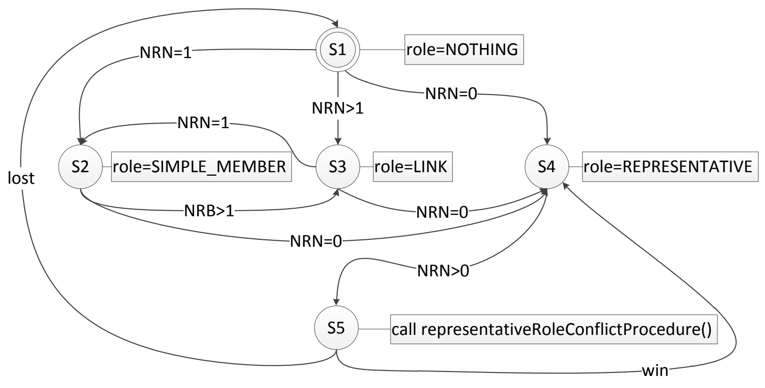


Figure 11 Role attribution in MWAC

4.6 How to simulate such an embedded multi-agent system?

Problem. In the context of real world systems, multi-agent simulation tools must operate the applicative system according to realistic physical models (environment, energy consumption models, wave propagation models and so on). Tools must facilitate the use of these models and a lot of others used by the agents (interaction models, organization models, user models etc.).

When the simulated solution meets the requirements, it is necessary to embed the solution into the real world devices. A specific effort must be achieved to tune algorithms to fit the resources of the devices. They must be degraded to accommodate, for example, memory limitations, reduced computation capacities (because of sleep mode). From these modifications may result deviations of the global behavior. The tools must help to control effects of local changes on the behavior of entities on the global behavior.

Hw/sw hybrid simulation allows to verify that there is no deviation between the behavior of the real embedded implementation of the MAS and the designed software part behavior of the MAS.

MAS related works. Multi-agent simulations can represent systems (and not only software) that exhibit complex, dynamic relationships between their components. In the military domain they are used to simulate systems of systems (Ilachinski, 2003).

Due to this complexity and the physical constraints the approach must allow for errors and incompleteness in the models (Dewar et al., 1996) that can be built with multi-agent paradigms at varying levels of fidelity and completeness.

The simulation phase also allows a risk-based study of whole-system model to evaluate safety policies (Alexander et al., 2009). The whole approach must derive a physical system from qualitative specifications. Many multi-agent models provide visualization of ongoing simulation runs, allowing users to watch what is happening, making the simulation valuable for the comprehension of system

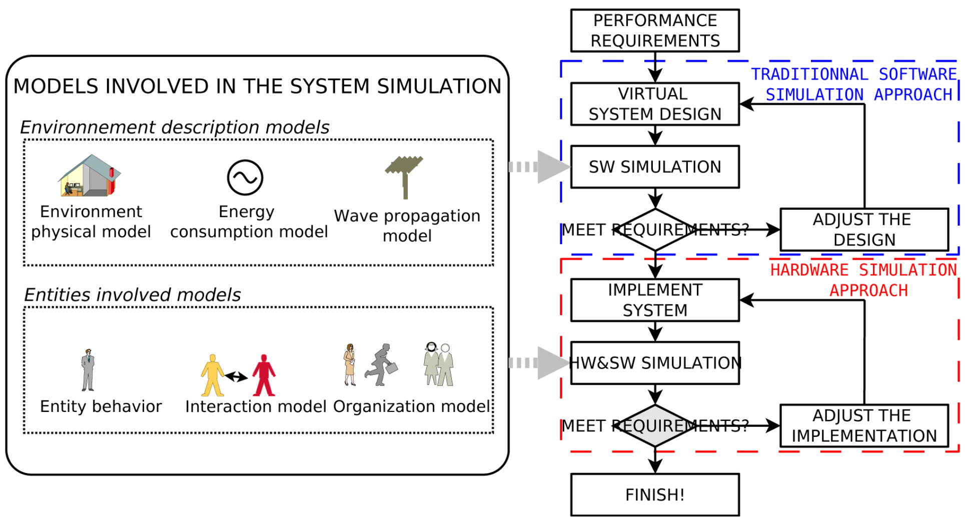


Figure 12 Using MASH to design and to deploy real world MAS (from Jamont et al. (2013))

behaviors. Embedded multi-agent simulation can also enable to evaluate and correct deviation between logical and physical behaviors.

