Mouvement, Interaction, Calcul partout et à tout moment avec l'Ordinateur

Abdelkader Gouaich

Numéro d'identification :

Académie de Montpellier

Université Montpellier II

— **Sciences et Techniques du Languedoc** —

Thèse

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

Spécialité	: Informatique
Formation Doctorale	: *Informatique*
Ecole Doctorale	: *Information, Structures, Systèmes*

Movement, Interaction, Calculation as Primitives for Everywhere & Anytime Computing

par

Abdelkader GOUAICH

Soutenue le 01 Juillet 2005 devant le Jury composé de :

Mr. Luzeaux Dominique, HDR, DGA/DCE/CTA, ... Rapporteur
Mr. Omicini Andrea, Professeur, Università di Bologna (Italie), ... Rapporteur
Mr. Ferber Jacques, Professeur, Université Montpellier II, ... Président
Mr. Sallantin Jean, Directeur de Recherche, LIRMM/CNRS, ... Examinateur
Mr. Cerri A. Stefano, Professeur, Université Montpellier II, ... Directeur de Thèse
Mme. Charlton Patricia, Senior Research Engineer, Motorola Labs (UK), ... Co-Directeur de Thèse

A ma grand-mère 'Aya'.

Remerciements

Je remercie respectueusement mes directeurs de thèse, le Professeur Stefano A. Cerri et Patricia Charlton, pour leurs conseils et encouragements exprimés tout le long de mon parcours de thèse.

Mes respects et ma gratitude vont également aux membres de mon jury qui m'ont fait l'honneur de juger ce travail et qui par leur disponibilité, leurs observations et leurs rapports m'ont permis d'enrichir mon travail.

Je tiens à remercier les membres du laboratoire d'informatique, robotique et microélectronique de Montpellier (LIRMM) de m'avoir accueilli pendant le déroulement de la thèse. Mes remerciements vont également aux membres du centre de recherche de Motorola Labs à Gif-sur-Yvettes qui ont initié les contacts afin de permettre que la thèse débute comme une collaboration entre le LIRMM et Motorola Labs. Je pense tout particulièrement à messieurs Bernard Burg (Morotola Labs) et Jean Sallantin (LIRMM).

Même si j'ai remercié formellement madame Patricia Charlton en tant que co-directrice de thèse, je me dois de réitérer mes remerciements car elle fut plus que cela. Il est certain, que sans ses analyses à la fois pertinentes et objectives, sans sa patience, je ne serais jamais arrivé à atteindre mes objectifs de thèse.

...

'Ne demande jamais ton chemin à quelqu'un qui le connaît, tu risques de ne pas te perdre'

Après quelques années de thèse je me rends compte, effectivement, qu'il est agréable de se perdre et de découvrir des sentiers imprévus. Mais je dois ajouter, qu'il est encore plus agréable de ne pas se perdre seul! Et bien pour ma part, j'ai eu la chance d'avoir comme compagnon dans la perdition scientifique Mr. Fabien Michel. Alors Fabien, j'espère pour l'avenir, qu'on ne trouvera jamais notre destination finale!

Je remercie tous les gens que j'ai côtoyé à Montpellier: Jose Baez, Toufik Bennouas, Gregory Beurrier, Jérome Chappelle, Christopher Dartnell, Celine Fiot, Simon Jaillet, Adrojan et Melinda Kiss , Daniele Maraschi, Denis Payet, Chedy et Ramy Raıssi, Didier Schwab, John Tranier, Mehdi Yousfi Monod.

Merci Isabelle pour ton investissement dans l'organisation le jour de la soutenance.

Merci à Lylia.

Merci à mon frère Khaled, à ma soeur Ibticem et à mes parents.

Contents

Chapter 1

The "Everywhere, Anytime, Computing" Paradigm

1.1 Introduction

The technological context of the software systems is continuously evolving. Nowadays, the availability of small communicating devices offers the opportunity to make the *'Everywhere, Anytime Computing'* (EAC) a reality that naturally integrates our societies and economies.

In this thesis we define the meaning of the Everywhere, Anytime Computing concept to that is built from three key concepts: *(i)* the concept of space and movement, *(ii)* the concept of time, and *(iii)* the fact that computers are involved. Obviously, these keywords are a simplification of a broader vision.

One may interpret the EAC as the ability to compute, in the sense of calculate, everywhere and at anytime. Obviously, this interpretation is very minimal and does not present any ambitious evolution of the current computer-based services (e-services) since these services already exist. For instance, any small calculator fits within this definition and offers a computation service to the users. So, the interpretation of the 'computing' is more related on the ability to offer *high-level* services rather than the primary and historical function of the computers that is calculation.

Next generation e-services are feasible when you consider the breadth of interpreting EAC concept. The innovation of the EAC can be captured only when introducing the communication and interaction abilities of the devices. In fact, the e-services are no longer closed, but they are open and interact contextually with their surroundings to fulfill their functions.

At the very heart of next generation e-services is the fact that although e-services are defined in particular places they are also available persistently and transparently everywhere.

The persistence and transparency of the e-services make them *ubiquitous*. The ubiquity of the devices and their services has been initially introduced by Mark Weiser [Weiser, 1991]. M. Weiser was one of the first to create a new, yet realistic, perspective on the evolution of informatics over the next twenty years by shifting the primary goal of computing from the role of calculating to the offering of services [Weiser & Brown, 1997]. However, there are differences between Weisers ubiquitous computing perspective and the vision is provided by this thesis.

In fact, the Weiser's vision emphasizes on the presence of the computers and software services in a persistence manner around the human user. The human user is the central entity of the system and all the interactions are defined between the human user and the ubiquitous services. So, the openness and interactions between the software services in order to offer more elaborated services are not considered as fundamental features. In contrast, our work considers the openness and interaction among the e-services as fundamental features of the EAC.

Currently, the term ubiquitous computing is overloaded. There are several other terms that appeared such as: pervasive computing, ambient computing, wearable computers, ad hoc computing etc. which imply some degree or sense of providing ubiquitous behaviour as a property of the system . Generally, these terms are defined to indicate different levels of granularity of the offered service. For instance, the service that manages the intensity of the light in a room does not have the type of complexity as the service that delivers courses and planning for the students in a university campus. In fact, while the former service can be performed in a stand-alone manner, the later may imply negotiation mechanisms and interactions in order to offer the right course at the right time to the students. Still, all these definitions fit in the same logical framework that is presented in the next section.

1.2 Definition of the EAC Context

In order to avoid ambiguities, we propose a unified framework to capture the main important features of the EAC context. Our framework belongs to the context of addressing EAC type applications.

The EAC context describes the software systems that exhibit the following features:

- they are open and delimited[1];
- they move within an abstract or physical topology;
- the movement of the systems causes their interactions; when the systems are interacting, a point to point communication link is established among them.
- when the systems are interacting, they perform computations and exchange data in order to achieve their design goals.
- the movement of the systems causes also their disconnection. In this case, the communication link disappears and the systems cannot communicate.

Hence, the above results in self-contained software systems, each system exhibits a property of openness and thus interaction must occur between such systems if they are to maintain this feature of self contained. It is the movement of the software systems within an abstract or physical topology that induces their connection and disconnection. When connected, the software systems can exchange data through the established point-to-point communication link. This communication link disappears as soon as the software systems are disconnected. This way to view the software systems is defined as the *EAC paradigm*.

It is worth noting that the management of the movement of the software systems and establishment of the communication links are external to the EAC framework.

[1]For the definition of the openness and delimitation of the software systems: (cf. §2.2, page 19).

1.3 Problem and Thesis Statements

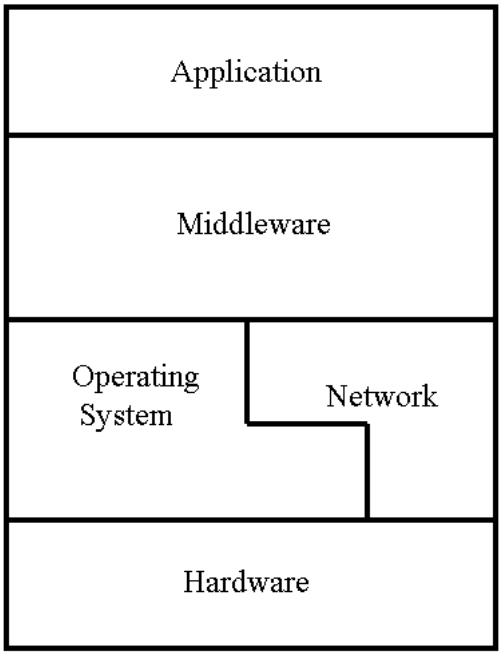


Figure 1.1: Positioning of the work at the middleware layer.

Figure 1.1 describes a logical decomposition of the EAC concerns in term of layers. Our work fits in the middleware layer. This layer is the intermediary layer between the operating system, network technologies and the applications. The middleware layer offers an approach along with libraries and tools to facilitate the design and implementation of the applications by using the services offered by the lower-layers.

More precisely, the lower-layers are assumed to offer the following elements:

- the events of connection with external systems: in this case a direct point-to-point link is established.
- the events of disconnection from external systems: in this case the communication link is no more available.
- the communication link is assumed to be bi-directional and reliable. The reliability of the communication link means that *(i)* the order of the emitted messages is preserved (Fist In First Out); and *(ii)* the failure of delivering a message is notified.

Up until now there has not been an EAC solution that supports the many levels of design in a single framework. This thesis produces such a framework that deals with:

1. the constraints of the communication medium in terms of intermittence of the communications;
2. the management of the on-the-fly composition and decomposition of the software systems;
3. the autonomy of the software entities and systems;

4. the management of the interoperability of the open systems.

A summary of the solution provided by this framework is as follows:

1. the use of the generative communication paradigm which is suitable to handle the constraints of the communication medium;
2. the use of formal algebraic structure in order to explicitly specify and implement the on-the-fly composition of the software systems;
3. the internal integrity of the software entities is a *sine qua none* condition to ensure the autonomy of the software entities;
4. the coordination point of view is used as a mean for the interoperability of autonomous and open systems.
5. the use of a social metaphor that is expressive and powerful metaphor and reduces the complexity of the engineering process. Hence, each software system is viewed as an open and self-contained society of autonomous individuals that interacts and coordinates some joint activities with other societies.

1.4 Contributions

The main contributions of the thesis are summarized as follows:

1.4.1 MIC*

The development of the MIC* environment makes a contribution to both the research fields of the blackboard paradigm and multi-agent systems.

The contribution to the blackboard paradigm and tuple space architectures is in the structuring of the communication medium in a way that guarantees the autonomy of action of the entities and eases the on-the-fly composition of software systems. Also, the information and interaction carriers are structured. So, in contrast with the tuple space approaches, MIC* provides a unified structure to carry information and to represent the interaction means.

The contribution to the multi-agent systems is on the identification of the autonomy as an important feature of this paradigm and the definition of objective criterion that guarantees the autonomy of the software entities at the implementation level. The internal integrity was found as a precondition for the autonomy. Hence, the MIC* approach emphasizes the central role of the deployment environment of the autonomous agents to guarantee their autonomy. Besides, the multi-agent system paradigm does not specify any particular paradigm of communication between the agents. For the EAC context we have suggested to use the generative communication as a communication paradigm. Consequently, the intermittence of the communications does not drastically affect the functioning of the systems.

1.4.2 Coordination and Conversation Protocols

The interoperability of open software systems is a key issue for the EAC. The interoperability has been addressed from a coordination point of view. Hence, independent systems may collaborate and execute joint activities at least if they agree on a common coordination protocol.

We have developed a dependency-based coordination model based on the coordination theory of Malone and Crowston [Malone & Crowston, 1994].

The contribution that has been made is to link the coordination protocols with their related conversations. Hence, the structure of the conversation between individuals that are in a coordination process is completely defined and one has to check the consistency of the coordination process by checking the consistency of the conversations.

This work has been integrated with the MIC* environment. Consequently, the deployment environment guarantees that the defined coordination protocols are actually followed by the autonomous software entities by observing the ongoing conversations.

1.4.3 EAC Engineering and Simulation Tools

To enable the benefits of the MIC* environment and the coordination theory developed in this thesis to be applied to an EAC context we have developed a social model of the software system. Within this social metaphor, the software system is considered as an open society of agents that interact with other societies in a *spontaneou*s manner. So, new organisational concepts have been introduced such as the internal and interface roles, intracives and extracives in order to adapt the social metaphor to the EAC context.

An integrated develompment environment (IDE) allows the modelling of the artificial society and generates automatically the source code of the software system using the MIC* environment and the coordination framework.

1.5 Outlines

The rest of the thesis is organized as follows:

- **Chapter 2**: This chapter describes more precisely the constraints and hypotheses of the EAC context and selects some existing approaches, which provide some of the key features, results and designs that are used and extended to create our solution.
- **Chapter 3**: This chapter presents the MIC* environment that: *(i)* offers a generative communication; *(ii)* guarantees the internal integrity of the software entities; *(iii)* models the on-the-fly composition of software systems.
- **Chapter 4**: This chapter deals with the interoperability that is treated from a coordination point of view. An introduction to the coordination theory and the state of the art of coordination models are presented in this chapter. Then the dependency-based coordination model is presented along with some methods in order to check the consistency of the coordination protocols by observing the ongoing conversations among the software entities.
- **Chapter 5**: This chapter presents an engineering approach to model software systems in the context of the EAC using the social metaphor. After the review of current organisational models to specify agent societies, we make a proposition of an organisational model that is adapted to the EAC context. An integrated development tool (IDE) that follows the organisational model is presented in this chapter and helps in *(i)* the modelling the software systems and *(ii)* the implementation of the software systems

by generating automatically their source code. The experiments with the developed applications are conducted in a virtual 3D simulator.

Chapter 2

State of the art

"Vérité dans un temps, erreur dans l'autre."
Charles de Montesquieu.

2.1 Introduction

This chapter studies what are the relevant constraints and hypotheses that characterize the EAC context. Starting from this study, some existing approaches such as the multi-agent system (MAS) and blackboard (BB) paradigms are selected as pertinent starting points since they answer some aspects of the EAC.

This chapter is organized as follows: section [2.2, page 19] presents the glossary of the terms that are used in the rest of the chapters; section [2.3, page 24] presents the constraints and hypotheses of the EAC; section [2.4, page 32] presents the BB paradigm and its technologies; section [2.5, page 38] presents the MAS paradigm and its technologies; finally, section [2.6, page 47] concludes this chapter.

2.2 Glossary

Before presenting the state of art, this section reviews a set of requirements common to most EAC related systems. Each term used to capture aspects of the requirements is described and working definitions are provided. The relationship between these are presented as a conceptual graph in Figure 2.1.

2.2.1 Definitions

Datum

A datum is a particular encoding (or signal) of information. The datum may be stored in the computer memory or exchanged through a communication medium.

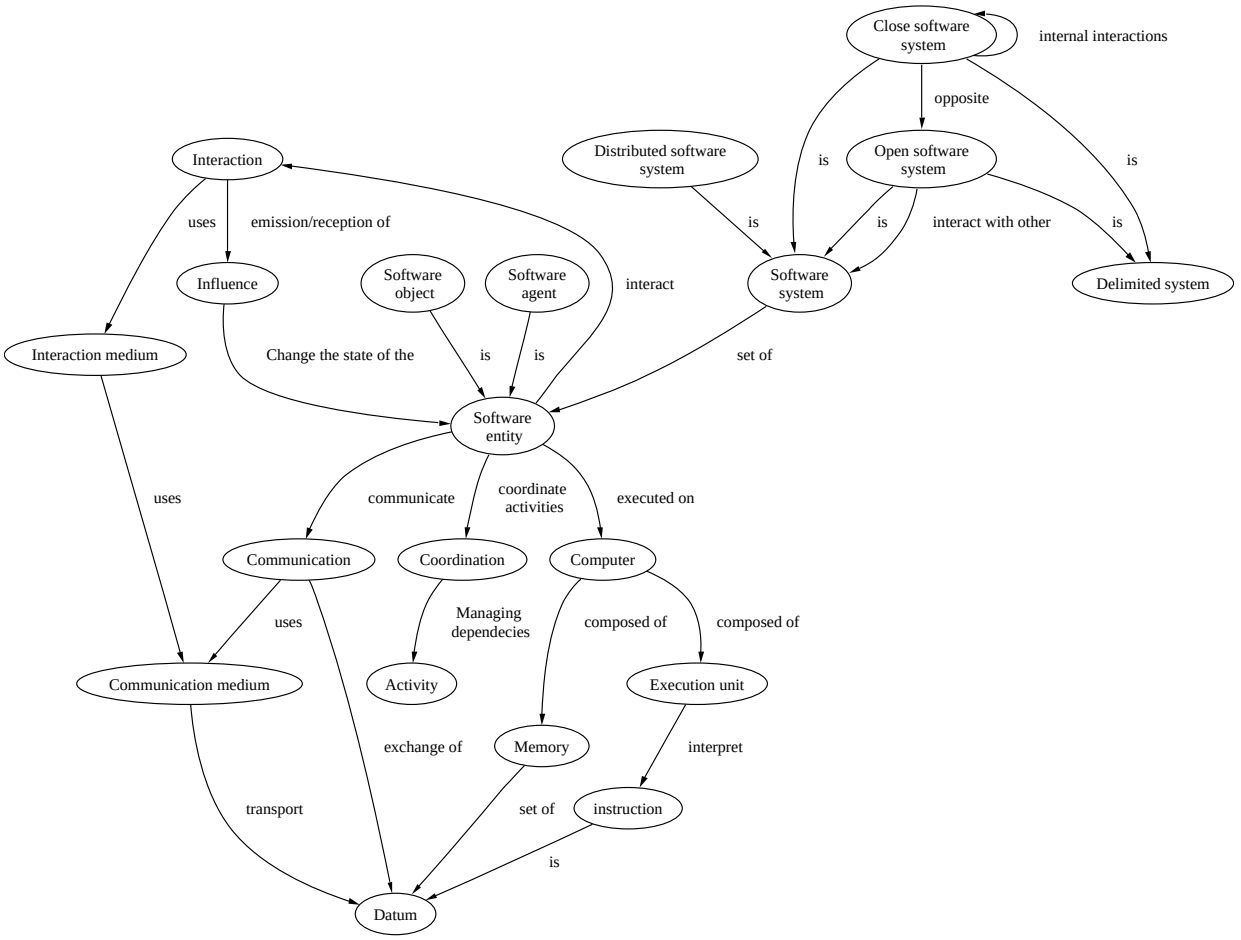


Figure 2.1: The glossary of the terms and their relationships presented as a conceptual graph.

Computer

Any device that is equivalent to the Turing machine [Turing, 1936] is considered as a computer. The computer owns a memory, used to store *data*; and an execution unit, used to interpret instructions. The program is a sequence of instructions. The interpretation of a program is viewed as a series of elementary modifications of the memory with regards to the semantics of the instructions.

Software Entity

This generic term refers to any datum or program that is stored in the memory or interpreted by the execution unit.