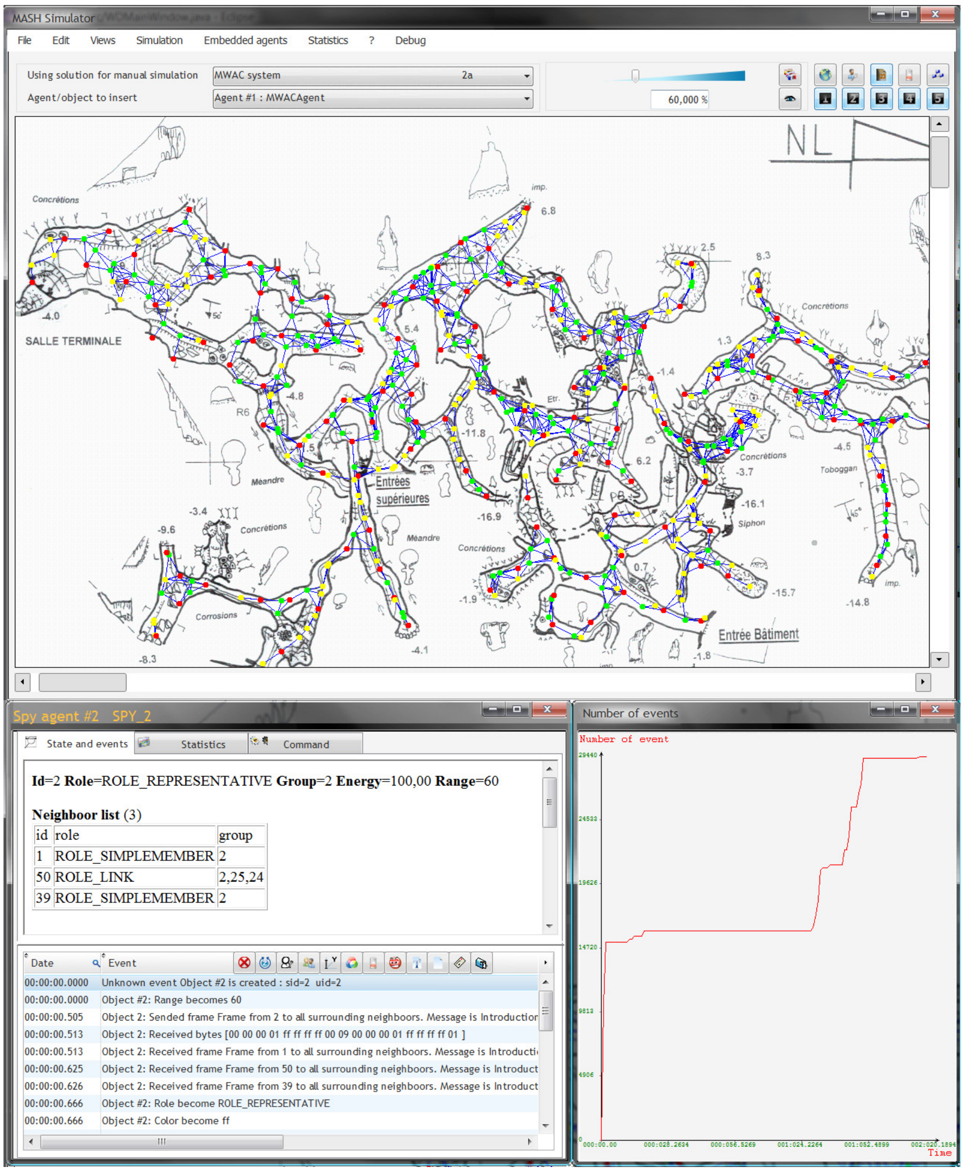
To the best of our knowledge, there is no multi-agent tool which enables the simulation of MAS simultaneously involving software agents and hardware agents embedded in the real world. In the context of automatic guided vehicles based transportation systems, Weyns et al. (2005) developed a tool which builds a virtual discrete environment from physical observations to plan actions. Hassaine et al. (2009) developed a simulator which acquires data from sensors to obtain more realistic virtual simulations of military operations.

Concerning the simulation of real world physical environment, the most popular and efficient tools are Matlab (Moore, 2011) and Labview (Fairweather and Brumfield, 2011). There are some works which attempt to bring together multi-agent systems and these tools. For instance, Ponci et al. (2005) and Conte et al. (2009) propose a Labview implementation of multi-agent systems in which agents are considered as virtual instruments. By this way authors lose the advantages of multi-agent simulators, such as the possibility of large scale simulation, or the use of multi-agent specific models. More recently, a matlab/simulink multi-agent toolkit dedicated to distributed networked fault tolerant control systems has been proposed (Mendes et al., 2010) and MACSimJX (an extension of Jade) has been implemented to enable interaction with Simulink (Robinson et al., 2010).

Our proposal. We have designed a tool called MASH (multi-agent Software/Hardware simulator) presented in (Jamont et al., 2013) and demonstrated in (Jamont et al., 2011; Jamont and Occello, 2013).

It enables both the simulation and the execution of embedded MAS:

- including real world software/hardware agents. Behavior of these agents are computed by the real world platform and injected in MASH;



Id=2 **Role**=ROLE_REPRESENTATIVE **Group**=2 **Energy**=100,00 **Range**=60

Neighboor list (3)

id	role	group
1	ROLE_SIMPLEMEMBER	2
50	ROLE_LINK	2,25,24
39	ROLE_SIMPLEMEMBER	2

```
Date            Event
00:00:00.0000   Unknown event Object #2 is created : sid=2  uid=2
00:00:00.0000   Object #2: Range becomes 60
00:00:00.505    Object 2: Sended frame Frame from 2 to all surrounding neighboors. Message is Introductior
00:00:00.513    Object 2: Received bytes [00 00 00 01 ff ff ff ff 00 09 00 00 00 01 ff ff ff ff 01 ]
00:00:00.513    Object 2: Received frame Frame from 1 to all surrounding neighboors. Message is Introducti
00:00:00.625    Object 2: Received frame Frame from 50 to all surrounding neighboors. Message is Introduct
00:00:00.626    Object 2: Received frame Frame from 39 to all surrounding neighboors. Message is Introduct
00:00:00.666    Object #2: Role become ROLE_REPRESENTATIVE
00:00:00.666    Object #2: Color become ff
```

Figure 13 The MASH main windows

- using realistic physical models at multi-agent level (environment models, wave propagation models etc.) or at agent level (energy consumption model etc.).

MASH assists the designer during the design and the deployment of the embedded multi-agent system (Figure 12). Indeed when the software simulation meets the requirements (according to the different models of the physical component of the multi-agent system), one has to embed the entities behavior on the real devices.