Software Object

The object-oriented (OO) paradigm [Booch, 1994] organizes the software entities as a set of features and a set of methods. This structure is known as the software object. The 'class' defines the model of a set of objects. The inheritance is a relationship that allows the reusability and extension of the existing classes by new classes.

Multi-Agent System (MAS)

A MAS is a system composed of multiple interacting agents, where each agent is a coarse-grained computational system in its own right. An *agent-based* system is one in which the key abstraction used is that of an agent.

Software Agent

An agent is an encapsulated software entity which exhibits the following characteristics [Gallimore *et al.* , 1998]:

- autonomy: operates without the direct intervention of humans or other agents and has control over its actions and internal state;
- responsiveness: perceives its environment (which may be the physical world, a user, a collection of agents, etc.) and responds in a timely fashion to changes that occur in it;
- proactiveness: does not simply act in response to its environment, but exhibits opportunistic, goal-directed behavior and takes the initiative where appropriate;
- social ability: interacts, when it deems appropriate, with other agents in order to complete its own problem solving and to help others with their activities.

Communication

The communication is a process that exchanges data among different software entities through a communication medium.

Interaction

The interaction is a special process of the communication where the interacting entities exchange influences to mutually change their states. These modifications affect their future behaviors. The influences are reified as data and carried by the interaction medium.

Software System

A software system is a set of software entities that have been aggregated to accomplish some particular functions.

Delimited System

A delimited system owns a clearly defined frontier. Consequently, one may know the inside and outside of the system.

Distributed Software System

The software entities of a distributed software system are distributed over several computers and linked by a communication medium.

Open Software System

An open software system is a delimited software system where the interaction is not limited to the internal elements but extended to the external world.

Closed Software System

By contrast to an open software system, the interaction and communication occur only between the internal elements of a closed system.

Communication Medium

The communication medium carries data from one point to another. Network technologies are examples of communication media. An introduction to network technologies is presented in Annex F.

Interaction Medium

The interaction medium carries the influences of the interacting entities. Usually, within informatics the influences are reified as data and carried by a communication medium.

Activity

A particular sequence of instructions is abstracted as a functional activity. An activity may need resources in order to be executed and can produce other resources when finalized.

Coordination

According to Malone and Crowston [Malone & Crowston, 1994] the coordination is defined as managing dependencies among the activities. The coordination protocol expresses these dependencies and gives the means to manage them.

2.2.2 Using the Terms in Examples

The terms are used in the following examples to give a more precise idea of their semantics. When a term appears, it is highlighted and written in italic.

Introduction to Informatics

The distinction between the information and its representation as *data* is one of the fundamental notion of the informatics. According to C. Shannon (1916-2001), the information is an immaterial concept that reduces, for an external observer, the uncertainty about the state of a physical or abstract system. For instance, the following sentence: *"Ten students are in the classroom"* gives information about the state of the classroom: currently only ten students are present. To materialize this information, one has to encode it as a *datum*. Thus, the presented sentence is a sequence of characters and words that represent the encoding of the information. The information is independent from its encoding. For instance, by using mathematical notations, the previous information is represented as follows: "room_number_student = 10". The presented *data* are encoded in a human readable manner. The *computers* can only understand *data* written using the binary system. In fact, the *computer* is an electronic device and the most elementary information that can be encoded using an electric signal owns only two states *(i)* the signal is present, by convention this state is represented as 1; *(ii)* the signal is absent, by convention this state is represented as 0. All *data* in informatics are sooner or later represented as a sequence of 1 and 0. The *computer* is composed of a *memory* and an *execution unit*. The *memory* can be viewed as a table, where the rows are numbered and contain *data*. The *execution unit* reads the *data* from the *memory*, and according to the semantics of an *instruction* modifies this *memory*. Thanks to Alan Turing (1912-1954), the *instructions* are also considered as *data* in modern *computers*. So, programs are also stored and read from the *memory*. According to A. Turing, one can summarize the function of a computer as a process that modifies the *memory* from an initial state to a final state according to the semantics of a set of *instructions*.

The *'software entity'* is a generic term that refers to any structure that is represented as a *datum* and stored in the *memory* of a *computer* (text files, image files, programs, databases and so on). A *software system* is a set of *software entities* which have been aggregated to accomplish a certain function. The operating systems are the common *software systems* met by the users. The operating system is a set of *software entities* that eases the use of the *computer's* hardware resources. As examples of operating systems, one can mention the following: Linux, Unix™, Windows XP™, Windows CE™.

For software engineering purposes, the *software objects* appeared as structured software entities composed of a set of features and methods that locally modifies these features. On the other hand, *software agents* appeared as an extension of the *objects* with some higher-level features.

Autonomy is one of the characteristics used to differentiate between an *object*-based system and an *agent*-based system. The notion of autonomy in *agent* systems is intended to attribute the *agent* with full control of its state of behaviour.

Modern Informatics: *Computers + Communication Media*

The *computers* that are distributed over different locations can be linked by a *communication medium* to exchange *data*. So, the programs are no longer restrained to local *data* but have access to the *data* of remote *software entities*. Within informatics, the process of exchanging *data* through a *communication medium* is called *communication*.

For a *software system*, when the *software entities* are distributed over different *computers* and can *communicate*, this system is called a *distributed system*. The *communication medium* introduces also the notion of *openness* and *closeness* of the *software systems*. The elements of a *closed* system communicate internally, while the elements of an *open* system communicate with entities located beyond the frontier of the system.

The *interaction* is distinguished from the *communication*. In fact, while the *communication* assumes only the exchange of *data*; the *interaction* goes further and assumes that the exchanged *data* modify the state of the *entities*. In this context, we speak about *influences* rather than *data*, so that the *communication* medium becomes the *interaction* medium.

The *coordination* specifies the dependencies of joint *activities* that are carried out by several distributed software entities. Hence, the *coordination* protocol specifies for each *entity* what action to perform and when to perform it.

2.3 Constraints and Hypotheses of the EAC

In order to present a coherent and relevant state of the art, one has to make a preliminary study of the constraints and hypotheses of the EAC. This preliminary study is used in order to select the existing works that may constitute relevant starting points for our work.

From the software engineering point of view, we have considered the following major constraints and hypotheses of the EAC:

- **Constraints of the Communication Medium**: Within the EAC, communication media are not centralized but appear locally and spontaneously as soon as the physical or logical communication links can be established. Besides, the communication between the software entities can be intermittent and disturbed. These disturbances have not to be considered as network problems but as hypotheses of work to be taken into account at the conceptual and implementation levels.
- **Hypothesis of the Autonomy**: Traditional software engineering approaches assume to have a complete knowledge and control of the software entities that compose the software system. However, this assumption is challenged in open and large-scale distributed systems. In fact, several systems developed by different authorities have to communicate and interact. For example, let us consider a typical scenario of an *Internet*-based application where an open software system developed by a company 'X' interacts with another software system developed by a company 'Y'. Neither of these companies controls the global software system that is made by the composition of the two sub-systems:

'X' plus 'Y'. However, these independent systems have to interact and coordinate their actions to offer their functions. The lack of knowledge and means to control the behavior of a software entity makes this entity behaving autonomously. The EAC assumes the autonomy of the software entities and systems. So, the autonomy feature has to be considered at the conceptual and implementation levels.

- **Hypothesis of the Openness**: As mentioned in section [1.2, page 14] the EAC assumes the openness of software systems. In fact, the added value of the EAC is created by the composition of open and interacting services. So, the EAC offers the potential to provide a new generation of e-services, only because new functionalities can emerge by the spontaneous interactions of more elementary services.

2.3.1 Dealing with the Constraints of the Communication Medium

Within information systems, software entities communicate using one of three paradigms: *(i)* synchronous communication, *(ii)* asynchronous communication and *(iii)* generative communication.

Synchronous Communication

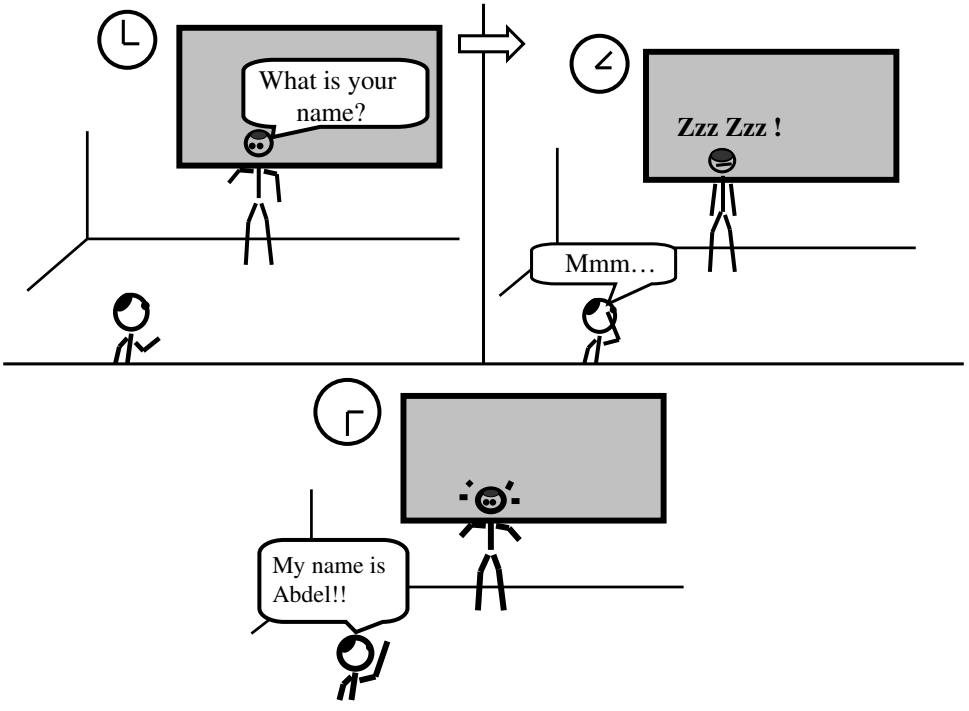


Figure 2.2: Informal example of the synchronous communication paradigm.

Figure 2.2 represents an imaginary classroom where a teacher gives a course to students. In the synchronous communication paradigm, when the teacher communicates with a student, she/he sends a message and stops all her/his activities until the student responds. For in-

stance, this sketch is used when the teacher asks a question and waits for the answer. Within informatics, this is similar to a function call. In fact, the communication process is considered as an evaluation of a function that returns a result. The entity that calls the function provides the parameters and waits until receiving the result. The Remote Procedure Call (RPC) [Birrell & Nelsen, 1984] is the exact mimic of the function call when the entities are distributed and joined by a communication medium. Currently, the synchronous communication paradigm is the dominant paradigm for building distributed and open software systems.

The following technologies follow this paradigm: RPC [Birrell & Nelsen, 1984], Common Object Request Broker Architecture[1] (CORBA) [Vinoski, 1997][OMG, 2002], Java™ Remote Method Invocation (RMI) [Sun-Microsystems, 1994], Hyper-Text Transport Protocol (HTTP) [Fielding *et al.* , 1999], Wireless Application Protocol (WAP™) [WAP-Forum, 2002], Service Object Access Protocol (SOAP) and Web Services [Gudgin *et al.* , 2003].

Asynchronous Communication

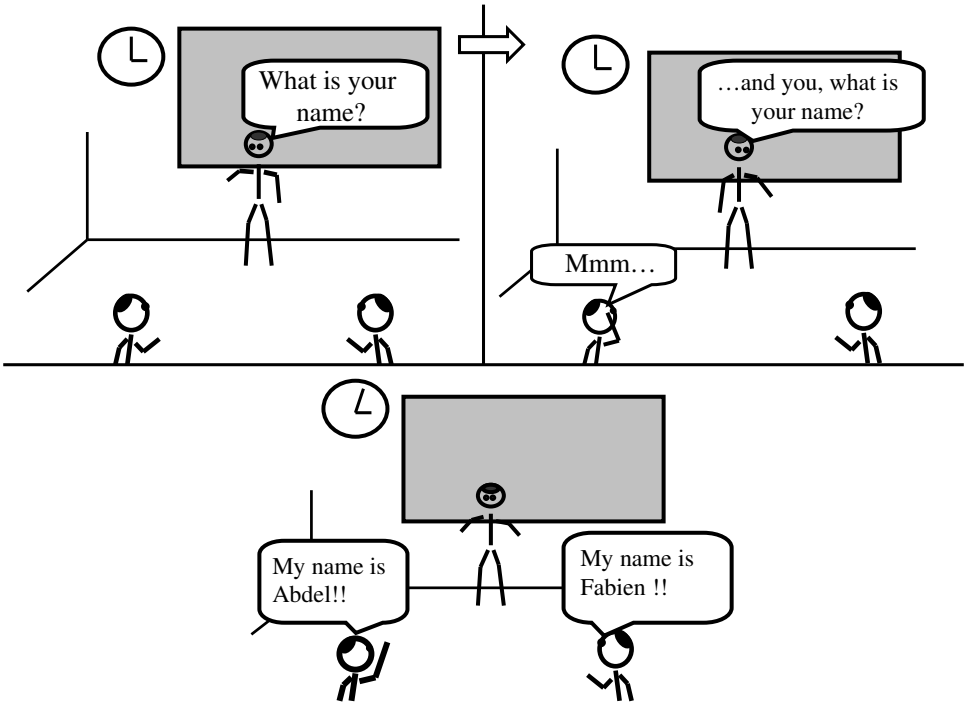


Figure 2.3: Informal example of the asynchronous communication paradigm.

The synchronous communication paradigm is suitable for controlled domains. For instance, this paradigm is used within distributed systems where the entities are deployed on a controllable communication medium such as a Local Area Network (LAN). However, Wide Area Networks (WANs), such as the *Internet*, are no longer controllable by a single authority and the software entities are autonomous. The asynchronous communication paradigm appeared then as an interesting alternative. In fact, this paradigm suggests that the entities

[1]The Object Management Group (OMG) has recognised the limits of this paradigm for large scale distributed systems and introduced the asynchronous messaging service in CORBA 3.0.

communicate asynchronously. This means that once the message has been sent, the emitting entity continues its activities. For instance, the teacher continues her/his activities as soon as the message has been sent (cf. Figure 2.3). When the teacher receives some messages from the students, she/he is free to adopt her/his own strategy in selecting the most important message to consider.

The following technologies follow this paradigm: MASs platforms such as MadKit [Gutknecht & Ferber, 2000a], JADE [Bellifemine *et al.* , 1999][Bellifemine *et al.* , 2003], Zeus [Nwana *et al.* , 1999].

Generative communication

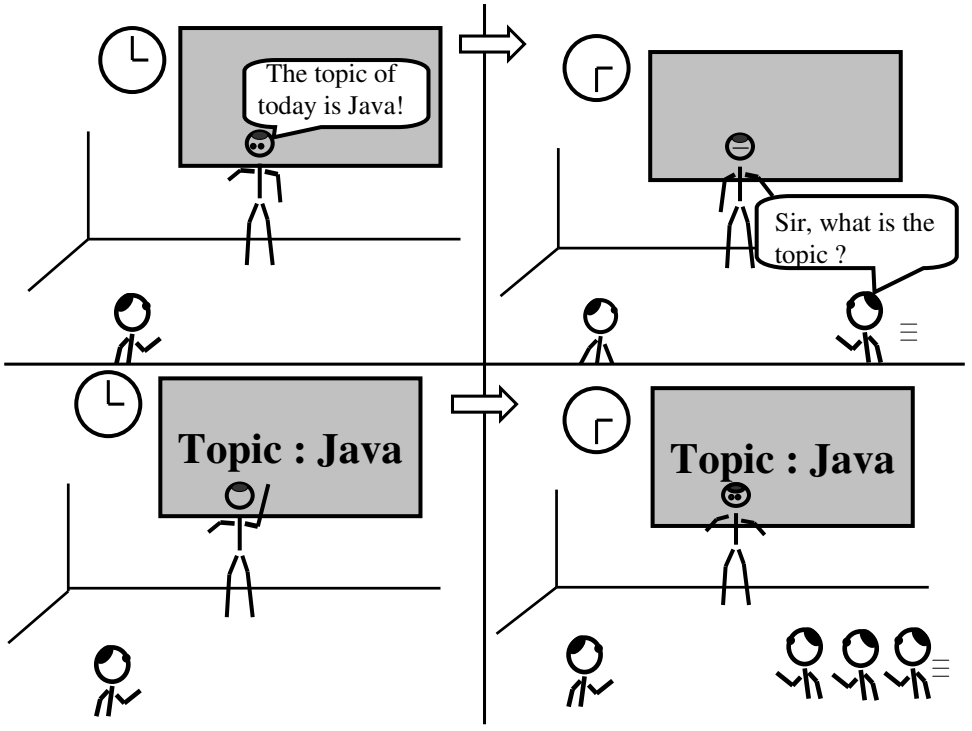


Figure 2.4: Informal example of the generative communication paradigm.

The generative communication paradigm [Gelernter, 1985] suggests to make the transmitted data persistent within the communication medium. For instance, when the teacher emits a message following the asynchronous communication paradigm, then she/he has to resend the message each time a new student arrives in the classroom. To avoid this, the teacher can write at once the message on the blackboard and all the students have access at different times to the message (cf. Figure 2.4).

As examples of technologies that follow this paradigm one may cite the applications developed around databases, blackboard architectures and tuple-spaces such as Linda [Gelernter, 1985], TuCSon [Omicini & Zambonelli, 1998], dMars [Cabri *et al.* , 2001] and TOTA [Mamei *et al.* , 2003].

Discussion

The synchronous communication paradigm is inappropriate for the EAC since the communication is instantaneous and freezes the activities of the software entities. This is inapplicable when the communications are intermittent and the software entities autonomous.

These points have been demonstrated experimentally. For instance, U. Saif and D.J Greaves in [Saif & Greaves, 2001] present why RPC is overly restrictive, inflexible and inefficient for the communication requirements of the EAC. Their conclusions are not applicable just for the RPC; they can be generalized to all the technologies that follow the synchronous communication paradigm.

The asynchronous communication paradigm is more flexible and respects the autonomy of the software entities. In fact, the behaviors are not halted and the software entities can select dynamically what message to consider depending on their priorities and state. For instance, Z. Guessoum, J.P Briot [Guessoum & Briot, 1999] and S. Cerri [Cerri, 1999] advocate that the dynamical scheduling of the arriving messages is an important feature for the autonomy of agents.

The asynchronous communication paradigm imposes that the communicating entities have to be on the same physical or logical location at exactly the same time in order to communicate. This is a constraining property especially for highly dynamical environments. Technologies using this paradigm to develop EAC applications have already noticed this difficulty and have suggested some *ad hoc* solutions such as the buffering of messages for a certain period before deleting them [Laukkanen *et al.* , 2002b] [Caire *et al.* , 2003]. These *ad hoc* solutions try to make the communication medium more persistent in order to deal with the intermittent nature of the communications. Still, these approaches are pure engineering solutions and do not have any paradigmatic or conceptual grounds.

Finally, the generative communication paradigm is the most appropriate for EAC since the communication is timely and spatially uncoupled. In fact, the communication medium maintains persistently the interactions among the software entities. Consequently, the interaction processes continue even when the software entities are temporarily disconnected from the communication medium.

2.3.2 Dealing with the Autonomy of the Software Entities

Interpretation of the Autonomy

Steels in [Steels, 1995] tries to understand what makes a software entity autonomous. This paper refers to a personal communication with Tim Smithers (1992) on the autonomy:

> "*The central idea in the concept of autonomy is identified in the etymology of the term: autos (self) and nomos (rule or law). It was first applied to the Greek city-state whose citizens made their own laws, as opposed to living according to those of an external governing power. It is useful to contrast autonomy with the concept of automatic systems. The meaning of automatic comes from the etymology of the term cybernetic, which derives from the Greek for self-steering. In other words, automatic systems are self-regulating, but they do not make the laws that their regulatory activities seek to satisfy. These are given to them, or built into them. They steer themselves along a given path, correcting and compensating for the*

> *effects of external perturbation and disturbances as they go. Autonomous systems, on the other hand are systems that develop, for themselves, the laws and strategies according to which they regulate their behavior: they are self-governing as well as self-regulating. They determine the paths they follow as well as steer along them."*

This description defines the autonomy of a system (or an entity) as its ability to produce its own laws and to follow them. By contrast, an automatic system (or entity) is given the laws and leaves them unchanged.

There are also other interpretations of the autonomy that will be presented in section [2.5.2, page 41]. Still, the actual question is not in knowing what is the 'correct' interpretation of the autonomy but what is the most appropriate and meaningful for the EAC. In fact, on some generic concepts there will be no consensus and people accept the definitions that are meaningful to their specific domain. For the EAC the most meaningful interpretation of the autonomy is the one that relates autonomy to the lack of knowledge and means to control the software entities. In other words, the more an observer lacks knowledge and means to control a software entity, the more she/he has to consider it as an autonomous entity.

However, there is still a gap between the conceptual debate on the autonomy and the engineering of autonomous software entities. In fact, within the literature of MASs several works present the *how* to build an autonomous software entity. For instance, for a cognitive agent, this agent *must* have motivations that steer its behaviors (cf. §2.5.2, page 41). These approaches assume too much knowledge on the agents' internal model. In fact, who can force all the agents of the EAC to have an explicit model of motivations in order to be considered as autonomous!

In this thesis we have adopted a different point of view: the primary question is not to have a recipe of how to build an autonomous software entity, but what are the low levels requirements to guarantee autonomy. In [Gouaïch, 2003], we argue that the *internal integrity* of a software entity is a *sine qua none* condition for the autonomy. The internal integrity is a programming constraint that avoids the direct access or modification of the software entities' internal state. So, the autonomous software entities have to be considered as bounded black-boxes that do not intersect. In fact, if the structure of the software entity is accessed or modified by any external entity, the decision process may be altered and consequently the autonomy feature is lost.

The internal integrity is also shown for collaborative learning where the learning occurs as an autonomous construction of knowledge. For instance, when an agent A "learns by being told" a piece of software from agent B, it is always A who decides to perform the change on its own state [Cerri *et al.* , 2004].

The internal integrity can be justified as a fundamental feature to implement autonomous software entities by either a top-down or bottom-up approach. We have adopted a top-down approach. In other words, we have started from some high-level considerations such as the interpretation of the autonomy and induced what are the low-level requirements to guarantee the autonomy. The bottom-up approach is illustrated by McCann [McCann, 2002] that justifies the internal integrity from a pragmatic point of view. In fact, J. McCann was interested in developing a flexible operating system for the EAC context, namely the *Go!* operating system [Law & McCann, 2000]. The 'Go!' operating system has been built entirely with the principle of the internal integrity. In fact, all the elements of the 'Go!' operating system are delimited and do not intersect. In order, to interact and exchange data these elements use a communication medium that prohibits the direct access to the elements. The

'Go!' operating system has experimentally shown interesting features such as the extensibility, adaptiveness and lightweightedness [Law & McCann, 2000].

Discussion

The autonomy of the systems and software entities is a fundamental feature for the EAC. In order to implement the autonomy, the internal integrity has been considered as a *sine qua none* condition. This means that neither paradigm nor architecture is suitable for the presented vision of EAC, if the autonomy and consequently the internal integrity of the software entities are not explicitly considered. This is an important discriminating criterion for the current approaches and paradigms to know if they are suitable or not for the EAC. For instance, the OO paradigm ignores completely the autonomy of the objects since their internal integrity is not a primary concern. In fact, an object may change the state of another object directly by setting a feature to a certain value or calling a method. So, the OO as a way to think the software system is not suitable for the EAC. This does not mean that the OO architectures cannot be used to build EAC applications. Still, this is only possible when adding solutions and concepts that have no paradigmatic grounds within the OO. The proper approach is to consider paradigms that deal explicitly with the autonomy of the software entities.

2.3.3 Interoperability of Open Systems

Concerns of the Interoperability

The communication is far from being sufficient to make heterogeneous open software systems working together and coordinating their joint activities. In fact, when the information is encoded as data and transmitted using the communication medium; the entity that receives the data has to be able to decode the information and to capture its meanings and semantics. So, while the communication media deal with the transfer of the encoding of information from one point to another; the interoperability means deal with the transfer of the semantics of information from one point to another. Furthermore, the interoperability concerns also the means that makes the open systems collaborating for the achievement of joint activities.

The first concern of the interoperability that is related to the transfer of the semantics is represented by the following works: eXtensible Markup Language (XML) [W3C, 2004a], RDF Resource Description Framework (RDF) [W3C, 2004c], OWL (Web Ontologies Language) [W3C, 2004b], Knowledge Interchange Format (KIF) [Genesereth & Fikes, 1994], DARPA Agent Markup Language (DAML) [McIlraith *et al.* , 2001], Knowledge Query and Manipulation Language (KQML) [Finin *et al.* , 1994], Semantic Language (SL) [FIPA, 2002c],FIPA Agent Communication Language (FIPA-ACL) [FIPA, 2001].

The following works represent the second concern of the interoperability which is related to the collaboration of open systems: coordination means presented in section [4.2, page 92], Web Services [W3C, 2002b], Web Service Choregraphy Interface (WSCI) [W3C, 2002a], Business Process Modelling Language (BPML) [DBMI, 2000], Web Services Flow Language (WSFL) [Leymann, 2001], XLANG [Thatte, 2001], Business Process Execution Language for Web Services (BPEL4WS) [Weerawarana & Curbera, 2002].

Discussion

The interoperability is a crucial point for the EAC. In fact, even if the current networking technologies allow the communication between open software systems, the transmitted data are useless until one provides the appropriate means to extract their semantics. The more a paradigm or architecture handles the interoperability problem, the more it is appropriate for the EAC. Furthermore, the interoperability is not limited to the exchange of information. In fact, the coordination of joint activities is also part of the interoperability to make open software systems collaborating and working together.

2.3.4 Overall Discussion

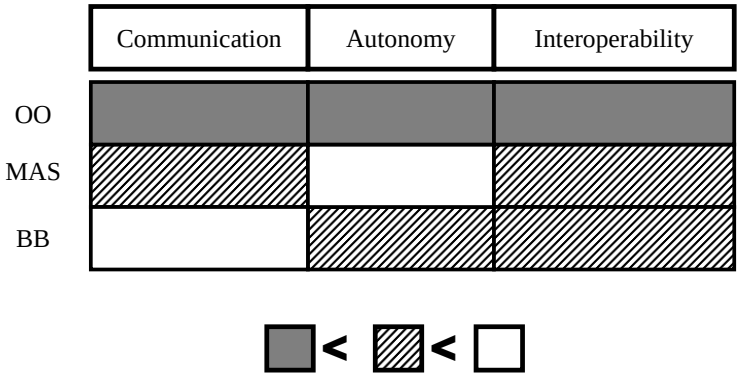


Figure 2.5: Preliminary analysis of current approaches that are suitable for the EAC.

We have tried to understand what are the constraints and hypotheses of the EAC. This helps in selecting the relevant and potential works which have to be considered as starting points for the engineering of EAC applications. From a software engineering point of view, we have set three major selection criteria related to: *(i)* the communication media, *(ii)* the autonomy of the software entities, and *(iii)* the interoperability of open software systems.

Figure 2.5 shows how the current major software engineering paradigms such as object oriented (OO), multi-agent systems (MAS), and blackboard (BB) are analyzed according to these criteria.

For the EAC the OO paradigm is the less appropriate approach to start with. In fact, the communication among the objects follows the synchronous communication paradigm. The communication paradigm is unsuitable for the EAC context (cf. §2.3.1, page 25). The autonomy of the software entities is not handled within the OO paradigm since their internal integrity is not guaranteed. In fact, the objects directly interact by modifying their states or by calling side-effect methods. During the interaction process of the objects, the encoding of the information is passed as parameters to the functions. This limits the integration of the

interoperability solutions. In fact, the type and name of the parameters are often assumed as enough to make the software systems interoperable. The classical example of this assumption is the Interface Description Language (IDL) that is used to describe the functions of CORBA objects. The IDL gives only the name of the functions and the type of their parameters; the user of the object has to guess what is the semantics of the functions having these elements.

Even if the MAS and BB approaches do not completely satisfy the requirements of the EAC; they are considered as good starting points for our work. In fact, the MAS paradigm pays attention to the interoperability of open systems and to the autonomy of the software entities. However, the communication paradigm which is used for the MAS is still not clearly defined and most of the current approaches use the asynchronous paradigm (cf. §2.3.1, page 26). For instance, the FIPA [FIPA, 1996] specifications consider the asynchronous communication paradigm.

On the other hand, the BB paradigm offers the opportunity to guarantee the autonomy of the software entities since their internal integrity is satisfied. In fact, since the interactions are always conducted through the shared data space, this prevents the software entities from directly changing their states. The communication within the BB paradigm follows the generative paradigm (cf. §2.3.1, page 27), which has been considered as the most suitable communication paradigm for the EAC. However, the first concern of the interoperability, which is related to the transfer of the semantics, is not always considered as a key issue within the BB paradigm and often a simple structure of information carrier is used. However, the second concern of the interoperability, which is related to the execution of joint activities, has been considered an important point. For instance, the works on the coordination media that follows the BB paradigm illustrate how the communication medium is used as a mean in order to make autonomous software entities coordinate their actions to execute joint activities (cf. §4.2.3, page 95).

The next two sections describe in more detail the BB and MAS paradigms and present their advantages and limits for the EAC.

2.4 The Blackboard Paradigm

First, let us make a clear distinction between the blackboard (BB) as paradigm and the blackboard as architecture. The blackboard paradigm is a way to think the software system as a set of software entities communicating through a shared persistent communication medium. Each approach that follows the blackboard paradigm defines its own way to build the persistent communication medium and how the access it. The blackboard architectures range from the simple ones to the very complex. For instance, a memory that is shared between different software entities on the same computer defines a simple blackboard architecture. In a software system based on artificial ants [Parunak, 1997], the communication is defined by the diffusion of pheromones. This system can also be considered as following the blackboard paradigm even if the use of blackboard architecture is not explicitly mentioned. In fact, the environment surrounding the ants and containing the pheromones constitutes the persistent shared communication medium. Another example is the *Internet*. In fact, when several users have access to web pages, they are using a shared and persistent communication medium which is the web.

2.4.1 Pioneer Works on the Blackboard Paradigm

The title of this section should be 'Pioneer works on the Blackboard Paradigm in Informatics'. In fact, the blackboard paradigm is probably the most ancient and natural paradigm that the humanity has followed to communicate in a timely and spatially uncoupled manner. So, its pioneer works have to be considered with the appearance of the civilizations and humanity. Still, this is too far from the problem treated by this thesis.

Within the informatics, the first works that mention the explicit use of a shared and persistent communication medium are those of the 'distributed artificial intelligence' (DAI). In contrast to the classical artificial intelligence (AI) field where the focus is made on the reasoning capabilities of a single 'intelligent' entity; the DAI focuses on the 'intelligence' of a group of entities. Hence, during the late 70s first DAI systems have emerged as an interesting alternative for complex problem resolution. Originally, this research was to shift the focus from computational control (e.g. algorithmic approach for problem resolution is about computational control) to the communication and co-ordination among distributed entities.

The HEARSAY II [Erman *et al.* , 1980] is a speech recognition system used to recognize English speeches and querying databases. It is considered as the first system that has used the BB paradigm: several software entities coordinate their actions and share knowledge through a common data space. Erman and colleagues have noticed that finding an algorithmic solution for the speech recognition problem is very hard. In fact, speech recognition involves different competences of different entities, called knowledge sources (KS), to recognize signals, syllables, words, sentences, semantics and so on. These entities need also to collaborate and to share knowledge. This was conducted through the blackboard. However, the entities' decision process was not totally autonomous and a meta-entity, called Focus, was defined in order to steer the execution of the system. Still, the global control was not known before the execution and determined as a result of the entities interactions.

The Ether system is another example of pioneer blackboard systems. The Ether has been introduced by a student of C. Hewitt (cf. §2.5.1, page 38), W.A. Kornfeld in [Kornfeld, 1979]. Kornfeld has noticed that the distribution is usually motivated by efficiency. Hence, the programs are executed concurrently to use more resources and to terminate more quickly. He suggested the use of distribution for more 'qualitative' criteria rather than 'quantitative' criteria. To demonstrate his approach, he developed a theorem-proving application: two different algorithms proving a mathematical property were implemented; when these algorithms where collaborating and sharing their partial results, the solution was found more quickly. Kornfeld's architecture is composed of two orthogonal components: the *Ether*, representing the persistent communication medium; and the *Sprites*, representing the software entities. The sprites communicate by sending and retrieving messages from the Ether. Furthermore, a sprite may stop its execution waiting for a particular event to arrive in the Ether. In contrast to its precedents architectures, the Ether has made the properties of the blackboard explicit. So, the blackboard is no longer a flat set of data without any structure but owns some particular properties that affect the dynamics of the software system. For instance, the Ether defines the property of the *commutability*: the sprite creation and appearance of a message in the Ether are commutative operations; and the *monotoniaty*: messages cannot disappear from the Ether.

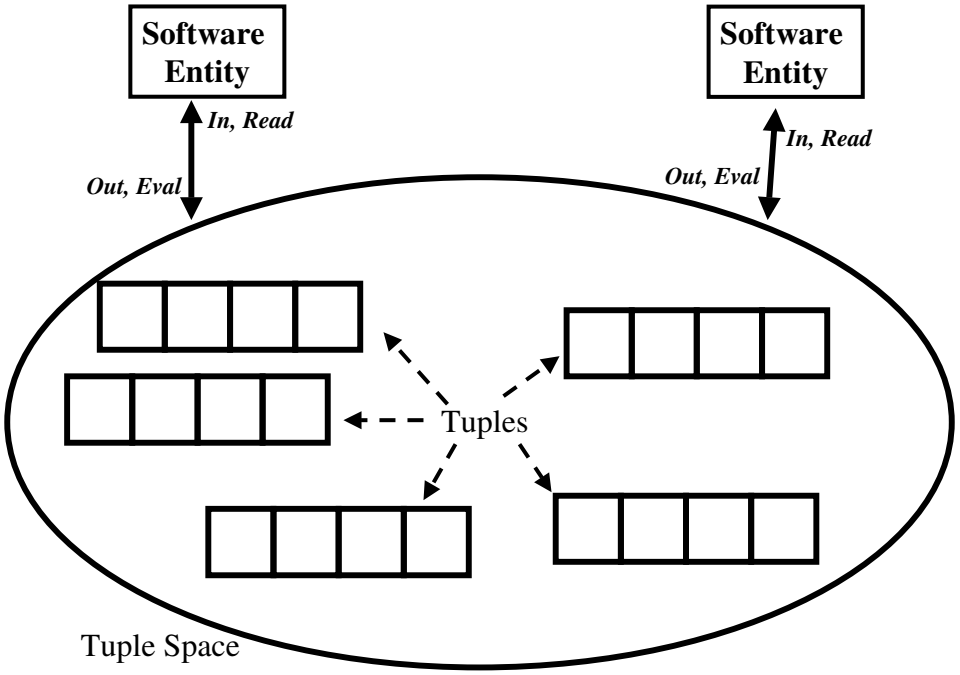


Figure 2.6: The main elements of the tuple space architecture: tuples and the software entities that communicate by reading and writing the tuples.

2.4.2 The Tuple Spaces

The *Tuple Spaces* (TSs) have been introduced by researchers at the Yale University where Linda [Gelernter, 1985] –the first tuple space-based system– has been developed. A TS system is composed by the following elements:

- **Tuple**: A tuple is basically a list of typed fields. Fields may be *actual* when they hold a value or *formal* when no value is contained.
- **Tuple Space**: A tuple space is an abstract storage location where tuples are deposited and retrieved by the software entities that called processes.
- **Processes**: The processes store and retrieve the tuples using the following elementary communication primitives:
  - **out**: inserts a tuple into a tuple space which becomes visible to all the processes with access to that tuple space.
  - **in**: extracts a tuple from a tuple space, with its argument acting as the template, *anti-tuple*, against which to match. When all the corresponding fields of a tuple match the template the tuple is withdrawn from the tuple space.
  - **read**: this is equivalent to 'in' except that a matched tuple is not withdrawn from the tuple space and remains visible to the other processes.
  - **eval**: this is similar to 'out', except it creates an independent process yielding the tuple, which is inserted in the tuple space.

2.4.3 Tuple Space Architectures

There are several architectures that follow the principles of the Linda architecture. The goal here is not to present an exhaustive list of these architectures. Only the architectures that are relevant and present some interesting features are presented. We start by presenting the original TS architecture Linda which has inspired the following models. The TuCSoN model is then presented as it introduces a new functionality to the TSs by making them programmable. The TOTA architecture is presented as it explicitly targets the engineering of software systems for the EAC context. Finally, the JavaSpaces architecture is presented to illustrate the strategic shift adopted by important software companies such as Sun Microsystems™ to benefit from the interesting features of the generative communication paradigm.

Linda

Linda [Gelernter, 1985][Gelernter *et al.* , 1985] is the first TS system developed as an alternative to the RPC for building distributed software systems. Linda consists of a reduced set of communication primitives to access and modify a shared memory containing the tuples, namely the tuple space. Even if the tuple space is conceptually centralized, it can be implemented in a decentralized manner and several works have addressed the distribution of the tuple spaces such as: [Alvez & Yovine, 1991][Karp, 1993][Douglas *et al.* , 1995][Hawick *et al.* , 2002]. Hence, several Linda sub-systems are composed in order to build a coherent distributed tuple space that guarantees the operational semantics of the communication primitives. Linda does not

make hypotheses on the internal models of the software entities. The software entities have only to follow the syntax of the tuples, anti-tuples and the communication primitives.

TuCSoN

TuCSoN is a Linda-like TS proposed by A. Omicini and F. Zambonelli [Omicini & Zambonelli, 1998]. The TuCSoN model has been designed for coordination purposes and not only for communication. For instance, the authors use the term of coordination medium rather than the communication medium to highlight the objectives of the TuCSoN model and architecture.