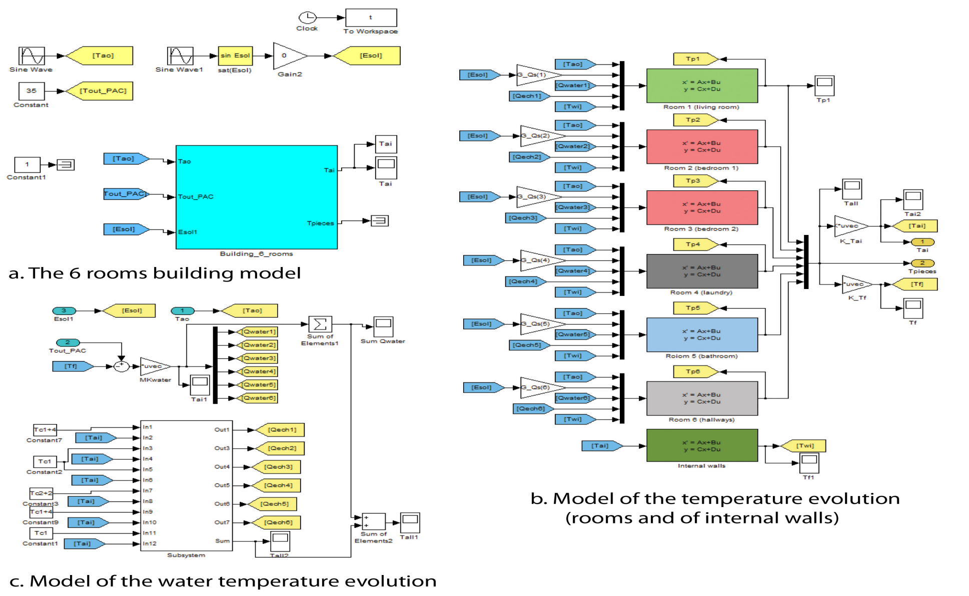


Figure 14 Environment physical model: Part of the Matlab block model (from Jamont and Occello (2011))

A hardware/software simulation is available to tune and to test the real world embedded devices. We may observe the effect of deviations of the behavior of virtual agents and the behavior of real world agents that are resources constrained. These deviations are observed by measuring the difference between the real world behavior of an embedded agent and the simulated one.

In Figure 13, we can see the main windows which allow to view the system according to customizable views (defined by the designer). It is possible to spy on an agent i.e. to inspect their internal states and the historical events. The inspected agent is Agent #2 which is a virtual agent. Among the various events, we can see the bytes received by this agent and their translation into logical messages. As an instance, the received bytes (`00 00 00 01 FF FF FF FF 00 09 00 00 00 01 FF FF FF FF 01`) correspond to a HELLO message broadcasted by real world agent #1 and received by agent #2. In MASH, a bit specification is associated to each type of message.

An implementation of this model has been proposed into MatLab/Simulink as a block diagram. An illustration of such diagrams is given in Figure 14. The figure shows the entire model of the six-room building that we simulate (Figure 14a) and how we compute the temperature of the rooms and of the internal walls composing the building according to equations described in Jamont and Occello (2011) under the form of a state space (Figure 14b).

MASH enables to easily use MatLab/Simulink models. The designer of an eMAS should only execute an m-script (a matlab script). This script enables the sharing of XML formatted data. It must be customized to specify the inputs (that an agent can act) and outputs (that an agent can perceive) of the matlab model.

5 Evaluation

In this last section, we first want to make a synthesis showing what aspects of multi-agent systems are a strength in this context, and what are points where steps forward have to be made to meet the requirements. We then make a synthesis about how these challenges are taken up with the DIAMOND methodology that we elaborate and that is dedicated to embedded multi-agent systems development. We conclude this evaluation with a synthesis of efforts of existing methodology on the issues covered all along the paper.

5.1 Synthesis about MAS relevance for NES challenges

Some of the challenges are, without any particular effort, taken into account by natural aspect MAS at the level of the paradigm through the intrinsic nature of MAS, of the methodologies through the specific approach proposed by MAS development, and of the platforms integrating specific technical features. They are synthesized in Table 9. However the multi-agent community has to take particular care of some specific multi-agent components and has to enhance multi-agent models, methodologies and platforms to meet some particular features of networked embedded systems. They are synthesized in Table 10.

Some of the required properties are reached through multi-agent paradigm specificities. They are naturally related to multi-agent models specificities through:

- Individual capabilities: Reactivity (1) and Timeliness (2) are related to decision, Heterogeneity aggregation (8) to knowledge and behavior management capabilities. Power Autonomy (4) refers to agent autonomy notions but is rarely considered as a decision parameter at a model level. Guaranteeing that a multi-agent system always reaches a satisfactory state is a difficult problem even more with systems strongly subject to their physical environment. Works about Liveness(3) are currently ongoing. They are crucial in the embedded context.

- Social capabilities: Integration (9) is related to cooperation, Self-organization (10), Mobility (11) and Integrity (12) to interaction and organization

	MAS Methodology	MAS Paradigm	MAS Platform
Reactivity (1)		⋆	⋆
Timeliness (2)		⋆	⋆
Liveness (3)		⋆	⋆
Power Autonomy (4)		⋆	⋆
Safety Management (5)	⋆		
Complexity (6)	⋆		
Concurrency (7)	⋆		⋆
Heterogeneity Aggr.(8)		⋆	
Integration (9)		⋆	
Self-organization (reconfiguration)	⋆	⋆	
Mobility (11)	⋆	⋆	⋆
Integrity (12)		⋆	⋆

Table 9 Challenges taken up by natural features of MAS

	MAS Methodology	MAS Paradigm	MAS Platform
Reactivity (1)			⋆ links to physical world
Timeliness (2)	⋆ time requirement modeling		real time OS
Liveness (3)		⋆ physical constraints	physical constraints
Power Aut. (4)		⋆ energy as decision parameter	energy as physical parameter
Safety man. (5)	⋆ safety requirement modeling		
Complexity (6)		⋆ multi-scale decision	
Concurrency (7)			⋆ really decentralized architectures
Heter. Aggr.(8)			⋆ interoperability
Integration (9)			
Self-org. (10)			
Mobility (11)	⋆ mobility management policies		⋆ mobile networks compliance
Integrity (12)	⋆ trust management policies	⋆ trust and reliability mechanisms	

Table 10 Challenges deserving an enhancement of multi-agent aspects

notions. Integrity can benefit from trust and reliability mechanisms recently introduced to secure interactions. The integration of multi-scale mechanisms (like holonic properties) can improve Complexity mastering (6), much needed in the embedded context. Heterogeneity aggregation (8) can be dealt with at a social level if it is considered as an interoperability feature.

Some are naturally taken into account at a methodological level, concerning the design of:

- Structures: Complexity (6) is related to the decentralized nature of the analysis, Concurrency (7) to sophisticated agent architectures,
- Behaviors: Safety management (5), Concurrency (7), Self-organization (10) and Mobility (11) are addressed by the analysis. A special attention should be paid to Safety management (5) specifications in relation to the physical context importance. Some efforts have to be made on the time aspects modeling to meet the Timeliness (2) challenge, including for example real-time reasoning in agent models. Mobility (Mobility (11)) and trust (Integrity (12)) management policies have to be introduced in the analysis process.