Despite the fact that TuCSoN is a Linda-like TS, there are significant differences between these architectures. In fact, just as MARS [Cabri *et al.* , 2001] (another Linda-like TS), the TuCSon model introduces the idea of a programmable coordination medium: while Linda is composed of tuple spaces that define statically the interaction schemes between the tuples and the anti-tuples, within TuCSoN the interaction schemes of the *tuple centres* [Omicini & Denti, 2001a] are programmable and can be dynamically changed using the ReSpecT [Omicini & Denti, 2001b] logical programming language. In other words, each tuple centres may define its own rules of reaction to the elementary communication primitives. This is a significant modification of the initial approach proposed by Linda. In fact, the communication medium is now an active entity that affects the dynamics of the global software system. Consequently, the communication medium has to be considered explicitly as a first class entity[2] during the engineering phases of the software system.

TOTA

Tuples On The Air (TOTA) [Mamei *et al.* , 2003] is a TS based architecture that offers a middleware to develop pervasive applications. TOTA has no notion of a centralized or shared tuple spaces. The tuples are injected into the network of TOTA nodes from any node and can propagate and diffuse according to tuple-specific propagation patterns. The software entities can exploit a programming interface to define and inject new tuples into the TOTA network and to locally sense both tuples and events associated with changes in the tuples' distributed structures such as the arrival and departure of tuples.

JavaSpaces™

JavaSpaces [Sun, 2000] service specification provides a distributed and persistence object exchange facility for the Java-based systems. JavaSpaces technology uses Java RMI for the interaction between the distributed objects and Jini for implementing distributed services. The JavaSpaces service holds entries that describe a group of objects. An entry is copied into a JavaSpaces service and used for future lookup requests. This mechanism is similar to Linda [Menezes *et al.* , 2001]. Hence, a distributed software system is viewed as a set of distributed objects that exchange persistent objects, or messages, through a JavaSpaces service using write and lookup operations.

Sun Microsystems™ has already developed the Jini™ and the Java RMI™ technologies for distributed computing. However, these technologies were not adapted for large-scale systems

[2]Recently, the MAS community has recognized that this is a key issue. The E4MAS workshop [E4MAS, 2004] studies how to consider the communication medium as a first class entity within the MASs.

that operate on unreliable communication media such as the *Internet* or wireless networks. As for the CORBA, the interaction model among the software entities may explain this failure. To avoid this problem, the JavaSpaces adopts the generative communication paradigm.

2.4.4 Advantages and Limitations of the BB Paradigm and TS Architectures for the EAC

Obviously, the main advantage of the BB paradigm and TS architectures is the generative communication. In fact, the constraints of the communication medium are handled gracefully and flexibly by uncoupling the communication and interaction between the software entities on space and time. So, the intermittent communications, disconnections and momentary failure of some components do not disturb drastically the entire software system.

Moreover, the internal integrity is also respected by the BB paradigm. In fact, the software entities do not have a direct access to the software structures of other entities, but interact by the mediation of the communication medium. Furthermore, as shown by the TuCSoN model, the communication medium is not passive but may own its proper reaction rules. However, there are still some disadvantages and weaknesses to be considered in order to build a framework for the EAC.

The first point concerns the soundness of the communication primitives' semantics and our hypotheses of work. For instance, the semantics of the 'in' primitive is inconsistent with the autonomy. In fact, the 'in' primitive introduces the problem of shaded actions. The shaded action problem appears when the software entity's actions or influences, which are reified as tuples, are always retrieved by another software entity. So, this entity shades completely the actions of the emitting entity and from an external point of view the shaded entity does not exist in the software system since its actions or influences are never perceived. Introducing some reaction rules in the tuple centres can solve this problem. Still, the real problem is that the tuple space is a simple flat set without any structure. Consequently, other entities might change a state that another entity is responsible for without any obvious tracking of which entity changed the data. This means that the autonomous definition of the entities is lost because you can not claim an ownership of that data. The lack of structure also reduces the scalability of TS based applications as the whole data space has to be synchronized with the remote data spaces for in/out operations.

The second point concerns the tuples themselves. The tuples are the information carrier that reifies the knowledge and actions of the software entities. Still, the tuples are unstructured. For instance, does an aggregation of tuples constitutes a tuple? Furthermore, the anti-tuple is introduced as an *ad hoc* mean to lookup tuples. But, the anti-tuples and tuples are not unified in a single structure. This prevents from having a certain level of introspection within the TS. For instance, it is not possible to observe an observer. In other words, one cannot make a lookup request using the 'in' primitive to lookup anti-tuples.

The third point is a consequence of the second point and is caused by the artificial introduction of anti-tuples. This point concerns how the software entity interacts with its surroundings. Generally, we can identify two kinds of interaction methods: the active interaction and the passive interaction. Within the active interaction, the software entity requests explicitly to sense its surroundings. So, the sensing means of this entity are only defined between the time when the entity has requested to sense its surroundings and the time when the calculation of its interaction is terminated. In the passive interaction, the software entity defines its sensing means in a persistent manner. These sensing means continuously sense the surroundings and

store information even if the software entity is performing another activity. The TS adopts an active interaction method. In fact, the software entity cannot store its anti-tuples in the TS and retrieves the results whenever it wants. Again, it is clearly a structural problem. In fact, except the time when the software entities are actively requesting to sense their surroundings, the TS does not define any location where to store the sensing means and the perceptions of a software entity.

If the advantages of using the tuple space design to form part of the EAC solution then resolving the above limitations of the tuple space design adequately are required.

2.5 The Multi-Agent Systems Paradigm

The MAS field is currently one of the most prolific field of DAI and software engineering in terms of scientific publications and research work. Still, contrary to the OO architectures, MAS architectures until now have not gained the necessary credibility to be considered as the leading paradigm for the distributed software systems engineering. In fact, few companies trust the MAS approach enough to build 'serious' applications that are not just showcases. This can be explained by the conservatism of the technological market. In fact, until a new technology actually proves that it can do what the known technologies cannot, the major actors of software engineering do not give it the required credibility. For the classical software environments, that are mostly controlled environments, the fundamental features of the agents are not required and the MAS approach cannot show clearly what are its advantages. In fact, in these controlled environments, the MAS approach is viewed more as an amelioration or evolution of software engineering and not as a revolution. However, the constraints of EAC give the MAS paradigm a chance to demonstrate the promised revolution on software engineering and informatics. This point of view is shared by F. Zambonelli and V. Parunak [Zambonelli & Parunak, 2002] that see in the properties of EAC the signs of the revolution in computer science and software engineering and a chance for the MAS paradigm to prove its expressiveness and soundness.

In order to introduce the MAS paradigm, the following section presents some pioneer works; then a more contemporary vision of the MASs is given in section [2.5.2, page 40]. In this section, we discuss two major trends of MASs namely, the agent centered and the system centered approaches. Section [2.5.4, page 45] presents briefly the FIPA standard and its proposition to address the EAC. Finally, section [2.5.5, page 47] enumerates the advantages and weaknesses of the MAS approach to address the EAC context.

2.5.1 Pioneer Works of MASs

The *actor* model has been initially introduced by C.E. Hewitt in [Hewitt, 1976a] and was joined later by G. Agha [Agha & Hewitt, 1985][Agha, 1986]. The actors are active software entities that *(i)* communicate by sending asynchronous messages; *(ii)* create other actors; *(iii)* and change their behaviors by becoming other actors. The dynamics of the software system and the actors is not fully determined before the execution of the system; it is determined by the relationships among the events generated by the actors. Consequently, as stated by Hewitt in his paper:

> "*the control structure emerges as a pattern of passing messages among objects being modeled.*"

Notice that contrary to the HEARSAY system, which targeted a particular speech recognition problem, the actor model is a generic pattern for the distributed resolution of problems. The communication paradigm also differs between the HEARSAY and the actor model. In fact, while HEARSAY used a blackboard architecture as a communication medium, the actors communicate using asynchronous messages. The actor model can be considered as the ancestor of MAS and the differences between the actors and agents are not clear at least at the conceptual model. In fact, Hewitt and Agha motivations for the Actor model are the same as those presented by the MAS community. Probably, the autonomy feature makes a clear distinction between the actors and the agents. Indeed, despite the fact that the theoretical descriptions of the actors emphasize on the autonomy, this feature is not reified explicitly at the implementation level. Consequently, when sending a message to another actor, the emitting actor remotely controls its behavior.

D. Lenat has developed the PUP6 system where specialists, called *beings*, work together to solve problems [Lenat, 1975]. The questions that are not answered by beings are forwarded to a human operator. The beings differ from the HEARSAY knowledge sources: they do not use blackboard architecture and are continuously modifying their internal structure and behaviors.

These pioneer works of DAI have inspired the works conducted during the 1980s to consider a software system as *an artificial organization*. For instance, M. Fox in his paper [Fox, 1981] gives an organizational view of software systems by considering the HEARSAY and actors as examples [Erman *et al.* , 1980]. Fox has noticed that the human organizations and software systems share the same property on the entities, namely the *bounded rationality*. According to H. Simon [Simon, 1957], a rationally bounded entity is an entity which reasoning capabilities are limited. For instance, humans are rationally bounded and their intellectual faculties are quickly flooded under complex information and tasks. The limitation of the human mind to absorb complex information creates the need for grouping in organizations to solve complex problems. The software entities are also rationally bounded. In fact, each software entity can process only a limited amount of information. Consequently, the software entities have to coordinate their efforts in order to solve complex problems. As an example of a coordination pattern, R.G. Smith has proposed an economical metaphor for dynamical task allocation: *the contract net protocol*[Smith, 1980]. This protocol assumes a topology of independent nodes; each node has its own resources and communicates with other nodes using a communication medium. The problem to be resolved is decomposed in several tasks; the tasks are then allocated to the appropriate node using a negotiation mechanism between a *coordinator* and the *executors*. This distributed negotiation process is characterized by the following points:

1. the control is locally achieved and does not imply centralization;
2. the information flows are bi-directionally exchanged between the coordinator and executors;
3. each entity evaluates information from a local point of view;
4. the final agreement is achieved by mutual selection, which means that the coordinator and the executor agree on the task to be executed.

The Distributed Vehicle Monitoring Testbed (DVMT) [Lesser & Corkill, 1983] is also an important example of DAI application developed by V. Lesser and D. Corkill at the Massachusetts University. The DVTM simulates a control network of vehicles. The nodes of DVMT

are problem solvers that analyze acoustically sensed data to identify, locate, and track patterns of vehicles moving through a two dimensional space. Each node is associated with a physical sensor sensing the signals generated by mobile vehicles. These signals are generally altered by the surrounding environment, which induces local errors on the perceptions. The DVMT application has demonstrated that the errors of perception were minimized when the nodes work together. Hence, by sharing their local knowledge, the nodes were able to locally correct errors and the global system gives more accurate results.

The Multi-Agent Computing Environment (MACE) platform [Gasser *et al.* , 1987] was developed by Gasser and colleagues. The MACE platform was designed as a simulation platform to test DAI applications and provides:

- a language for describing agents;
- a mapping of agents into processors;
- a communication medium that handles inter-agents communications.

The MACE active entities were called *agents* instead of actors or active objects since they exhibit additional features such as the exploitation of an explicit organizational structure; model of other agents; explicit representation of their goals, tasks, roles and capabilities.

For more readings, a summary of DAI works of this period (1980s) have been presented by Bond and Gasser in [Bond & Gasser, 1988].

2.5.2 Current Vision(s) of MASs

The MAS paradigm seems to be an interesting approach for the design and development of open and distributed software systems [Ferber, 1995] [Jennings & Wooldridge, 1997]. According to Demazeau [Demazeau, 1995] a MAS can be described with four main concepts which are: Agent, Environment, Interaction and Organization (AEIO). The agents are the autonomous goal-directed entities that populate MAS. They achieve their design goals by interacting with other agents and by using the resources available in their environment. The interaction is defined by Ferber [Ferber, 1995][Ferber, 1999] as the set of mechanisms used by the agents either to share knowledge or to coordinate joint activities. The organization can be defined as the set of mechanisms that reduces the entropy of the system and makes it ordered behaving as a coherent whole.

The most consensual definition of the agents is given by Wooldridge and Jennings [Wooldridge & Jennings, 1995]:

> "*Perhaps the most general way in which the term agent is used is to denote a hardware or (more usually) software-based computer system that enjoys the following properties:*
>
> - *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state [Castelfranchi, 1995];*
> - *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language [Genesereth & Ketchpel, 1994];*

- *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*
- *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative."*

Since there is no formal or standardized definition, the agents are given just some fundamental features. Some of these features are also shared by other entities such as the actors or objects. For instance, the software object can interact also with other objects and may react to an external stimulus by returning a certain result. Still, we state that the most important feature that distinguishes the agents from other known entities in software engineering is the autonomy feature.

The autonomy has several interpretations within the literature of MASs. The following section explores these interpretations and gives our point of view on the autonomy and why this feature makes the MAS suitable for the EAC.

The Autonomy and the MAS Paradigm

Two main interpretations have been selected for the purpose of this thesis: the autonomy as self-governance and as independence.

Autonomy as Self-Governance Castelfranchi [Castelfranchi, 1995] defines an autonomous agent as a software program able to excise a choice that is relevant in the context of goals-directed behavior. Luck and D'iverno in several works [Luck & d'Inverno, 1995][Luck & D'Inverno, 2001][d'Inverno & Luck, 1996] share also the same interpretation of autonomy and specify formally using the *Z* notations [Spivey, 1987] what is an autonomous agent and what are the features that distinguish it from a software object. From Luck and D'iverno perspective, an object is a simple software entity with some features and actions. An autonomous agent is defined as an object with a set of motivations that steer the agent in selecting what are the goals to be achieved. Guessoum and Briot [Guessoum & Briot, 1999] address also this issue and enrich a pure object architecture, Actalk, with some mechanisms such as controlling message reception and selecting what behavior to adopt in order to build an agent platform named DIMA.

Autonomy as Independence The Social Dependence Network (SDN) has been introduced by Sichman *et al.* in [Sichman *et al.* , 1994]. The SDN gives some information about the social context of an agent. This social context is then used by the agents to achieve their individual goals. More precisely, the agents have external descriptions representing models about their neighbors. An external description of an agent is composed by its goals, actions, resources and plans. The goals represent the state of affair that the agent wants to reach; actions are operations that an agent is able to perform; resources represent the means under the control of the agent; and finally plans are sequences of actions and resources. Notice that the authors adopt the hypothesis of external description compatibility implying that all the agents have the same models about the others. This is unachievable in an open context where the set of agents is not static and the communication is asynchronous. The SDN framework distinguishes three forms of autonomy. An agent is a-autonomous for a given goal according to a set of plans,

if there is a plan in this set that achieves the goal, and every action in each plan belongs to its capabilities. An agent is considered as r-autonomous for a given goal according to a set of plans, if there is a plan in this set that achieves the goal and every resource in each plan belongs to its resources. Finally, an agent is s-autonomous when it is both a-autonomous and r-autonomous. According to this definition, an agent is autonomous for a particular goal if it does not depend for resources or actions on another agent.

Discussion Sichman *et al.* relate the autonomy of the agent on its social context. On the other hand, Castelfranchi and colleagues relate the autonomy of an agent only to its own behaviors. The latter definition seems to be more appropriate to catch what an autonomous software agent means for the EAC context. In fact, as mentioned by [Luck & d'Inverno, 1995], a pocket calculator that has the resources and actions to calculate some arithmetical operations is seen as an autonomous agent according to the SDN definition. But the outputs of the calculator are completely known given some inputs.

Our interpretation of the autonomy is more related on the decision process of the agents. Still, it differs from the interpretation of Castelfranchi and colleagues. In fact, while Castelfranchi and colleagues give an absolute definition, it is more interesting to give a relative definition of the autonomy. In other words, an entity can be autonomous or not depending on the observer of this entity.

If the external observer of an entity has enough knowledge and means to control its behaviors, this entity is not autonomous for this observer. By contrast, if the external observer lacks knowledge and means to control the behaviors of the entity, the later is autonomous since its outputs may change in response to the same stimulus. So, no matter if the behaviors of the software entity are actually changing over time; it is the lack of knowledge and control means that makes the software entity behaving as autonomous.

As a consequence of this interpretation of the autonomy, the software agents that have been developed locally and that are under the total control of their designers do not exhibit the same level of autonomy as foreign agents that have been developed by other authorities. For the EAC context, the openness hypothesis implies that the software agents have to be considered as autonomous entities since they have been developed by different authorities.

Objective Criterion to Implement Autonomous Agents

Within the MASs there is still a gap between the debate at the conceptual level on the autonomy and the implementation of autonomous agents. In fact, most of the definitions and interpretations of the autonomy give a subjective point of view without some objective criteria to implement the autonomy. So, as Weiss and colleagues [Weiss *et al.* , 2003] have pointed out, objective implementation criteria are necessary to define and guarantee the autonomy of a software agent. We propose the *internal integrity* as an objective criterion to guarantee the autonomy of the agents [Gouaïch, 2003].

The internal integrity is a programming constraint that considers an autonomous agent as a bounded system which internal dynamics and structure are neither controllable nor observable directly by an external entity. In fact, if the agent's software structure is accessed or modified by another entity, the decision process and behaviors may be altered. Since the decision process of an agent has to be entirely determined only by its own perception and behaviors, the internal integrity becomes a *sine qua none* condition to implement autonomous agents.

Ensuring the Internal Integrity The internal integrity criterion also raises some issues with respect to the implementation of MASs: on one hand, the internal integrity has to be taken into account to guarantee the autonomy; on the other hand, the autonomous agents are interacting entities that need to act and modify the perceptions of other agents. Since these perceptions are included within the boundaries of the agents, this contradicts the internal integrity statement. In other words, the problem is to enable the interaction between autonomous agents which boundaries do not intersect.

To avoid this paradox, the MAS have to identify a non-agent entity that manages and carries out the interactions. This entity has been named the agents' deployment environment (DE). Chapter 3 presents our proposition of a DE named MIC*.

2.5.3 Agent Centered MAS and System Centered MAS

The MASs works can be divided into two main categories depending on the point of view from which the MAS is studied. In fact, the MAS can be analyzed and studied from the agent point of view. This represents the agent centred MAS (ACMAS) approach. On the other hand, the MAS can be analyzed and studied as a whole at the systemic level. This represents the systemic centered MAS (SCMAS). This decomposition has been proposed by Ferber in [Ferber *et al.* , 2003]. Ferber's uses the term organization centered MAS (OCMAS) instead of SCMAS. The term system has been preferred to the term organization as it is more general to represent the study of the dynamics of the overall MAS.

By using Demazeau's decomposition one can briefly gives an order of the important components in each approach. For the ACMAS approach the central component is the agent. The interaction, organization and environment are defined from the agent's perspective. In the SCMAS the agents lose the dominant place and the organization, environment, interaction become the central concepts. The SCMAS try to study the MAS by assuming minimal knowledge and hypothesis on the agents' internal models.

ACMAS

The illustrative example of this approach is the Belief, Desire, Intention (BDI) model of the agents' mental state proposed by A. Rao and M. Georgeff [Rao & Georgeff, 1995]. In this model, the agents have three logical sets holding some cognitive predicates representing believes, desires and intentions. These three sets represent the mentalistic model of the agent that steers its behavior.

In ACMAS The interaction is defined from the agent's mental model perspective. For instance the KQML [Finin *et al.* , 1994], Arcol [Breiter & Sadek, 1996], FIPA ACL [FIPA, 2001] are agent communication languages (ACLs) which semantics assumes that the interacting agents follow a specific mental model such as the BDI model.

The organizational aspects of the MAS are also defined from the agent's mental model perspective. The cooperative problem solving (CPS) models developed by the ACMAS approach describes the cooperation and the joint actions of a group of agents by assuming the complete access and knowledge of the agents' mental states. For instance, Cohen and Levesque have developed the joint intention model that specifies how a group of agents act together by sharing certain mental beliefs about the cooperative actions [Levesque *et al.* , 1990]. The Wooldridge-Jennings CPS model [Wooldridge & Jennings, 1994] is another illustrative exam-

ple of this approach. This model is based on the joint intention theory and specifies the process of the CPS by four-stages:

1. recognition: during this phase the agents recognize the potential for teamwork with respect to a common goal. This implies that the agents share their goals and use the same ontology to describe them.
2. team formation: in this phase the group of agents that share the same goals attempt [Wooldridge & Jennings, 1994] to build a group of agents.
3. plan formation: during this phase the agents forming the group negotiate the plan of actions to be conducted in order to achieve the team goals.
4. team action: during this phase the agents execute actions of the agreed plan. The agents are also required to adapt some social conventions described in [Jennings, 1993] to monitor their teamwork progress.

The ACMAS approach is suitable for controlled environments since it assumes a lot of knowledge and hypotheses on the agents. For instance, H. Chen and T. Finin [Chen & Finin, 2002] show how the constrains and hypotheses of the EAC make the ACMAS approaches insufficient to guarantee the teamwork and coordination. In fact, the limitation of perceptions, planning limitations and device mobility challenge completely the applicability of the joint intention theory.

More generally, the failure of the ACMAS approach to address the EAC context can be explained by the contradiction that exists within this approach. In fact, you cannot state that the agents are autonomous on one hand and assumes how they function internally on the other hand. In fact, as stated previously, it is the lack of knowledge and control means that make the agents behaving as autonomous. So, the less an approach assumes how the agents function internally, the more it is general and suitable to address open and uncontrollable environments such as the EAC.

SCMAS

Within the SCMAS the focus is not the agent but the entire system. The organization and the interaction are studied and described independently from the agent's mental states. Hence, the organization centered MAS (OCMAS) [Ferber *et al.* , 2003] tries to identify how to organize the MAS system using organizational models that are independent from the mental states of the agents. For instance, the AGR organizational model [Ferber & Gutknecht, 1998] proposed by Ferber and Gutknecht specifies the organizational structure of a MAS without assuming how the agents are built internally. In fact, the agents are abstracted as roles [Kendall, 1999] that describe what function the agent is expected to perform in the system and not how this function is performed. So, the concept of role allows to abstract the agents' function and to talk about them without manipulating them explicitly. As consequence, within the same MAS heterogeneous agents built with different models can collaborate as soon as they share the same interaction language.

Concerning the interaction, the SCMAS approach offers ACLs which semantics does not depend on the mental model of the agents. This is illustrated for instance by the work of M. Singh. We can mention the following statement taken from Singh's paper [Singh, 1998]:

> *"I am not convinced, however, that the existing work on ACLs, especially on the semantics, is heading in the right direction. It appears to be repeating the past mistake of emphasizing mental agency –the supposition that agents should be understood primarily in terms of mental concepts, such as beliefs and intentions. It is impossible to make such a semantics work for agents that must be autonomous and heterogeneous: This approach supposes, in essence, that agents can read each others minds. This supposition has never held for people, and for the same reason, it will not hold for agents."*

In [Singh, 1998] M. Singh discusses the limitation of the ACMAS approach to define the interaction between the agents in term of ACLs that are dependent on the mental states of the autonomous agents and suggests to use the *social context* as the basis for the semantics of ACLs.

P. Charlton and E. Mamdani [Charlton & Mamdani, 1999] show the limitations of the mental agency based ACLs and how the *social context* and *social policies* are used within the ACLs in order to specify the social commitments of the autonomous agents which ease the coordination of the joint activities.

Still, in our knowledge there is not a general-purpose SCMAS ACL that has gained acceptance within the MASs community. In practice, KQML and FIPA-ACL are used within the SCMAS, but the agents never implement their semantics. So, these interaction languages are used more for their expressiveness to have human readable messages rather than messages that affect the BDI mental states of the software agents.

2.5.4 The FIPA Proposition to Address the EAC

The Foundation for Intelligent Physical Agents (FIPA, www.fipa.org) is a standardization organization promoting development and specification of agent technologies. The FIPA specifications aim to make agent systems interoperable by defining a standardized agent interaction language, namely the FIPA-ACL [FIPA, 2001], and an abstract architecture of the Agent Platform (AP).

The FIPA AP is required to implement three capabilities:

- **The Agent Management System (AMS)**: The AMS is responsible for agent creation and deletion, maintenance of a directory service where every agent deployed on the AP has to be registered (white pages), and the agent life-cycle management.
- **The Agent Communication Channel (ACC)**: The ACC represents the communication medium used to exchange FIPA ACL messages between the agents. The routing of FIPA ACL messages is based on the globally unique Agent Identifier (AID) and the agent address. Hence, the ACC may deliver messages either to an agent deployed locally within the same AP or contact the ACC of a remote AP that contains the agent.
- **The Directory Facilitator (DF)**: The DF maintains a directory service where agents deployed on the AP register the description of their services. Several DFs can be federated and the search of services is extended to all the federated DFs.

Currently there are several APs that implement the FIPA specifications: JADE [Bellifemine *et al.* , 2003], JADE-LEAP [Berger *et al.* , 2003], FIPA-OS (fipa-os.sourceforge.net), and MicroFIPA-OS [Laukkanen *et al.* , 2002a].

Although the original FIPA specifications have addressed some aspects of the EAC context such as the use of bit-efficient encoding of ACL messages to preserve the network bandwidth [FIPA, 2002a] and the buffering of the messages to handle the intermittent nature of the communication medium [FIPA, 2002b], there were several difficulties when adapting directly the FIPA specifications for the EAC context.

In 2002, the FIPA has created the Ad Hoc Technical Committee[3] (TC Ad Hoc). The objective of the TC Ad Hoc is either to modify existing specifications or create new specifications to ensure interoperability between FIPA-compliant APs in mobile *ad hoc* networks (MANETs) that we have identified as the EAC context.

The JADE-LEAP [Berger *et al.* , 2003] project has already experimented the distribution of a FIPA AP over several small devices such a phones, PDAs and computers. The idea was to define several light containers running on the limited devices and a main container holding the FIPA mandatory services such as the AMS and the DF. The main container runs on a powerful device such as a personal computer (PC) and communicates with the other containers using wireless network protocol such as GSM or GPRS. Still, the sub-containers do not constitute a FIPA AP and thus are not interoperable with other FIPA-compliant platforms when the communication medium is not available with the main container. In contrast to the LEAP-JADE approach, the goal of the TC Ad Hoc is to specify an entire self-contained FIPA compliant AP that runs in each device [Berger, 2002]. In fact, the JADE-LEAP approach is limited since the communication link between the main container and the sub-containers are not always available in the EAC context.

Probably the experiment of the FIPA standard with the EAC is a concrete example of how the EAC context challenges the engineering assumptions. For instance, some minor contradictions of the FIPA standard have been highlighted by the EAC constraints and have been revised. In fact, the main goal of the FIPA is to make MASs interoperable; and for this important results have been brought such as the FIPA-ACL, FIPA interaction protocols set, FIPA semantic language (FIPA SL), and the integration of the ontologies in the software engineering process. Still, the interoperability can be performed without specifying how to manage the states of the agents and services. So, the contributions of the DF and AMS for the interoperability are not clear.

For instance, the service discovery is a function that can be implemented in a customized manner for each specific MAS. Furthermore, the service management model of the FIPA using the DF and the federation of several DFs cannot be implemented efficiently in the EAC context [Berger, 2002]. For this reason, in March, 2004 the TC Ad Hoc has changed the FIPA agent management specification [FIPA, 2004] to consider the DF as an optional component.

The other problem revealed by the TC Ad Hoc is related on the semantics of the FIPA ACL for group communication (multicast/broadcast). In fact, the semantics of the FIPA ACL is defined between two identifiable agents. So, the communication between an agent and a group of unknown agents has no semantical grounds for the FIPA ACL. The TC Ad Hoc has offered an *ad hoc* solution by making a specific reserved name to broadcast messages to the unknown agents. More generally, this problem can be related on the routing mechanism used by the FIPA ACL and that is based on the AIDs. In fact, there are several methods for routing the messages in the MASs that are independent from the exact identities of the agents. For instance, MadKit [Gutknecht & Ferber, 2000a] uses an organizational routing mechanism. The messages are delivered by roles and not by identities.

[3]The web page of this TC is: www.fipa.org/activities/ad_hoc.html

2.5.5 Advantages and Limitations of the MAS Paradigm to Address the AEC

The major advantages of the agent-oriented software engineering (AOSE) are: the autonomy of the software entities; the expressiveness of the interaction means to ease the interoperability and coordination; and the natural modeling of the software systems as artificial societies of agents.

Among the limitations of the AOSE for the EAC one can mention the following points:

- The lack of objective criterion to implement and guarantee the autonomy of the software agents: MAS works often do not relate the theoretical and conceptual results and the implementation of the MASs. We have proposed a specific interpretation of the autonomy for the EAC context and how to implement it by guaranteeing the structural integrity of the agents.
- The specification of the communication medium paradigm : the MAS paradigm tells that the agents communicate and interact but without specifying which communication model to use. Hence, the dominant approach considers the asynchronous paradigm as a model of communication within MASs. To handle the constraints of the communication medium within the EAC context, we suggest the use of the generative paradigm as the model of communication.
- The deployment environment (DE) is not considered as first-class entity within MAS: few works have considered the general study of the DE that holds the autonomous agents. For this reason, in 2004 the Environments For Multi-agent Systems (E4MAS) workshop has been proposed for the Autonomous Agents and Multi-Agents Systems[4] (AAMAS) conference under the initiative of D. Weyns, V.D. Parunak and F. Michel. The workshop has highlighted the importance of the DE to be considered as a first-class entity in the MAS. In fact, the DE is not a simple passive container holding the agents; it defines their interactions and thus affects the dynamics of the MAS. For this reason, we have developed an algebraic model of the DE that is presented in Chapter 3.

2.6 Conclusion

The BB and MAS has been considered as relevant starting points for the engineering of software systems in the EAC context. Still, only the interesting features of each paradigm have been selected. For instance, the generative communication paradigm has been selected from the BB paradigm, while the results on the software entities autonomy and the use of high-level interaction means have been selected from the MAS paradigm.

Figure 2.7 presents how the rest of the thesis is organized:

- **Chapter 3**: This chapter presents the MIC* model of the DE where the autonomous agents are deployed. The MIC* DE guarantees the internal integrity of the autonomous agents while defining their interactions. MIC* offers the opportunity to compose on-the-fly several DEs to model the EAC applications.

[4]The AAMAS conference is the leading conference that publishes the works on the different aspects of the MAS. The AAMAS conference series can be found at this page: www.aamas-conference.org

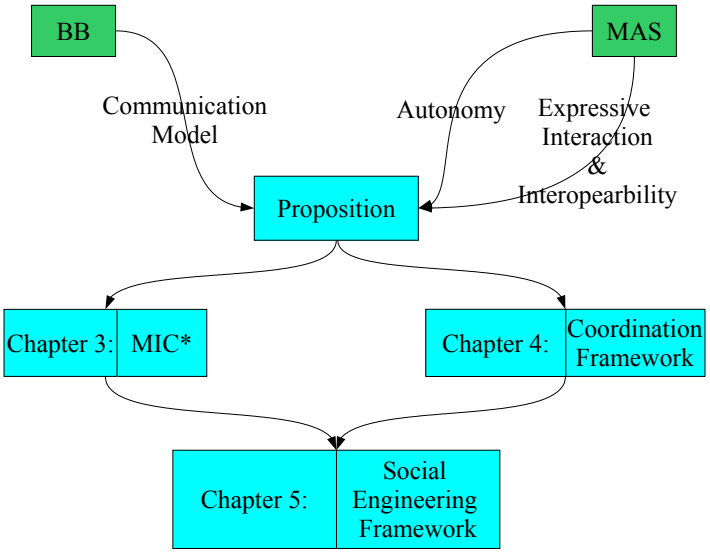


Figure 2.7: Contributions of the state of the art to our proposition and organization of the following chapters.

- **Chapter 4**: This chapter is concerned with the interoperability of open systems from a coordination point of view. Starting from the theory of coordination of Malone and Crowston [Malone & Crowston, 1994] a dependency-based coordination framework is presented in order to model coordination processes between distributed software entities. Furthermore, by using the Queue Petri Nets (QPNs), the coordination process is monitored by the DE only by observing the conversations of the agents.
- **Chapter 5**: Finally, this chapter gathers the results of the previous chapters in an operational engineering framework. This framework is based on the social metaphor. Hence, an application within the EAC context is viewed as a self-contained open artificial society of software agents that meets and interacts with other open societies. Some applications developed using this engineering framework are presented and deployed in a realistic simulator.

Chapter 3

The MIC* Deployment Environment

"Human Society is becoming increasingly dependant on integrated computer systems. Development of technologies to cope requires a comprehensive, interdisciplinary development of a theoretical base framework. Considered as systems, the interacting processes in their domain must be recognised, analysed, and managed as multi-tool, multi-level, multi-agent feedback systems."

M.M. Lehman, taken from the talk given in the SOCE workshop [SOCE, 2000]

3.1 Introduction

In order to ensure that the software system actually follows the design requirements in an open context, we split the global software system in two orthogonal dimensions: the autonomous agents, representing the active interacting entities designed to achieve certain individual and collective goals; and the deployment environment, representing a structured container that holds these agents.

According to Russell and Norvig [Russell & Norvig, 1995] the environment is one of the fundamental concepts in programming and designing MASs. However, there are still some ambiguities on the definition of the environment. For instance, situated agents, such as agent-based simulation, artificial life or robotics, consider the environment as the actual space where agents are situated; perceive their vicinity; and access to the available resources. On the other hand, agent-based software engineering considers the environment as the software components surrounding the agents and offering them some computing facilities. The later definition is referenced as the *deployment environment* (DE).

The goal of this chapter is to study the properties of the DE independently from the agents themselves. In other words, this chapter identifies a general structure of a DE without considering the models of the autonomous agents (SCMAS approach).

3.2 Motivations for the Explicit Representation of the Deployment Environment

3.2.1 Guaranteeing the Autonomy

Once the definition of the autonomy has been sketched (cf. §2.3.2, page 28), the problem is to know how to implement it. In other words, are usual techniques such as concurrent object sufficient to implement the autonomy? This question has been partially answered by [Luck & d'Inverno, 1995] by introducing the concept of motivation. Hence, motivated agents generate their goals by considering their motivations. Nevertheless, introducing motivations in an agent seems to be just shifting the problem from who controls the agent's goals, to who controls the agent's motivations. In fact, any external entity that has a full control of the software structure representing an agent's motivations controls the behavior of this agent.

The internal integrity of the agents has been considered as a *sine qua none* condition to implement the autonomy (cf. §2.3.2, page 30). In fact, if the agent's software structure is either accessed or modified by another agent, the decisional processes and behaviors are altered. For instance, if an external agent accesses the set of motivations of a motivated-agent, this agent loses its autonomy. Consequently, the autonomous agent should not allow any external agent to change its internal structure either by setting a feature to certain value or by calling a side-effect method.

On the other hand, the agents are still interacting entities. So, the agents change the perceptions of other agents. Since the perceptions of agents are part of their software structure, this contradicts the internal integrity principle.

To avoid this contradiction, a non-agent entity is required [Gouaïch, 2003]. This entity is in charge of achieving the interaction among the agents and is defined as the DE. The DE is not an autonomous agent and is considered as an automatic software system. Consequently, the rules of the DE are defined at once and are applied similarly to all the agents. This does not mean that dynamic DEs cannot be constructed. However, any change of the rules represents a new version of the DE. This result is particularly interesting to establish the agent's responsibility in an open and untrusted context. In fact, without this feature the responsibility of the agents when they challenge the rules cannot be established as shown hereafter.

3.2.2 Enabling the Identification of the Responsibility of the Autonomous Agents

Biological and physical systems are examples of MASs that are separated in two dimensions: the individuals (or agents) and the physical environment. Biological agents are subject to the environmental rules. The agents do not modify the environmental rules; they have to deal with them in order to achieve their goals. For instance, if a human agent's goal is to fly, she/he will never modify the physics law on gravity, but use other physics laws on aerodynamics in order to achieve its goal. This analogy can be used in order to build consistent software systems. In fact, the software system may have some predefined rules that guarantee its consistency. Any deployed autonomous agent has to deal with these rules. Besides, the DE has to identify the agents that challenge the rules. This defines the agent's responsibility as being coherent to the DE rules. To guarantee these points, the agents are not allowed to change directly their environment. Their actions have to be discrete and explicitly represented as attempts

of actions, namely influences [Ferber & Muller, 1996]. The DE defines how to react to these influences. The influences that are not consistent with the DE rules are explicitly handled and the emitting agent is considered as responsible for challenging the DE's rules. Furthermore, the agents cannot deny this responsibility since the DE is an automatic system that does not modify its rules. In fact, if this was not the case, the agents may claim that they cannot conform with the rules since they are arbitrarily changed.

3.2.3 Modeling On-the-fly Composition of the Software Systems

The on-the-fly composition is a fundamental characteristic of the EAC software systems. This composition has to be explicitly modeled at the conceptual level and flexibly implemented at the implementation level. In other words, one should know what is the result of the composition or decomposition of several software systems at any time. We suggest managing the composition of the software systems by composing their DEs. On one hand, the composition of several DEs builds a new DE holding all the software entities of the system. On the other hand, the decomposition of a DE builds several independent sub-systems.

3.3 The MIC* Model

3.3.1 Intuitive Introduction to MIC*

This section introduces informally the MIC* model starting from the BB paradigm and especially from the TS architectures.

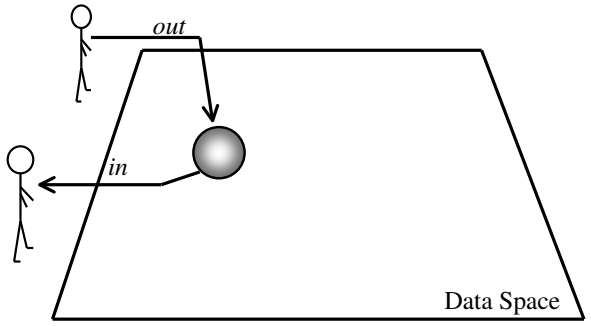


Figure 3.1: Presentation of a simple TS architecture.