Finally some of the challenges are addressed at a platform level through the features of:

- Nodes: Reactivity (1) or Concurrency (7) depends on infrastructures features, nodes especially have to offer ability to access the physical world and a real physical decentralization. Timeliness (2) or Power Autonomy (4) depends on execution infrastructure performances, their importance increases with the criticality of the application.
- Communications: Mobility (11) and Integrity (12) rely on communication infrastructure features.

5.2 Assessment of DIAMOND for NES challenges

In the following we give an insight into benefits and weaknesses of DIAMOND and its associated tools from the analysis of real world applications that validate it.

5.2.1 Benefits

The principal strength of DIAMOND and MASH is that they can produce eMAS contributing to enhance MAS response to NES for ten challenges out of twelve.

The approach we propose brings agents closer to the physical world. The VAICTEUR AIR2 project (G) showed the use of realistic environments for agents (in that case the thermal environment). A real co-evolution of virtual and real physical agents offers to project virtual adaptation of agents to the actual embedded entities as a tuning tool as illustrated by the Kurasu project (C).

Concerning the dynamics, we showed in the project (A) the impact of the real time aspects on the design of the agents and shown that they can be taken into account for each ability of the agents and within each level of the design. From a development point of view (A) adopted a cross-layering architecture to merge location service access and data-processing through several protocol layers. Our approach enables to employ techniques using a real time OS that makes embedded software more independent from its hardware support.

eMAS enforce the adaptation to physical constraints of collective decision embedded software. For example, energy level directly measured by physical sensors is integrated as a decision parameter in the ENVironment SYStem project (B) or the PULSER Project (A). In DIAMOND, physical parameters can be considered as criteria for the system sizing. The PULSER Project (A) took benefit of this ability with energy.
The special attention to characteristic behaviors of the user specified in the SIET project agents (E) illustrated how the safety requirements modeling can be operationalized using DIAMOND. In the same way, the PALETTE (F) multi-robot system benefited from DIAMOND for the specification of a collective management of the safety of the operator.

Almost all the discussed systems involve really decentralized architectures with agentified decision-making kernels. For security and fault tolerance reasons, this kind of systems imposes the decentralization of the decision-making process among the entities constituting the network.
The introduction of virtual agents linked to pure physical objects, embedded software/hardware architecture or even other external systems are a real advantage to treat interoperability. Virtual agents are key elements of the ASAWOO ANR project (H) were they are called "avatar" of connected objects on the Web of things.

eMAS can efficiently model mobility management policies treated finally as an adaptation problem as in the PULSER Project (A). Mobility can also be considered as compliance with mobile networks as emphasized in the ENVironment SYStem project (B).

The trust management and reliability policies implemented in agents of the SIET project (E) for ensuring their data integrity showed that advanced NES that manage high level knowledge take benefit of availability in DIAMOND modeling.

As a synthesis we find on Table 11 the challenges we addressed in the different realizations introduced section 4, we made using DIAMOND and MASH.

	DIAMOND	MASH
Reactivity (1)		⋆ links to physical world (G) (C)
Timeliness (2)		⋆ real time OS (A)
Liveness (3)		⋆ physical constraints (B) (G)
Power Aut. (4)	⋆ energy as decision parameter (A) (B)	⋆ energy as physical parameter (A)
Safety man. (5)	⋆ safety requirements modeling (F) (E)	
Concurrency (7)		⋆ really decentralized architectures (A) (B) (C) (E) (G) (H)
Heter. Aggr.(8)		⋆ interoperability (H)
Self-org. (10)		
Mobility (11)	⋆ mobility management policies (B)	⋆ mobile networks compliance (A)
Integrity (12)	⋆ trust management and reliability policies (E)	

Table 11 Challenges addressed in applications using DIAMOND/MASH

Finally, we can emphasize some general strengths of DIAMOND. DIAMOND completes the multi-agent life cycle by adding essential phases coupling software and hardware practices that are architecture dedicated simulation and prototyping with MASH. DIAMOND allows specifying different kinds of ability at different levels from behavior to physical aspects. To this end DIAMOND is based on different adapted existing formalism deriving to multiple architectures and targets. This technique makes embedded agents more independent from their hardware support. Above all we can claim that DIAMOND is real world improved thanks to all the real world applications described all along the paper.

5.2.2 Weaknesses

We must admit that none of the use cases deal with very complex behaviors, knowledge or architectures and that none of them address the composition or the integration of very heterogeneous systems. DIAMOND would deserve to be completed by real world applications emphasizing the two challenges we did not really cover in addressed use cases: (6) Complexity and (9) Interoperability.

PULSER Project (A) showed that the approach presents some limitations if we want to address pure physical optimization. Some problems are encountered specifying cross-layering architecture, fault tolerance or limitation of the traffic.

The KURASU project (C) addressed the lack of a quality assessment phase in DIAMOND. This was especially true in the definition of the evaluation metrics based upon the multi-agent analysis result. We are working on the identification of the impact of uncertainties of measures on the agent control. Those uncertainties should be taken into account in the MASH simulation models.

The SIET project (E) confronted us to the lack in DIAMOND of a Human Machine Interface. DIAMOND focuses on the analysis of interfaces between agents and the real world environment forgetting interface with humans. An HMI analysis must be integrated to solve difficult problems of interaction between people and agents.

Overall, the general weakness of DIAMOND is the bad coverage of software engineering and code management tools. Currently, the software instrumentation of the methodology is quite poor. Neither DIAMOND nor MASH propose a documentation generator or a real Integrated Development Environment. Testing a solution remains a manual task. A scenario editor is available in MASH, but the platform does not supply a complete test tool with automatic scenario generation. In the same way, no tool for code generation is proposed in MASH. The code deployment remains one of the weak points. MASH allows tuning an agent code and assists the code portability and adaptation. But the platform does not supply any tool to automatically deploy a high number of nodes. This remains an important drawback limiting the dissemination of the tool.