We have shown in section [2.4.3, page 35] how TS architectures propose a generative communication between the software entities. The entities communicate by depositing and retrieving the information carriers, called tuples, in a shared and persistent data space (cf.

Figure 3.1). Within the MIC* model the information carriers are called interaction objects (IOs) instead of tuples. We will see after that the IOs do not only represent the information carriers but also the means of the interaction.

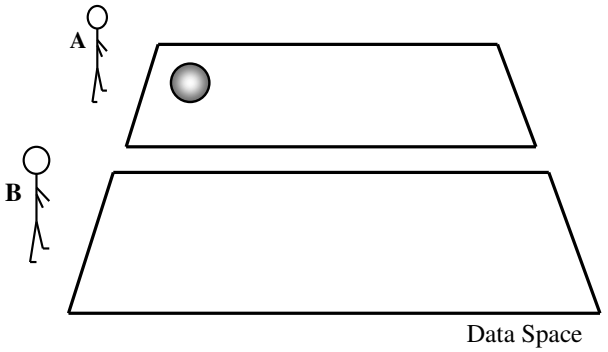


Figure 3.2: The horizontal decomposition of the data space by agents.

The first problem revealed by section [2.4.4, page 37] is related to the structure of the data space. To solve this problem, we suggest to split the data space horizontally such that each software entity owns its proper region where to deposit and retrieve IOs (cf. Figure 3.2). This structural decomposition solves the problem of the shaded actions since two different software entities access different and independent regions.

The other decomposition is vertical and concerns the interaction spaces (cf. Figure 3.3). In fact, it is interesting to gather the IOs that have a common sense within the same context, namely the interaction space (IS). So, the software entities have not a total access to the entire data space but only to some ISs. From a practical point of view the vertical decomposition is also interesting since the search space is divided into small spaces and the effort to lookup information and maintain consistency of the data space is optimized by applying the 'divide and conquer' principle.

By combining the vertical and horizontal decompositions, we arrive to a matrix structure presented in Figure 3.4. The rows of this matrix are the software entities or agents; and the columns are the ISs.

The interaction scheme of the MIC* model is completely different from the one presented by the TS. In fact, the horizontal decomposition avoids accessing the IOs of other software entities. So, the lookup mechanism using the anti-tuples cannot be used. In order to present the interaction scheme of MIC*, first let us present the structure of the IOs.

The IOs are closed under composition. This means that an aggregation of several IOs is still an IO. Besides, within the aggregation of several IOs the order is not important. This is suitable to model simultaneity of IOs since their order of arrival is not considered as an additional information. Finally, we identify a special IO which is empty called the zero 0.

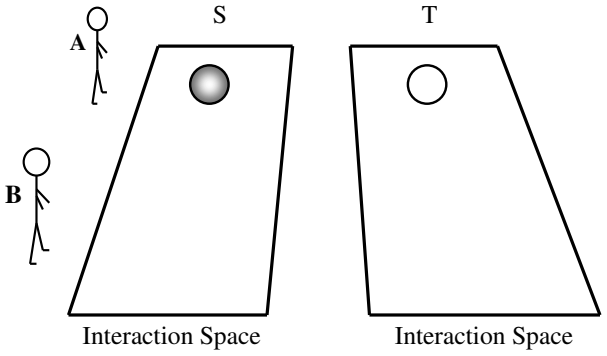


Figure 3.3: The vertical decomposition of the data space by interaction spaces.

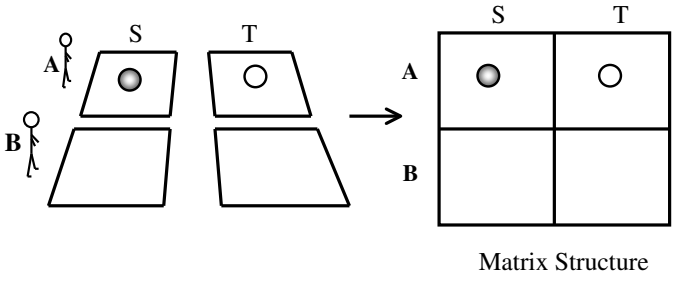


Figure 3.4: Integrating the horizontal and vertical decompositions to build a matrix structure.

The other difference between MIC* and TS concerns the unification of the IOs. In fact, within a TS the information carriers are represented as tuples; while the means of interaction are represented as anti-tuples. Within MIC*, both of the information carriers and interaction means are represented uniformly as IOs.

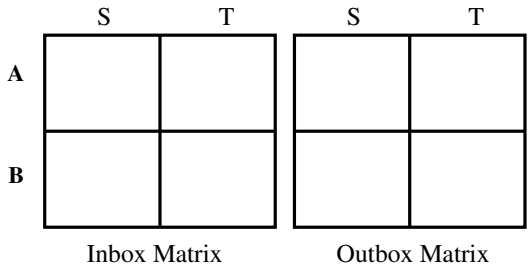


Figure 3.5: The static structure of the MIC* DE.

The TS does not define a place where to hold the interaction means and the result of the interactions when the software entities are not explicitly requesting the interaction (cf. §2.4.4, page 37). For this reason, we suggest to duplicate the matrix of MIC* as an inbox and outbox matrix. The outbox matrix holds the IOs of the agents representing the: information carriers, influences and interaction means. The inbox matrix holds the IOs that have been perceived by the interaction means of the agents. The final structure presented in Figure 3.5 is known as the static structure of MIC*.

Now, we explore the dynamics of this static structure. In other words, we are interested in identifying some evolutions of the static structure that have a special semantics within the MAS paradigm.

The first identifiable evolution of the static structure is the interaction. The interaction is conducted using the *interaction functions* that are locally defined within the context of an IS. The interaction functions take a couple of IOs as argument, namely the *sensor* and *effector*, and returns an IO, namely the *interaction result*. The interaction functions calculate the result of the interaction of the effector on the sensor and the interaction result is then appended to the inbox corresponding to the sensor.

Figure 3.6 presents a simple example. The MIC* DE holds three agents A, B and C. All these agents are in the same IS named S. The agent A's outbox within S contains an IO representing a blue color; the agent B's outbox within S contains an IO representing a red color; and the agent C's outbox within S contains an IO representing a camera sensitive to only blue objects. If S contains an interaction function stating that the camera sensitive to blue objects senses only the blue IOs then the evolution of the MIC* DE adds the blue

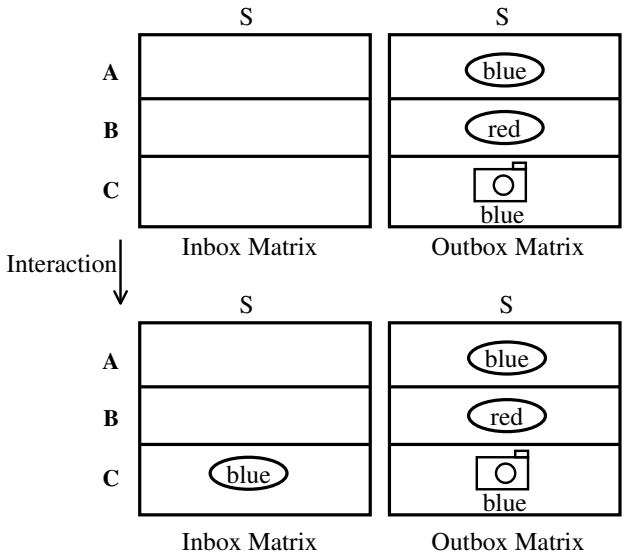


Figure 3.6: The evolution of the DE that is interpreted as an interaction.

IO in the inbox of the agent C. The red IO does not interact with the agent C interaction means. Moreover, since the means of interaction and the information carriers are uniformity represented as IOs, one can easily imagine adding another IO that senses the camera instead of the colors and consequently the observer can be observed.

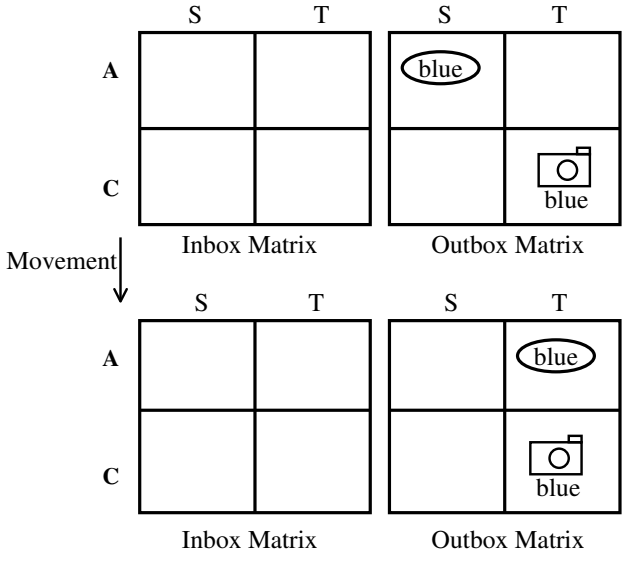


Figure 3.7: The evolution of the DE that is interpreted as a movement.

Another evolution that has been identified concerns the movement of the IOs among different ISs. It is a logical movement where the IOs simply disappear from their original IS

and appear within the destination IS (cf. Figure 3.7). Obviously, when the IOs appear in the destination IS they can interact with other IOs that were already present.

The third evolution is related to the computation of the agents. In order to compute, the agents retrieve their perceptions; deliberate internally and send influences reified as IOs within the outbox matrix.

The composition of software systems is treated by the composition of their DEs. Thus, the common ISs are shared and synchronized on some operations in order to offer a virtual DE composed by the aggregation of the sub DEs. Thanks to the structure of the MIC* model the number of synchronization operations is minimized. In fact, the synchronization occurs only when calculating the interactions of the software entities. In this case, the interaction means expressed as IOs are sent to the remote ISs and the interaction result is returned to the initial IS. Consequently, the disconnections are gracefully handled since they do not imply any management of the consistency of the local data space.

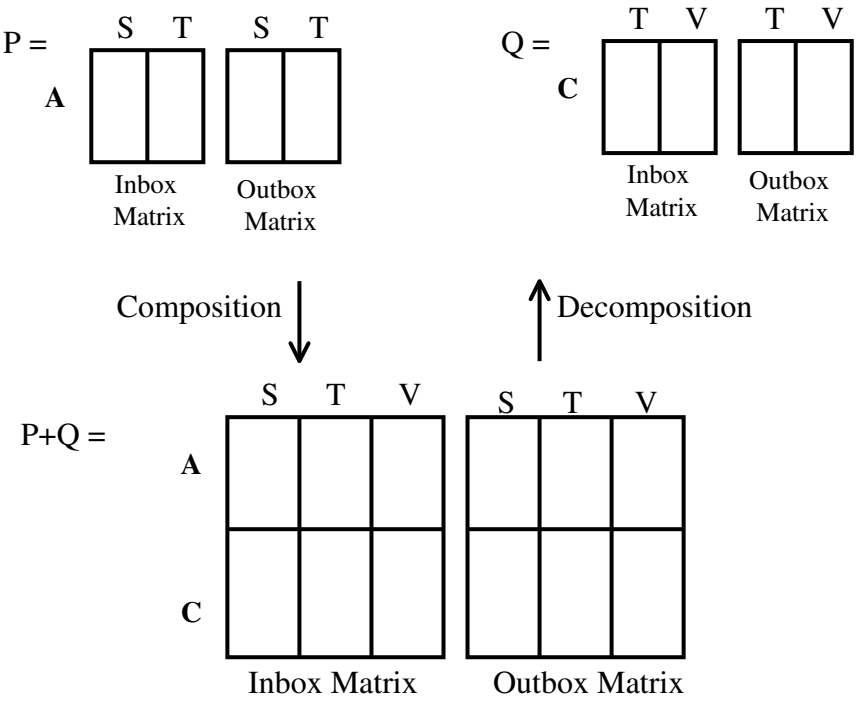


Figure 3.8: Example of the composition of two DEs P and Q.

Figure 3.8 presents an example of two systems P and Q. The system P contains agent A and two ISs S and T. The system Q contains agent C and two ISs T and V. The composition of P and Q leads to a DE composed by the agents A and C and the ISs S, T and V. The agents A and C may interact in this composed DE since they are located in a common IS, namely T. The decomposition of the global DE leads to the initial configuration of P and Q.

3.3.2 Formalizing the Intuitions

The previous informal presentation of MIC* is concretely modeled in this section using simple linear algebra tools. The definition of the MIC* static structure is as follows:

Definition 3.3.1. Let $(\mathcal{O}, +)$ be a commutative group; $\mathcal{A}$ and $\mathcal{S}$ two sets. The $\mathcal{O}$ elements are called interaction objects; $\mathcal{A}$ represents the set of agents, $\mathcal{S}$ the set of interaction spaces. The set of elements $(o_{i,j})_{(i,j)\in\mathcal{A}\times\mathcal{S}}$, where $o_{i,j} \in \mathcal{O}$, is represented by the following notation: $\mathcal{O}^{(\mathcal{A}\times\mathcal{S})}$.
The MIC* DE is defined as a structure composed of two matrices called the outbox and the inbox: $\mathcal{T} = \mathcal{O}^{(\mathcal{A}\times\mathcal{S})} \times \mathcal{O}^{(\mathcal{A}\times\mathcal{S})}$. Each element t of $\mathcal{T}$ is represented as:

$$t = \underbrace{\left(\underbrace{\begin{bmatrix} \underbrace{[o_1]_a}_{\mathbf{(C)}} \\ \vdots \end{bmatrix}_s}_{\mathbf{(B)}} \cdots \right)}_{\mathbf{(A)}} \underbrace{\left(\underbrace{\begin{bmatrix} \underbrace{[i_1]_a}_{\mathbf{(G)}} \\ \vdots \end{bmatrix}_s}_{\mathbf{(F)}} \cdots \right)}_{\mathbf{(E)}}$$

(A) : the outbox matrix ; (B) : the interaction space 's' ; (C) : the outbox of agent 'a' ; (E) : the inbox matrix ; (F) : interaction space 's' ; (G) : the inbox of agent 'a'.

Interpretation 3.3.2. The MIC* structure $\mathcal{T} = \mathcal{O}^{(\mathcal{A}\times\mathcal{S})} \times \mathcal{O}^{(\mathcal{A}\times\mathcal{S})}$ is composed by two matrices that are described as follows:

1. The outbox matrix: the rows of this matrix represent the agents $i \in \mathcal{A}$ and the columns represent the ISs $j \in \mathcal{S}$. Each element of the matrix $o_{(i,j)} \in \mathcal{O}$ is a representation of the agent i in the IS j. This is the only way for an agent to exist and operate in the MAS. So, the elements of this matrix model the means that enable an agent to perceive and to influence the universe in a particular IS. Notice that the means used to perceive the universe are distinguished from the result of the perception. The perception results are placed in the inbox matrix. When $o_{(i,j)} = 0$, the agent i neither influences nor perceives the universe in the IS j: agent i does not exist in IS j.
2. The inbox matrix: the rows of this matrix represent agents $i \in \mathcal{A}$ and the columns represent the ISs $j \in \mathcal{S}$. Each element of the matrix $o_{(i,j)} \in \mathcal{O}$ represents the result of the perceptions of the agent i in the IS j.

Utilities Functions

Some useful functions are introduced in order to access some parts of $\mathcal{T}$ elements:

- $[] : (\mathcal{T}\times\mathcal{A}\times\mathcal{S}) \longrightarrow \mathcal{T}$ is a projection function. $[](t, a, s)$ selects only the elements indexed by the agent a and the interaction space s and sets all the rest to 0. The notation $t[a, s]$ is preferred to $[](t, a, s)$. $t[*, s]$ and $t[a, *]$ are abbreviations for the following expressions:

$$t[*, s] = \sum_{k\in\mathcal{A}} t[k, s]$$

$$t[a, *] = \sum_{k\in\mathcal{S}} t[a, k]$$

- $out : \mathcal{T} \longrightarrow \mathcal{O}^{(\mathcal{A}\times\mathcal{S})}$ selects the outbox matrix of the MIC* element.
- $in : \mathcal{T} \longrightarrow \mathcal{O}^{(\mathcal{A}\times\mathcal{S})}$ selects the inbox matrix of the MIC* element.
- $t \in \mathcal{T}, out(t)_{j\in\mathcal{S}} \in \mathcal{O}^{(\mathcal{A})}$: $(out(t))_j$ selects the j column of the outbox.
- $t \in \mathcal{T}, in(t)_{j\in\mathcal{S}} \in \mathcal{O}^{(\mathcal{A})}$: $(in(t))_j$ selects the j column of the inbox.

These functions are extended to access the information about a particular agent $a \in \mathcal{A}$ within a particular interaction space $s \in \mathcal{S}$ as follows:

- $out_{(i\in\mathcal{A},j\in\mathcal{S})} : \mathcal{T} \longrightarrow \mathcal{O}$ selects the outbox of the agent i in the interaction space j.
- $in_{(i\in\mathcal{A},j\in\mathcal{S})} : \mathcal{T} \longrightarrow \mathcal{O}$ selects the inbox of the agent i in the interaction space j.

Interpretation 3.3.3. *out* and *in* are tools to select particular components of MIC* elements $t \in \mathcal{T}$. These functions are extended to select a particular agent's outbox and inbox within a particular IS.

3.3.3 MIC* Elements as Algebraic Equations

To define formally the composition of MIC* elements it is interesting to have a linear representation. Still, before presenting this linear representation, we need to introduce formally some structures and their properties. The first definition presents the formal definition of indexed sequences.

Definition 3.3.4. Let G be a group and X a set. $G^{(X)}$ represents the set of indexed sequences $x = [x_i]_{i\in X}$. The elements of the sequence x belong to G and are indexed by the elements of X. Each index, element of X, is used only once in the sequence x.

Interpretation 3.3.5. The structure $G^{(X)}$ defines a mapping: G represents the values, and the elements of X represent the keys. A key cannot be used to index more that one value.

Definition 3.3.6. Let G be a group and X a set; the natural sequence composition law, $+$, of the $G^{(X)}$ structure is defined as follows:

$$\begin{aligned} + : G^{(X)} \times G^{(X)} &\longrightarrow G^{(X)} \\ (x, y) &\mapsto z \ : \ \forall i \in X \ z_i = x_i + y_i \end{aligned}$$

Interpretation 3.3.7. In other words, we have defined a composition law between map tables. The resulting map table is defined by the elements of the sub-tables, which are composed only and only if they have the same key.

Proposition 3.3.8. If G is a group, then the structure $(G^{(X)}, +)$ defines also a group. Besides, if G is a commutative group then $(G^{(X)}, +)$ is also a commutative group.