Efforts for the enhancement of existing multi-agent methodologies for embedded challenges

All over the paper in each section, we examined related works. Table 12 shows a comparative synthesis of efforts for the enhancement of existing multi-agent methodologies for NES. Each column corresponds to a section of the paper and the discussion can be found there.

	adapted life cycle	requirements	specifications	co-design	co-simulation	deployment	Specific contribution
ADELFE		(+)					Werneck et al. (2007)
GAIA and extensions		(+)					Blanes et al. (2009)
MASE and extensions		+	+	+			Badr et al. (2008); Harmon et al. (2009)
MASSIVE and extensions						(+)	Lind (2001)
MESSAGE and extensions			++ timeliness				Julian and Botti (2004)
PASSI and extensions	(+)				(+)		Cossentino et al. (2008); Chella et al. (2004)
Prometheus	(+)					(+) artefacts	Padgham et al. (2005)
SODA		(+) security					Omicini (2001)
TROPOS and extensions		+ safety					Asnar et al. (2011)
DIAMOND	++	++	++	++	++	+	

Table 12 Enhancements of MAS methodologies for NES problematic

We can notice among these works:

- non specific enhancements that can be useful for embedded multi-agent systems, noted (+).
- specific enhancements of the methodology for embedded multi-agent systems, noted ++.
- external contributions to the methodology not integrated into it, noted +.

But none of them have been built for this purpose as the DIAMOND methodology. Yet next generation AOSE methodologies (Dam and Winikoff, 2013) will have to integrate networked embedded systems challenges.

6 Conclusion

In this paper we argued that networked embedded systems, as a support for decentralized embedded systems, can be viewed as artificial complex systems. We argued that the design of networked embedded systems and software can benefit from multi-agent systems and approaches.

The paper introduced twelve challenges that have to be met to efficiently cover up the features of the field. Even if these challenges could have been used as evaluation criteria for models, methodologies and platforms, our aim was not to make methodologies or models comparisons in order to argue in favor of one methodology or class of models for agentified networked embedded systems.

However DIAMOND and MASH successfully introduced enhancements to meet all of the challenges.

DIAMOND proposes contributions in terms of hybrid software/hardware multi-agent life cycle. It especially integrates all the phases of development from analysis to implementation. It introduces a multi-paradigm spiral life cycle. It proposes components, used as tools for integration, allowing software or hardware derivation.

MASH completes DIAMOND for the simulation and the help to the development. It contributes to test physically decentralized behaviors by simulating realistic physical environment, energy consumption and mobility.

Classical general methodologies can partially be used in this context due to the intrinsic properties of multi-agent paradigm. Finding a unified approach dedicated to the design of embedded multi-agent systems is a real and promising research area in the field of agent oriented software engineering.

Acknowledgement

The authors wish to thank Ms. Cécile Metge who assisted in the proof-reading of the article.

References

Adams, G. and Paques, J.-J. (1988). Gemma, the complementary tool of the grafcet. In *Proceedings of the Fourth Annual Canadian Conference on Programmable Control and Automation Technology Conference and Exhibition*, Toronto, Canada. IEEE.

Adams, J. and Thomas, D. (1996). The design of mixed hardware/software systems. In *33rd Design Automation Conference*, Las Vegas, USA. ACM.

Alexander, R., Alexander-Bown, R., and Kelly, T. (2008). Engineering safety-critical complex systems. In *Proceedings of the 1st CoSMoS Workshop*.

Alexander, R., Hall-May, M., Despotou, G., and Kelly, T. (2009). Towards using simulation to evaluate safety policy for systems of systems. In *Safety and Security in Multiagent Systems*, volume 4324 of *LNCS*, pages 49–66. Springer.

Ambuhl, C., Clementi, A. E., Penna, P., Rossi, G., and Silvestri, R. (2004). Energy consumption in radio networks: Selfish agents and rewarding mechanisms. In *Approximation and Online Algorithms*, volume 2909 of *LNCS*, pages 329–330. Springer.

Asnar, Y., Giorgini, P., and Mylopoulos, J. (2011). Goal-driven risk assessment in requirements engineering. *Requirements Engineering*, 16(2):101–116.

Badr, I., Mubarak, H., and Göhner, P. (2008). Extending the mase methodology for the development of embedded real-time systems. In *Languages, Methodologies and Development Tools for Multi-Agent Systems*, pages 106–122. Springer.

Baeijs, C. (1998). *Emergent functionality in a society of autonomous agents:*. PhD thesis, Institut National Polytechnique de Grenoble.

Bard, M. L. (2003). *Architectural Modeling and Analysis of Complex Real-Time Systems*. PhD thesis, Mälardalen University.

Bernon, C., Gleizes, M. P., Peyruqueou, S., and Picard, G. (2002). Adelfe: A methodology for adaptive multi-agent systems engineering. In *3rd Int. Workshop on Engineering Societies in the Agents World*, volume 2577, pages 156–169. Springer.

Blanes, D., Insfran, E., and ao, S. A. (2009). Re4gaia: A requirements modeling approach for the development of multi-agent systems. In *Advances in Software Engineering*, volume 59 of *Communications in Computer and Information Science*, pages 245–252. Springer.

Bouroche, M. and Cahill, V. (2008). We don't need to agree to coordinate. In *Workshop on Dependable Network Computing and Mobile Systems (DNCMS'08) associated with the 27th IEEE Symposium on Reliable Distributed Systems (SRDS 2008)*, pages 47–51, Naples, Italy. IEEE.

Braubach, L., Pokahr, A., Krempels, K.-H., and Lamersdorf, W. (2005). Deployment of distributed multi-agent systemss. In *Engineering Societies in the Agents World V, ESAW 2004*, volume 3451 of *LNCS*. Springer.

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

Bresciani, P., Giorgini, P., Mouratidis, H., and Manson, G. (2004). Multi-agent systems and security requirements analysis. In *Software Engineering for Multi-Agent Systems II*, volume 2940 of *LNCS*, pages 352–354. Springer.

Bunse, C. and Groß, H.-G. (2006). Unifying hardware and software components for embedded system development. In *Architecting Systems with Trustworthy Components*, volume 3938, pages 120–136. Springer.

Carrasco, A., Romero-Ternero, M. C., Sivianes, F., Hernández, M. D., and Escudero, J. I. (2010). Multi-agent and embedded system technologies applied to improve the management of power systems. *JDCTA*, 4(1):79–85.

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

Cernuzzi, L., Cossentino, M., and Zambonelli, F. (2005). Process models for agent-based development. *Journal of Engineering Applications of Artificial Intelligence*, 18:205–222.

Cesário, W. O., Gauthier, L., Lyonnard, D., Nicolescu, G., and Jerraya, A. A. (2004). Object-based hardware/software component interconnection model for interface design in system-on-a-chip circuits. *Journal of Systems and Software*, 70(3):229–244.

Chella, A., Cossentino, M., Sabatucci, L., and Seidita, V. (2004). From passi to agile passi: Tailoring a design process to meet new needs. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IAT '04, pages 471–474, Washington, DC, USA. IEEE Computer Society.

Chella, A., Cossentino, M., Sabatucci, L., and Seidita, V. (2006). Agile passi: An agile process for designing agents. *International Journal of Computer Systems, Science and Engineering. Special Issue on Software.*

Chen, B. and Cheng, H. H. (2010). A review of the applications of agent technology in traffic and transportation systems. *Trans. Intell. Transport. Sys.*, 11(2):485–497.

Chun, I., Park, J., Kim, W., Kang, W., Lee, H., and Park, S. (2010). Autonomic computing technologies for cyber-physical systems. In *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, volume 2, pages 1009–1014.

Conte, G., Scaradozzi, D., Perdon, A., and Morganti, G. (2009). Multi-agent system theory for resource management in home automation systems. *Journal of Physical Agents*, 3:15–19.

Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., and Russo, W. (2008). Passim: a simulation-based process for the development of multi-agent systems. *International Journal of Agent-Oriented Software Engineering*, pages 132–170.

Culler, D., Hill, J., Buonadonna, P., Szewczyk, R., and Woo, A. (2001). A network-centric approach to embedded software for tiny devices. In *Proceedings of the First International Workshop on Embedded Software*, volume 2211 of *LNCS*, pages 114–130, London, UK. Springer.

Dam, H. K. and Winikoff, M. (2013). Towards a next-generation {AOSE} methodology. *Science of Computer Programming*, 78(6):684 – 694.

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

Dewar, J. A., Bankes, S., Hodges, J., Lucas, T., Saunders-Newton, D., and Vye, P. (1996). Credible uses of the distributed interactive simulation (dis) system. Technical report, Technical Report MR-607-A, RAND, France.

Dinkloh, M. and Nimis, J. (2003). A tool for integrated design and implementation of conversations in multi-agent systems. In *In Proc. of the 1st International Workshop on Programming Multiagent Systems: languages, frameworks, techniques and tools, ProMAS-03, held with AAMAS-03*, pages 84–98. Springer. To.

Ebrahimipour, V., Rezaie, K., and Shokravi, S. (2010). Enhanced fmea by multi-agent engineering fipa based system to analyze failures. In *Proceedings of Reliability and Maintainability Symposium (RAMS) 2010*, pages 1 – 6, San Jose, CA, USA.

Elmenreich, W. (2003). Intelligent methods for embedded systems. In *Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems*, pages 3–11.

Fairweather, I. and Brumfield, A. (2011). *LabVIEW: A Developer's Guide to Real World Integration.* Taylor and Francis.

Ganeriwal, S., Balzano, L. K., and Srivastava, M. B. (2008). Reputation-based framework for high integrity sensor networks. *ACM Transactions on Sensor Nerworks*, 4(3):1–37.

Hanninen, K., Maki-Turja, J., Nolin, M., Lindberg, M., Lundback, J., and Lundback, K.-L. (2008). The rubus component model for resource constrained real-time systems. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems*, pages 177–183. IEEE Industrial Electronics Society.

Harmon, S. J., DeLoach, S. A., and Robby (2009). Abstract requirement analysis in multiagent system design. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '09, pages 86–91. IEEE Computer Society.

Hassaine, F., Moulton, R., and Fink, C. (2009). Composing a high fidelity hla federation for littoral operations. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, pages 2087–2092. ACM.

Hassan, S., Al-Jumeily, D., and Hussain, A. (2009). Autonomic computing paradigm to support system's development. In *Developments in eSystems Engineering (DESE), 2009 Second International Conference on*, pages 273–278.

Henzinger, T. A. and Sifakis, J. (2006). The embedded systems design challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pages 1–15. Springer.

Herlea, D. E., Jonker, C. M., Treur, J., and Wijngaards, N. J. E. (1999). 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 1647, pages 8–27. Springer.

Hla, K. H. S., Choi, Y., and Park, J. S. (2010). Multi agent community to support information processing in wireless sensor network applications. *Int. J. Intell. Inf. Database Syst.*, 4(1):81–102.

Huang, H.-P., Liang, C.-C., and Lin, C.-W. (2001). Construction and soccer dynamics analysis for an integrated multi-agent soccer robot system. In *Natl. Sci. Counc. ROC(A)*, volume 25, pages 84–93.

Huebscher, M. C. and McCann, J. A. (2008). A survey of autonomic computing&mdash;degrees, models, and applications. *ACM Comput. Surv.*, 40(3):7:1–7:28.

Hutzler, G., Klaudel, H., and Wang, D. (2005). Towards timed automata and multi-agent systems. In *Formal Approaches to Agent-Based Systems*, volume 3228 of *LNCS*, pages 161–172. Springer.

Ilachinski, A. (2003). Exploring self-organized emergence in an agent-based synthetic warfare lab. *Kybernetes: The Int. Journal of Systems & Cybernetics*, 32(1):38–76.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process.* Addison-Wesley.

Jamont, J.-P., Médini, L., and Mrissa, M. (2014a). A web-based agent-oriented approach to address heterogeneity in cooperative embedded systems. In *Trends in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*, volume 293 of *Advances in Intelligent Systems and Computing*, pages 45–52. Springer International Publishing.