Proof. The proof of this proposition is quite simple. In fact, one has to check the following properties to demonstrate that $(G^{(X)}, +)$ is a group:

- neutral element: the neutral element for $G^{(X)}$ is simply the sequence containing only 0_G, the neutral element of the group G. This element is represented by $0_{G^{(X)}}$.

$$\begin{aligned} \forall x \in G^{(X)}\ x + 0_{G^{(X)}} &= [x_i]_{i\in X} + 0_{G^{(X)}} \\ &= [x_i + 0_G]_{i\in X} \\ &= [x_i]_{i\in X} \\ &= [0_G + x_i]_{i\in X} \\ &= 0_{G^{(X)}} + x \\ &= x \end{aligned}$$

- inverses: each element of the set $G^{(X)}$ has an opposite that is defined as follows:

$$\forall\ x \in G^{(X)}\ x^{-1} : \forall j \in X\ x_j^{-1} = -(x_j)$$

One can easily check that:

$$\begin{aligned} \forall\ x \in G^{(X)}\ x + x^{-1} &= [x_j]_{j\in X} + [(-x)_j]_{j\in X} \\ &= [(x + -x)_j]_{j\in X} \\ &= [(0_G)_j]_{j\in X} \\ &= 0_{G^{(X)}} \end{aligned}$$

- associativity:

$$\begin{aligned} \forall\ x, y, z \in G^{(X)}\ (x + y) + z &= [(x_i + y_i) + z_i]_{i\in X} \\ &= [x_i + (y_i + z_i)]_{i\in X} \\ &= x + (y + z) \end{aligned}$$

The structure $(G^{(X)}, +)$ has been proved to be a group. When G is commutative, one has to check that $(G^{(X)}, +)$ defines also a commutative group:

- commutativity:

$$\begin{aligned} \forall\ x, y \in G^{(X)} x + y &= [x_i + y_i]_{i\in X} \\ &= [y_i + x_i]_{i\in X} \\ &= y + x \end{aligned}$$

So, when G is commutative group then, $(G^{(X)}, +)$ is also a commutative group. □

Definition 3.3.9. Let $\mathcal{O}$ be the commutative group of IOs and $\mathcal{A}$ the set of agents. $\mathcal{O}^{(\mathcal{A})}$ is defined as the set of sequences of IOs that are indexed by agents. We admit the following notation, x_p, to access an IO in a sequence x indexed by the agent p.

Corollary 3.3.10. $\mathcal{O}^{(\mathcal{A})}$ defines a commutative group under the + law used in definition 3.3.6.

Example 3.3.1. Let $\mathcal{A} = \{a, b, c, d\}$ and $\mathcal{O} = (\mathbb{Z}, +)$. In this case $\mathcal{O}^{(\mathcal{A})}$ is defined as the set of sequences of four number elements of $\mathbb{Z}$ that are indexed with the elements of $\mathcal{A}$:

$$\mathcal{O}^{(\mathcal{A})} = \{[(x)_a, (y)_b, (z)_c, (v)_d]\ :\ (x, y, z, v) \in \mathbb{Z}^4\}$$

The sequence $x = [(-1)_a, (0)_b, (1)_c, (2)_d]$ belongs to $\mathcal{O}^{(\mathcal{A})}$ and one may retrieve a particular interaction object by using elements of $\mathcal{A}$ as index. For instance, $x_d = 2$.

Definition 3.3.11 (mapping interaction objects to agents)**.** Let . (dot) be a law defined between IOs set and the agents set as follows:

$$\begin{array}{rcl} . : \mathcal{O} \times \mathcal{A} & \longrightarrow & \mathcal{O}^{(\mathcal{A})} \\ (e,p) & \mapsto & o.p = x \ : \ \forall j \in \mathcal{A} \begin{cases} x_j = e, & \text{for } j = p \\ x_j = 0, & \text{for } j \neq p. \end{cases} \end{array}$$

Example 3.3.2. This example uses the same definition of $\mathcal{O}$ and $\mathcal{A}$ as example 3.3.1. Using the . (dot) law, one can easily build elements of $\mathcal{O}^{(\mathcal{A})}$. For instance $x = (-1).(d)$ belongs to $\mathcal{O}^{(\mathcal{A})}$ and is defined as follows: $x = [(0)_a, (0)_b, (0)_c, (-1)_d]$. One can notice that the . (dot) law is distributed over the $+$. For instance:

$$\begin{array}{rcl} (3).(d) + (-2).(d) & = & [(0)_a, (0)_b, (0)_c, (3)_d] + [(0)_a, (0)_b, (0)_c, (2)_d] \\ & = & [(0)_a, (0)_b, (0)_c, (2-3)_d] \\ & = & [(0)_a, (0)_b, (0)_c, (-1)_d] \\ & = & (-1).(d) \\ & = & (2-3).(d) \end{array}$$

Proposition 3.3.12. The . (dot) law is distributed on the $+$ law:

$$\forall (m,n,p) \in \mathcal{O} \times \mathcal{O} \times \mathcal{A} : (m+n).p = m.p + n.p$$

Proof.

$$\begin{array}{rcl} (m+n).p & = & [x_j]_{j \in \mathcal{A}} \ : \ \begin{cases} x_j = m+n, & \text{for } j = p \\ x_j = 0, & \text{for } j \neq p. \end{cases} \\ & = & [x'_j]_{j \in \mathcal{A}} + [x''_j]_{j \in \mathcal{A}} \ : \ \begin{cases} x'_j = m, x''_j = n & \text{for } j = p \\ x'_j = 0, x''_j = 0, & \text{for } j \neq p. \end{cases} \\ & = & m.p + n.p \end{array}$$

□

Definition 3.3.13. Let $\mathcal{O}$ be the commutative group of IOs; $\mathcal{A}$ the set of agents and $\mathcal{S}$ the set of ISs. $\mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}}$ is defined as the set of sequences of elements of $\mathcal{O}^{(\mathcal{A})}$ that are indexed by interaction spaces. We admit the following notation, $x_{(p,s)}$, to access an interaction object in a sequence x indexed by the agent p and the interaction space s.

Corollary 3.3.14. $\mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}}$ defines a commutative group under the $+$ law defined in definition 3.3.6.

Proof. Since $\mathcal{O}^{(\mathcal{A})}$ is a commutative group, then the structure $\mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}}$ is also a commutative group according to proposition 3.3.8. □

Example 3.3.3. In this example we use the following values for each set:

$$\begin{array}{rcl} \mathcal{O} & = & (\mathbb{Z}, +) \\ \mathcal{A} & = & \{a, b\} \\ \mathcal{S} & = & \{x, y\} \end{array}$$

In this case $\mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}}$ represents sequences indexed by interaction spaces of sequences of interaction objects indexed by agents:

$$\mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}} = \{[([(\alpha_1)_a, (\alpha_2)_b])_x, ([(\alpha_3)_a, (\alpha_4)_b])_y] \ : \ (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{Z}\}$$

$x = [[(-1)_a, (-2)_b]_x, [(-3)_a, (-4)_b]_y]$ is an example of an element that belongs to $\mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}}$. To access the interaction object of the agent a located in the interaction space y one has to write the following expression $(x_a)_y = -3$ (or $x_{(a,y)}$).

Definition 3.3.15 (mapping agents to interaction spaces)**.** Let @ (at) be a law defined between $\mathcal{O}^{(\mathcal{A})}$ elements and interaction spaces as follows:

$$\begin{array}{rcl} @ : \mathcal{O}^{(\mathcal{A})} \times \mathcal{S} & \longrightarrow & \mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}} \\ (t, s) & \mapsto & t@s = x \ : \ \forall j \in \mathcal{S} \begin{cases} x_j = t, & \text{for } j = s \\ x_j = 0, & \text{for } j \neq s. \end{cases} \end{array}$$

Example 3.3.4. This example uses the same definition of $\mathcal{O}$, $\mathcal{A}$ and $\mathcal{S}$ as example 3.3.3. We consider the following expression that uses both of the . (dot) law and @ (at) law to build an element belonging to $\mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}}$:

$$\begin{array}{rcl} ((-3).(b))@(y) & = & ([(0)_a, (-3)_b])@(y) \\ & = & [[(0)_a, (0)_b]_x, [(0)_a, (-3)_b]_y] \end{array}$$

As shown by this example, the elements of $\mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}}$ are sequences indexed by interaction spaces, $\{x, y\}$, of other sequences containing interaction objects indexed by agents $\{a, b\}$.

Proposition 3.3.16. The @ law is distributed on + agents law:

$$\forall (x, y, s) \in (\mathcal{O}^{(\mathcal{A})} \times \mathcal{O}^{(\mathcal{A})} \times \mathcal{S}) \ (x + y)@s = x@s + y@s$$

Proof. The proof of this proposition is very similar to the one presented for the distribution of the . (dot) law on interaction objects. □

Having these definitions, we are now able to translate any element of MIC* $t \in \mathcal{T}$ as a system of equations defined by using a translation function ϑ as follows:

$$\begin{array}{rcl} \vartheta : \mathcal{T} & \longrightarrow & \mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}} \times \mathcal{O}^{(\mathcal{A})^{(\mathcal{S})}} \\ t & \mapsto & \vartheta(t) = \begin{cases} \text{outbox:} & \sum_{s \in \mathcal{S}} (\sum_{p \in \mathcal{A}} out_{p,s}(t).p)@s \\ \text{inbox:} & \sum_{s \in \mathcal{S}} (\sum_{p \in \mathcal{A}} in_{p,s}(t).p)@s \end{cases} \end{array}$$

Proposition 3.3.17. ϑ is a bijective function.

Proof.

- ϑ is an onto application. From any linear expression it is possible to build trivially an element that belongs to $\mathcal{T}$. In fact, one has just to build the outbox and inbox matrices and setting the values of their cells using the corresponding values in the linear expression. Thanks to the associativity and distribution properties, one can write each linear expression in this form:

$$\forall p \in \mathcal{O}^{(\mathcal{A})(\mathcal{S})} \times \mathcal{O}^{(\mathcal{A})(\mathcal{S})}, p = \begin{cases} \text{outbox:} & \sum_{j\in\mathcal{S}}(\sum_{i\in\mathcal{A}} \alpha_{i,j}.i)@j \\ \text{inbox:} & \sum_{j\in\mathcal{S}}(\sum_{i\in\mathcal{A}} \beta_{i,j}.i)@j \end{cases}$$

  The element, $\vartheta^{-1}(p)$, is well defined such that:

$$\vartheta^{-1}(v) \in \mathcal{T} : \forall (i,j) \in \mathcal{S} \times \mathcal{A} \begin{cases} out_{(i,j)}(\vartheta^{-1}(p)) = x_{(i,j)} = \alpha_{i,j} \\ in_{(i,j)}(\vartheta^{-1}(p)) = y_{(i,j)} = \beta_{i,j} \end{cases}$$

  Where,

$$\begin{cases} x = & \sum_{j\in\mathcal{S}}(\sum_{i\in\mathcal{A}} \alpha_{i,j}.i)@j \\ y = & \sum_{j\in\mathcal{S}}(\sum_{i\in\mathcal{A}} \beta_{i,j}.i)@j \end{cases}$$

- ϑ is a one to one application:

$$\begin{aligned} \forall t_1, t_2 \in \mathcal{T}, \vartheta(t_1) = \vartheta(t_2) \quad &\Longrightarrow \quad \forall i,j \in \mathcal{A} \times \mathcal{S} \begin{cases} out_{(i,j)}(t_1) = out_{(i,j)}(t_2) \\ in_{(i,j)}(t_1) = in_{(i,j)}(t_2) \end{cases} \\ &\Longrightarrow \quad t_1 = t_2 \end{aligned}$$

□

Summary of the Algebraic Notations Properties

This is a summary of the introduced algebraic notations. $x, y, z \in \mathcal{O}$ are interaction objects; $a, b, c \in \mathcal{A}$ are agents; and $s, t, v \in \mathcal{S}$ are interaction spaces.

Properties of MIC* Algebraic Objects:	
Neutral elements	$0.a = 0$ $0 + x.a = x.a + 0 = x.a$ $0@s = 0$ $0 + x.a@s = x.a@s + 0 = x.a@s$
Inverse elements	$-(x.a) = (-x).a$ $-(x.a@s) = (-x).a@s$
Commutativity of interaction objects under the + law	$(x + y).a@s = (y + x).a@s$
Associativity of interaction objects	$((x + y) + z).a@s = (x + (y + z)).a@s$
Commutativity of Agents under the + law	$(x.a + y.b)@s = ((y.b + x.a)@s$
Associativity of Agents under the + law	$((x.a + y.b) + z.c)@s = (x.a + (y.b + z.c))@s$

Properties of MIC* Algebraic Objects:	
Commutativity of Spaces under + law	$(x.a)@s + (y.b)@t = (y.b)@t + (x.a)@s$
Associativity of Spaces under + law	$((x.a)@s + (y.b)@t) + (z.c)@u$ $= (x.a)@s + ((y.b)@t + (z.c)@u)$
Distribution of Agents on interaction objects	$(x + y).p = (y).p + (x).p$
Distribution of Spaces on Agents	$(x.a + y.b)@s = x.a@s + y.b@s$

Example 3.3.5 (Illustrating the projections and linear notations)**.** Let $\mathcal{A} = \{a, b\}$ be the set of agents and $\mathcal{S} = \{s, t\}$ the set of interaction spaces. Let t_1 be a MIC* element defined as follows using the matrix notation:

$$\begin{aligned} t_1 &\in \mathcal{O}^{(\mathcal{A}\times\mathcal{S})} \times \mathcal{O}^{(\mathcal{A}\times\mathcal{S})} \\ t_1 &= \left(\begin{bmatrix}[o_1]_a \\ [o_3]_b\end{bmatrix}_s \quad \begin{bmatrix}[o_2]_a \\ [o_4]_b\end{bmatrix}_w\right)\left(\begin{bmatrix}[i_1]_a \\ [i_3]_b\end{bmatrix}_s \quad \begin{bmatrix}[i_2]_a \\ [i_4]_b\end{bmatrix}_w\right) \end{aligned}$$

The defined projection functions give the following results for this element:

$$\begin{aligned} t_1[a,s] &= \left(\begin{bmatrix}[o_1]_a \\ [0]_b\end{bmatrix}_s \quad \begin{bmatrix}[0]_a \\ [0]_b\end{bmatrix}_w\right)\left(\begin{bmatrix}[i_1]_a \\ [0]_b\end{bmatrix}_s \quad \begin{bmatrix}[0]_a \\ [0]_b\end{bmatrix}_w\right) \\ t_1[a,*] &= \left(\begin{bmatrix}[o_1]_a \\ [0]_b\end{bmatrix}_s \quad \begin{bmatrix}[o_2]_a \\ [0]_b\end{bmatrix}_w\right)\left(\begin{bmatrix}[i_1]_a \\ [0]_b\end{bmatrix}_s \quad \begin{bmatrix}[i_2]_a \\ [0]_b\end{bmatrix}_w\right) \\ t_1[*,s] &= \left(\begin{bmatrix}[o_1]_a \\ [o_3]_b\end{bmatrix}_s \quad \begin{bmatrix}[0]_a \\ [0]_b\end{bmatrix}_w\right)\left(\begin{bmatrix}[i_1]_a \\ [i_3]_b\end{bmatrix}_s \quad \begin{bmatrix}[0]_a \\ [0]_b\end{bmatrix}_w\right) \\ out(t_1) &= \left(\begin{bmatrix}[o_1]_a \\ [o_3]_b\end{bmatrix}_s \quad \begin{bmatrix}[o_2]_a \\ [o_4]_b\end{bmatrix}_w\right) \\ out_{a,w}(t_1) &= o_2 \\ out_{a,s}(t_1) &= o_1 \\ out_{b,w}(t_1) &= o_4 \\ out_{b,s}(t_1) &= o_3 \\ in(t_1) &= \left(\begin{bmatrix}[i_1]_a \\ [i_3]_b\end{bmatrix}_s \quad \begin{bmatrix}[i_2]_a \\ [i_4]_b\end{bmatrix}_w\right) \\ in_{a,w}(t_1) &= i_2 \\ in_{a,s}(t_1) &= i_1 \\ in_{b,w}(t_1) &= i_4 \\ in_{b,s}(t_1) &= i_3 \end{aligned}$$

When using the linear notations, t_1 is represented as:

$$t_1 = \begin{cases} \text{outbox:} & ((o_1).a + (o_3).b)@s + ((o_2).a + (o_4).b)@w \\ \text{inbox:} & ((i_1).a + (i_3).b)@s + ((i_2).a + (i_4).b)@w \end{cases}$$

Example 3.3.6 (Illustrating the Composition of MIC* Elements)**.** This example presents the composition of MIC* elements. Let us suppose that two independent environments $t_1 \in \mathcal{T}$

and $t_2 \in \mathcal{T}$ are defined as follows:

$$\begin{aligned} t_1 &= \left(\begin{bmatrix} [o_1]_a \\ [0]_b \end{bmatrix}_s \quad \begin{bmatrix} [0]_a \\ [o_2]_b \end{bmatrix}_w \right) \left(\begin{bmatrix} [i_1]_a \\ [0]_b \end{bmatrix}_s \quad \begin{bmatrix} [0]_a \\ [i_2]_b \end{bmatrix}_w \right) \\ t_2 &= \left(\begin{bmatrix} [o_3]_c \end{bmatrix}_s \quad \begin{bmatrix} [0]_c \end{bmatrix}_w \quad \begin{bmatrix} [o_4]_c \end{bmatrix}_v \right) \left(\begin{bmatrix} [i_3]_c \end{bmatrix}_s \quad \begin{bmatrix} [0]_c \end{bmatrix}_w \quad \begin{bmatrix} [i_4]_c \end{bmatrix}_v \right) \end{aligned}$$

t_1 contains two interaction spaces s and w and two agents a and b. t_2 contains three interaction spaces s, w, v and only one agent c.

By using the linear notations t_1 and t_2 are written as follows:

$$\begin{aligned} t_1 &= \begin{cases} \text{outbox:} & o_1.a@s + o_2.b@w \\ \text{inbox:} & i_1.a@s + i_2.b@w \end{cases} \\ t_2 &= \begin{cases} \text{outbox:} & o_3.c@s + o_4.c@v \\ \text{inbox:} & i_3.c@s + i_4.c@v \end{cases} \end{aligned}$$

Now let us suppose that these two terms are composed. This is translated in the MIC* model as the composition under the + law:

$$\begin{aligned} & t_3 = t_1 + t_2 \\ &= \begin{cases} \text{outbox:} & o_1.a@s + o_2.b@w + o_3.c@s + o_4.c@v \\ \text{inbox:} & i_1.a@s + i_2.b@w + i_3.c@s + i_4.c@v \end{cases} \\ &= \begin{cases} \text{outbox:} & (o_1.a + o_3.c)@s + o_2.b@w + o_4.c@v \\ \text{inbox:} & (i_1.a + o_3.c)@s + i_2.b@w + i_4.c@v \end{cases} \\ &= \left(\begin{bmatrix} [o_1]_a \\ [0]_b \\ [o_3]_c \end{bmatrix}_s \quad \begin{bmatrix} [0]_a \\ [o_2]_b \\ [0]_c \end{bmatrix}_w \quad \begin{bmatrix} [0]_a \\ [0]_b \\ [o_4]_c \end{bmatrix}_v \right) \left(\begin{bmatrix} [i_1]_a \\ [0]_b \\ [i_3]_c \end{bmatrix}_s \quad \begin{bmatrix} [0]_a \\ [i_2]_b \\ [0]_c \end{bmatrix}_w \quad \begin{bmatrix} [0]_a \\ [0]_b \\ [i_4]_c \end{bmatrix}_v \right) \end{aligned}$$

The composed deployment environment t_3 is made of all the interaction spaces and agents found in the sub-environments. One can notice that when the agents are in the same interaction space they are still located in the same interaction space in the composed environment. So, they can interact using interaction mechanisms defined in section [3.3.4, page 66].