Jamont, J.-P. and Occello, E. M. M. (2011). A framework to simulate and support the design of distributed automation and decentralized control systems: Application to control of indoor building comfort. In *IEEE Symposium on Computational Intelligence in Control and Automation*, pages 80–87, Paris, France. IEEE.

Jamont, J.-P. and Occello, M. (2007). Designing embedded collective systems: The diamond multiagent method. In *19th IEEE International Conference on Tools with Artificial Intelligence*, pages 91–94.

Jamont, J.-P. and Occello, M. (2013). Using MASH in the context of the design of embedded multiagent system. In *Proc. of the 11th Int. Conference on Practical Applications of Agents and Multiagent Systems*, Advances in Intelligent and Soft Computing. Springer.

Jamont, J.-P., Occello, M., and Lagrèze, A. (2010). A multiagent approach to manage communication in wireless instrumentation systems. *Measurement*, 43(4):489–503.

Jamont, J.-P., Occello, M., and Mendes, E. (2011). Real world embedded multiagent systems: Simulation of hardware/software mixed agent societes in realistic physical models. In *Proceedings of the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 435–436, Lyon, France. IEEE Computer Society.

Jamont, J.-P., Occello, M., and Mendes, E. (2013). Decentralized intelligent real world embedded systems: a tool to tune design and deployment. In *Proceedings of the 11th International Conference on Practical Applications of Agents and Multiagent Systems*, Advances in Intelligent and Soft Computing. Springer.

Jamont, J.-P., Raievsky, C., and Occello, M. (2014b). Handling safety-related non-functional requirements in embedded multi-agent system design. In *Proceedings of the 12th International Conference on Practical Applications of Agents and Multi-Agent Systems.* Springer.

Jaszczyk, P. and Król, D. (2010). Updatable multi-agent osgi architecture for smart home system. In *Agent and Multi-Agent Systems: Technologies and Applications*, volume 6071 of *LNCS*, pages 370–379. Springer.

Jawawi, D., Deris, S., and Mamat, R. (2006). Enhancements of pecos embedded real-time component model for autonomous mobile robot application. In *Proceedings of the IEEE International Conference on Computer Systems and Applications*, pages 882–889. IEEE Industrial Electronics Society.

Jez, G. (2011). Kurasu : Colonie d?objets semi-vivants. Technical report, Grenoble INP, CEA Grenoble.

Johnson, C. (2005). The glasgow-hospital evacuation simulator: Using computer simulations to support a risk-based approach to hospital evacuation. Technical report, University of Glasgow.

Julian, V. and Botti, V. (2004). Developing real-time multi-agent systems. *Integr. Comput.-Aided Eng.*, 11:135–149.

Kephart, J. and Chess, D. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Lakner, R., Németh, E., Hangos, K. M., and Cameron, I. T. (2006). Agent-based diagnosis for granulation processes. In Marquardt, W. and Pantelides, C., editors, *16th European Symposium on Computer Aided Process Engineering and 9th International Symposium on Process Systems Engineering*, volume 21 of *Computer Aided Chemical Engineering*, pages 1443 – 1448. Elsevier.

Le, V. T., Bouraqadi, N., Stinckwich, S., Moraru, V., and Doniec, A. (2009). Making networked robots connectivity-aware. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pages 1835–1840, Piscataway, NJ, USA. IEEE Press.

Lee, E. A. (2002). Embedded software. In *Advances in Computers*, page 2002. Academic Press.

Leitao, P., Marik, V., and Vrba, P. (2013). Past, present, and future of industrial agent applications. *Industrial Informatics, IEEE Transactions on*, 9(4):2360–2372.

Leveson, N. G. (2004). A new accident model for engineering safer systems. *Safety Science*, 42(4):237–270.

Lind, J. (2001). *Iterative Software Engineering for multiagent systems: The MASSIVE Method*, volume 1994. Springer.

Lindoso, A. N. and Girardi, R. (2006). The sramo techique for analysis and reuse of requirements in multi-agent application engineering. In *WER*, pages 41–50.

Liu, L. and Yu, E. S. K. (2004). Designing information systems in social context: a goal and scenario modelling approach. *Inf. Syst.*, 29(2):187–203.

Massawe, L. V., Aghdasi, F., and Kinyua, J. (2009). The development of a multi-agent based middleware for rfid asset management system using the passi methodology. *Information Technology: New Generations, Third International Conference on*, 0:1042–1048.

Mendes, M., Santos, B., and da Costa, J. S. (2010). A matlab/simulink multi-agent toolkit for distributed networked fault tolerant control systems. In *Proceedings of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*.

Meng, Y. (2005). An agent-based reconfigurable system-on-chip architecture for real-time systems. *2nd Int. Conference on Embedded Software and Systems*, 0:166–173.

Mercier, A., Raïevsky, C., Occello, M., and Genthial, D. (2013). Solutions multi-agents pour la prise en charge à domicile des séniors. *Ingénierie des Systèmes d'Information*, 18(6):83–112.

Möller, A., Fröberg, J., and Nolin, M. (2004). Industrial requirements on component technologies for embedded systems. In *Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings*, pages 146–161.

Monostori, L., Vancza, J., and Kumara, S. (2006). Agent-based systems for manufacturing. *PCIRP Annals - Manufacturing Technology*, 55(2):697–720.

Moore, H. (2011). *MATLAB for engineers*. ESource–the Prentice Hall engineering source. Prentice Hall.

Mrissa, M., Medini, L., Jamont, J.-P., Le Sommer, N., and Laplace, J. (2015). An avatar architecture for the web of things. *Internet Computing, IEEE*, 19(2):30–38.

Muller, J.-P. (2004). Emergence of collective behaviour and problem solving. In Omicini, A., Petta, P., and Pitt, J., editors, *Engineering Societies in the Agents World IV*, volume 3071 of *Lecture Notes in Computer Science*, pages 1–20.

Müller, J. P. and Fischer, K. (2014). Application impact of multi-agent systems and technologies: A survey. In Shehory, O. and Sturm, A., editors, *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, pages 27–53. Springer.

Munroe, S., Miller, T., Belecheanu, R., Pechoucek, M., McBurney, P., and Luck, M. (2006). Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. *Knowledge Eng. Review*, 21(4):345–392.

Mylopoulos, J., Chung, L., and Yu, E. S. K. (1999). From object-oriented to goal-oriented requirements analysis. *Commun. ACM*, 42(1):31–37.

Naji, H. R., Wells, B. E., and Etzkorn, L. (2004). Creating an adaptive embedded system by applying multi-agent techniques to reconfigurable hardware. *Future Gener. Comput. Syst.*, 20:1055–1081.

Occello, M., Jamont, J., Guillermin, R., and Pezzin, M. (2008). A multiagent approach for an uwb location embedded software architecture. In *Fifth IEEE/ACM International Conference on Soft COmputing as Transdisciplinary Science and Technology*, pages 279–285, Paris. ACM.

Omicini, A. (2001). Soda: Societies and infrastructures in the analysis and design of agent-based systems. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 185–193. Springer Berlin Heidelberg.

Padgham, L., Winikoff, M., and Poutakidis, D. (2005). Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, 18(2):173 – 190. Agent-oriented Software Development.

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

Pechoucek, M. and Maríк, V. (2008). Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems*, 17:397–431.

Pedroni, V. A. (2004). *Circuit Design with VHDL*. The MIT Press.

Picard, G. and Gleizes, M.-P. (2004). The adelfe methodology - designing adaptive cooperative multi-agent systems (chapitre 8). In *Methodologies and Software Engineering for Agent Systems: The AOSE handbook*, pages 157–176. Kluwer Publishing.

Polaków, G. and Metzger, M. (2009). Agent-based framework for distributed real-time simulation of dynamical systems. In *Proc. of the 3rd KES Int. Symp. on Agent and Multi-Agent Systems: Technologies and Applications*, pages 213–222, Berlin. Springer.

Ponci, F., Deshmukh, A., Monti, A., Cristaldi, L., and Ottoboni, R. (2005). Interface for multi-agent platform systems. In *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, pages 2226 – 2230. IEEE.

Pottie, G. J. and Kaiser, W. J. (2009). *Principles of Embedded Networked Systems Design*. Cambridge University Press.

Raïevsky, C., Mercier, A., Genthial, D., and Occello, M. (2014). Doubt removal for dependant people from tablet computer usage monitoring. In *Highlights of Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection - PAAMS 2014 International Workshops, Salamanca, Spain, June 4-6, 2014. Proceedings*, pages 44–53.

Raja, A., Barley, M., and Zhang, X. (2009). Towards safe coordination in multi-agent systems. In *Safety and Security in Multiagent Systems*, volume 4324 of *LNCS*, pages 1–7. Springer.

Robinson, C. R., Mendham, P., and Clarke, T. (2010). A multiagent approach to manage communication in wis. *Journal of Physical Agents*, 4(3):489 – 503.

Rodriguez-Fernández, C. and Gómez-Sanz, J. J. (2010). Self-management capability requirements with selfmml & ingenias to attain self-organising behaviours. In *Proc. of the 2nd Int. Workshop on Self-organizing Architectures*, pages 11–20. ACM.

Russell, S. J. and Norvig, P. (1995). *Artificial intelligence - a modern approach: the intelligent agent book*. Prentice Hall series in artificial intelligence. Prentice Hall.

Salzwedel, H. (2011). Comparison of can, flexray, and ethernet architectures for the design of abs systems. *SAE Technical Paper*.

Shakshuki, E. and Malik, H. (2007). Agent based approach to minimize energy consumption for border nodes in wireless sensor network. In *Proc. of the 21st Int. Conf. on Advanced Networking and Applications*, pages 134–141, USA. IEEE.

Sichman, J., Conte, R., and Demazeau, Y. (1994). A social reasoning mechanismbased on dependence networks. In *Proceedings of European Conference on Artificial Intelligence - ECAI'94*, Amstardam.

Spanoudakis, N. I. and Moraitis, P. (2007). An ambient intelligence application integrating agent and service-oriented technologies. In *SGAI Conf.*, pages 393–398.

Sterling, L. and Taveter, K. (2009). *The Art of Agent-Oriented Modelling*. MIT Press.

Stewart, D. B., Volpe, R., and Khosla, P. K. (1997). Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Transaction on Software Engineering*, 23(12):759–776.

Takimoto, M., Mizuno, M., Kurio, M., and Kambayashi, Y. (2007). Saving energy consumption of multi-robots using higher-order mobile agents. In *Agent and Multi-Agent Systems: Technologies and Applications*, volume 4496 of *LNCS*, pages 549–558. Springer.

Tesauro, G., Chess, D. M., Walsh, W. E., Das, R., Segal, A., Whalley, I., Kephart, J. O., and White, S. R. (2004). A multi-agent systems approach to autonomic computing. In *Proceedings of the Third International Joint Conference on Autonomous Agents and*

Multiagent Systems - Volume 1, AAMAS '04, pages 464–471, Washington, DC, USA. IEEE Computer Society.

van Breemen, A. and Vries, T. D. (2000). An agent-based framework for designing multi-controller systems. In *Proc. of the Fifth International Conference on The Practical Applications of Intelligent Agents and Multi-Agent Technology*, pages 219–235.

van Ommering, R. C., van der Linden, F., Kramer, J., and Magee, J. (2000). The koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85.

Ward, P. T. and Mellor, S. J. (1989). *Structured Analysis for Real Time Systems.* Prentice-Halls.

Werneck, V., Kano, A. Y., and Cysneiros, L. M. (2007). Evaluating adelfe methodology in the requirements identification. In *10th Workshop in Requirements Engineering*, pages 13–24.

Weyns, D., Schelfthout, K., Holvoet, T., and Lefever, T. (2005). Decentralized control of e'gv transportation systems. In *4rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems - Industrial Applications*, pages 67–74. ACM.

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

Wu, C.-L., Liao, C.-F., and Fu, L.-C. (2007). Service-oriented smart-home architecture based on osgi and mobile-agent technology. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 37(2):193–205.

Zurawski, R. (2009). *Embedded Systems Handbook, Second Edition: Networked Embedded Systems.* CRC Press.