Example 3.3.7 (Illustrating the Decomposition of MIC* Environments)**.** One can also consider the disjunction of deployment environments. For instance, let us calculate the retrieval of t_2 from the composed environment t_3 defined in the previous example.

$$\begin{aligned} t_4 &= t_3 - t_2 \\ &= \begin{cases} \text{outbox:} & (o_1.a + o_3.c)@s + o_2.b@w + o_4.c@v - (o_3.c@s + o_4.c@v) \\ \text{inbox:} & (i_1.a + o_3.c)@s + i_2.b@w + i_4.c@v - (i_3.c@s + i_4.c@v) \end{cases} \\ &= \begin{cases} \text{outbox:} & o_1.a@s + o_2.b@w \\ \text{inbox:} & i_1.a@s + i_2.b@w \end{cases} \\ &= \left(\begin{bmatrix} [o_1]_a \\ [0]_b \end{bmatrix}_s \quad \begin{bmatrix} [0]_a \\ [o_2]_b \end{bmatrix}_w \right) \left(\begin{bmatrix} [i_1]_a \\ [0]_b \end{bmatrix}_s \quad \begin{bmatrix} [0]_a \\ [i_2]_b \end{bmatrix}_w \right) \\ &= t_1 \end{aligned}$$

This result is correct and shows that when the t_2 leaves the composed environment, only t_1 remains.

Generating the Interaction Objects Commutative Group from a Simple Set

The structure of the interaction objects is very important in the definition of MIC*. In fact, the assumption that the interaction object set owns a structure of a commutative group has been widely used in almost all the presented definitions. However, one may ask what happens if the studied MAS does not present a 'natural' structure of a commutative group for its interaction objects (messages, pheromones and so on). The goal here is to show that starting from any set that has no precise structure, one may define a formal structure of a commutative group of interaction objects.

Definition 3.3.18. Let E be the interaction objects generator set. Let $\mathbb{Z}^{(E)}$ be the set of sequences of integers indexed by the elements of E. $\mathcal{O}[E] = (\mathbb{Z}^{(E)}, +)$ is defined as the interaction objects group generated by the set E.

Proof. Since $\mathbb{Z}$ is a commutative group, the structure $(\mathbb{Z}^{(E)}, +)$ is also a commutative group under the $+$ law defined in definition 3.3.6. □

Definition 3.3.19. The construction function τ is defined among the generator set E and the interaction object group $\mathcal{O}[E]$ as follows:

$$\begin{array}{rcl} \tau : E & \longrightarrow & \mathcal{O}[E] \\ m & \mapsto & \tau(m) = x : \forall j \in G \begin{cases} x_j = 1, & \text{for } j = m \\ x_j = 0, & \text{for } j \neq m \end{cases} \end{array}$$

When there is no ambiguity, the expression $\tau(m)$ is written simply as m.

Interpretation 3.3.20. The set E represents the elementary interaction objects or the atoms that are used to build the more complex structure $\mathcal{O}[E]$. $\mathcal{O}[E]$ represents simply a sequence of integers that are indexed by the atoms; each value expresses the number of occurrence of the atom in an interaction object element. This value may be negative. The construction function τ builds a sequence that represents only one atom. A sequence that represents an atom, sets the value of the element indexed by this atom to 1 and all other values are set to 0.

Example 3.3.8. Let $G = \{\text{'a'}, \text{'b'}\}$ be the generator set; 'a' and 'b' are considered as characters (or integers if we consider their ASCII encoding values) and have not to be misinterpreted with mathematical abstract variables. In this case, $\mathcal{O}[\{\text{'a'}, \text{'b'}\}]$ represents the set of all words written using these characters. Notices that since the interaction objects are assumed to be commutative the word[1] 'aabab' is exactly the same as 'bbaaa'. In fact, both of these words are equals to $((3)_{\text{'a'}}, (2)_{\text{'b'}})$.

The elements of $\mathcal{O}[\{\text{'a'}, \text{'b'}\}]$ can be manipulated in the same way as the commutative groups elements are manipulated. For instance one may be interested in calculating the

[1]The complete writing of these expressions is $\tau(\text{'a'}) + \tau(\text{'a'}) + \tau(\text{'b'}) + \tau(\text{'a'}) + \tau(\text{'b'})$ and $\tau(\text{'b'}) + \tau(\text{'b'}) + \tau(\text{'a'}) + \tau(\text{'b'}) + \tau(\text{'a'})$.

following expression:

$$\begin{aligned}
\text{'b\={a}'} + \text{'aabab'} - \text{'bbbaba'} &= [(-1)_{\text{'a'}}, (1)_{\text{'b'}}] + [(3)_{\text{'a'}}, (2)_{\text{'b'}}) - ((2)_{\text{'a'}}, (3)_{\text{'b'}}] \\
&= [(-1)_{\text{'a'}}, (1)_{\text{'b'}}] + [(3)_{\text{'a'}}, (2)_{\text{'b'}}) + ((-2)_{\text{'a'}}, (-3)_{\text{'b'}}] \\
&= [(-1)_{\text{'a'}}, (1)_{\text{'b'}}] + [(3-2)_{\text{'a'}}, (2-3)_{\text{'b'}}] \\
&= [(-1)_{\text{'a'}}, (1)_{\text{'b'}}] + [(1)_{\text{'a'}}, (-1)_{\text{'b'}}] \\
&= [(0)_{\text{'a'}}, (0)_{\text{'b'}}] \\
&= 0
\end{aligned}$$

3.3.4 The Dynamics of MIC*

An element $t \in \mathcal{T}$ is an instantaneous snapshot of the DE state. Within all the functions defined from $\mathcal{T}$ to $\mathcal{T}$, MIC* considers three classes which have a special semantics for MASs:

Interaction (φ): Two agents are considered as interacting when the perceptions of an agent are influenced by the influences of another. Consequently, the interaction modifies the perception results of an agent (defined in the inbox) according to its perception means and others influences (both defined in the outbox) within a defined IS. The set of all interaction evolutions is represented as φ.

Movement (μ): The mobility of an agent is defined as the mobility of its IOs among different ISs. During a movement no IO is created nor lost. In fact, this is an interesting feature to prevent incoherent duplications by guaranteeing that an agent actually disappears from its original IS and appears in its destination IS. The set of all movement evolutions is represented by μ.

Computation (γ): The computation is an internal process of the autonomous agents. The only way to observe that an agent has conducted a computation is when it changes autonomously its outboxes within ISs. To avoid confusion between the computation and movement, after a computation, the agents conserve their presence. In other words, an agent is not allowed to appear (respectively to disappear) suddenly in an IS when it was not present (respectively present) before the computation. Besides, the agents are rational entities that change their emissions according to their perceptions. So, the agent consumes its perceptions in order to make a computation. This is expressed in MIC* by resetting the inbox of the computing agent to 0. The set of all computation evolutions is represented by γ.

The core idea is that *(i)* the DE dynamics is discrete and *(ii)* any state of the DE is reached from the initial state by a sequence of functions that may be of three classes: (M)ovement, (I)nteraction, and (C)omputation (MIC*).

The above informal descriptions of movement, interaction and computation classes of evolution are formalised as follows:

Movement (μ)

Any movement evolution law is a function $m \in \mu$ defined on $\mathcal{T} \longrightarrow \mathcal{T}$ that fulfils the following requirements:

1) m does not modify the inboxes :

$$in \circ m = in$$

2) m leaves globally invariant the outboxes of a given agent:

$$\forall i \in \mathcal{A}, \quad \sum_{j \in \mathcal{S}} out_{i,j} \circ m \; = \; \sum_{j \in \mathcal{S}} out_{i,j}$$

3) The outboxes of an agent after movement depend only on the outboxes of the same process before the movement :

$$\forall t \in \mathcal{T} \quad \forall (i,j) \in \mathcal{A} \times \mathcal{S}, \quad out_{i,j} \circ m(t) \; = \; out_{i,j} \circ m(t[i,*])$$

4) A movement law, m, is the same for all agents:

$$\begin{aligned} \forall t \in \mathcal{T}, \, \forall j \in \mathcal{S}, \, \forall i, i' \in \mathcal{A}, out_{i,j}(t) \quad &= \quad out_{i',j}(t) \\ &\Rightarrow \\ out_{i,j}(m(t)) \quad &= \quad out_{i',j}(m(t)). \end{aligned}$$

Interaction (φ)

Any interaction evolution law is a function $f \in \varphi$ defined on $\mathcal{T} \longrightarrow \mathcal{T}$ that fulfils the following requirements:

1) f does not modify the outboxes :

$$out \circ f = out$$

2) When an agent is not present in a particular interaction space, it cannot perceive any interaction object:

$$\forall t \in \mathcal{T}, \forall i \in \mathcal{A}, \forall j \in \mathcal{S}, \quad out_{i,j}(t) = 0 \; \Rightarrow \; in_{i,j}(f(t)) = in_{i,j}(t)$$

3) The inbox of an agent after the interaction depends only on the outboxes of the other agents present in the same interaction space:

$$\forall t \in \mathcal{T}, \; \forall (i,j) \in \mathcal{A} \times \mathcal{S}, \quad in_{i,j} \circ f(t) \; = \; in_{i,j} \circ f(t[*,j])$$

Computation (γ)

A computation evolution law is a function $g \in \gamma$ defined on $\mathcal{T} \longrightarrow \mathcal{T}$ that fulfils the following requirements:

1) when an agent modifies one of its outboxes, its inbox is set to the empty interaction object 0:

$$\begin{aligned} \forall \quad & t \in \mathcal{T}, \forall i \in \mathcal{A}, \quad \Big(\exists j \in \mathcal{S}, out_{i,j} \circ g(t) \neq out_{i,j}(t) \Big) \\ \Rightarrow \quad & in_{i,j} \circ g(t) = 0 \end{aligned}$$

2) The outboxes of a particular agent depend only on the values of its inboxes and outboxes before the computation:

$$\forall t \in \mathcal{T}, \; \forall (i,j) \in \mathcal{A} \times \mathcal{S}, \quad out_{i,j} \circ g(t) \quad = \quad out_{i,j} \circ g(t[i,*])$$

Finally, the MIC* DE is fully defined by giving the commutative group $\mathcal{O}$ of interaction objects; the sets $\mathcal{A}$ and $\mathcal{S}$ representing the set of agents and interaction spaces; the set of movements, interactions and computations evolution functions and an initial state of the DE $t_0 \in \mathcal{T}$.

Example of Evolution Functions

The dynamics of MIC* is specified at a high-level. In fact, the dynamics does not specify exactly what are the evolution functions but gives only some specifications in order to consider an evolution function either as a movement, interaction or computation. In this section, we define some specific evolution functions that meet the presented requirements. Other MASs may have their own definition of the evolution functions and are still fitting the MIC* model, if the specifications are met.

Example of Interaction Evolution: As an example of an interaction evolution of the MIC* DE, we consider the evolutions that are defined by linear interaction functions between the interaction objects. The interaction function is defined as follows:

$$\begin{aligned} k : \mathcal{O} \times \mathcal{O} &\rightarrow \mathcal{O} \\ (s, e) &\mapsto k(s, r) = r \end{aligned}$$

The parameters of the interaction function (s, r) are called respectively the sensor and the effector. So, the k function calculates the result of the interaction of the effector on the sensor.

Having an interaction function k, one can define an operation between the vectors of interaction objects as follows:

$$\begin{aligned} \times_k : \mathcal{O}^{\mathcal{A}} \times \mathcal{O}^{\mathcal{A}} &\rightarrow \mathcal{O}^{\mathcal{A}} \\ (0, y) &\mapsto 0 \times_k y = 0 \\ x \neq 0,\ (x, y) &\mapsto x \times_k y = z : \forall i \in \mathcal{A}, z_i = \sum_{v \in \mathcal{A}} k(x_i, y_v) \end{aligned}$$

The operation $\times_k$ simply builds a new vector by applying the interaction function k to all the elements such that the i-th element of the result is given by the application of the interaction function k to the i-th element of x (the sensor) and the elements of y (the effectors)

The evolution of a MIC* element $t \in \mathcal{T}$ according to an interaction function k within an interaction space $j \in \mathcal{S}$ is defined as follows:

$$\begin{aligned} f_{(k,j)} : \mathcal{T} &\rightarrow \mathcal{T} \\ t &\mapsto t' = f_{(k,j)}(t) \end{aligned}$$

Such that:

$$\begin{aligned} out(t') &= out(t) \\ in(t') &= in(t) + ((out(t))_j \times_k (out(t))_j) \end{aligned}$$

The evolution function $f_{(k,j)}$ is a valid interaction evolution function since the specifications presented in section [3.3.4, page 67] are satisfied. The implementation of MIC* uses the interaction functions as described in the next section. Still, other implementations may use other interaction evolutions of the DE.

3.4 The Specification and Implementation of the MIC* Model

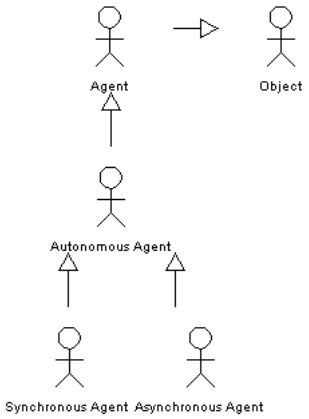


Figure 3.9: The agent classes and their relationships.

The MIC* framework follows the classification of Luck and d'Iverno [Luck & d'Inverno, 1995] stating that the agents can be considered as an extension of objects (cf. Figure 3.9). The autonomous agents are considered as special agents that have a total control of their internal decisional process and may change their behaviour. Two special types of autonomous agents are considered: the asynchronous agents and synchronous agents. The asynchronous agents execute their actions in parallel, while synchronous agents execute their actions sequentially. Ideally, the autonomous agents are considered as asynchronous, but in some fields such as the agent-based simulation (ABS) and artificial life (AL) this is impossible to achieve. In fact, within these fields an important number of agents is used in order to simulate systems; their asynchronous execution would require important computational resources that may be available only on specialised (and often costly) computers. As a DE, the MIC* framework manages the two types of agents differently since the time semantics is not the same for each class of application.

The main functionalities of the MIC* framework are : *(i)* populating the environment with agents, *(ii)* managing the perception/action loop of agents, and *(iii)* managing on-the-fly composition of several MIC* DEs.

3.4.1 Populating the Deployment Environment

As shown in Figure 3.10, the DE should offer some basic services to allow the agents to enter and leave the MAS. A software agent may be present in several DEs and may continue in own activities when leaving a particular DE. The agents interact only when they are defined on the same DE.

Entering in the Deployment Environment:

The software agent sends a login request to enter the MIC* DE (cf. Figure 3.11). The login message contains some confidential information identifying the agent. The DE checks the identity of the incoming agent and acts following two cases:

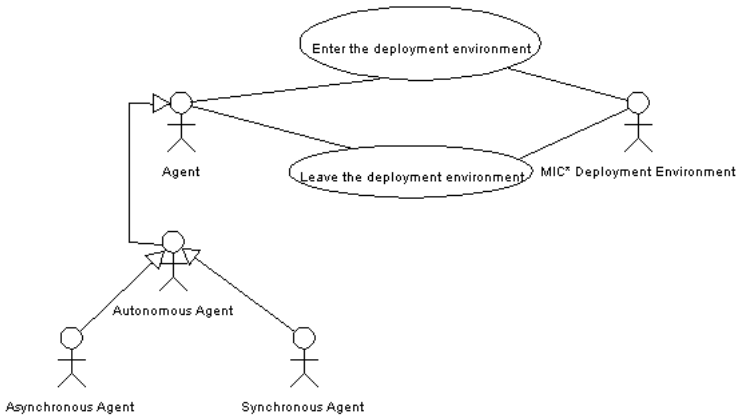


Figure 3.10: Populating and leaving the deployment environment use-case.

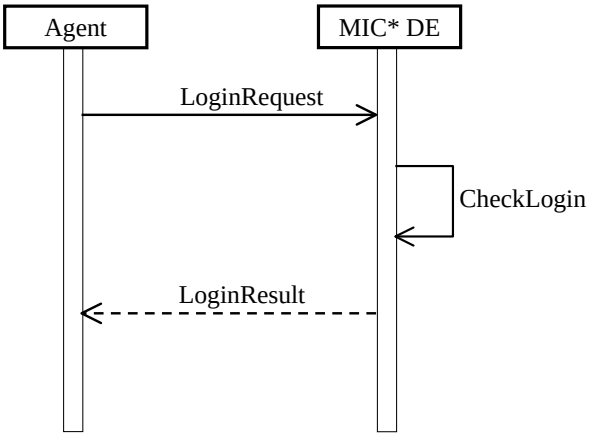


Figure 3.11: The login process of an agent on the deployment environment.

case 1 (successful login): when the login process is successful, the DE creates a locally unique handle for the incoming agent. This handle identifies the agent and should be included in all messages sent by this agent towards the DE.

case 2 (unsuccessful login): when the DE rejects the login request of the software agent, an error message is sent and describes the reason of the failure.

When successful, this operation is similar to adding an agent row to the formal MIC* structure representing the DE.

Leaving the Deployment Environment

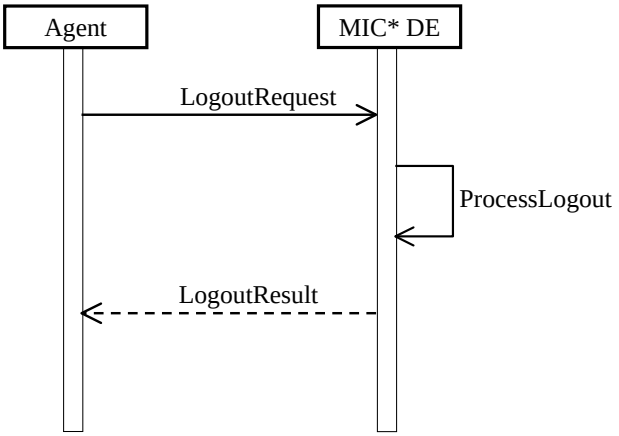


Figure 3.12: The logout process of an agent from the DE.

As presented in Figure 3.12, the software agent leaves the DE by sending a logout message. The DE frees all the retained computational resources and sends back an acknowledgement message. When leaving the DE, the software agent is no longer visible to other agents. This operation is similar to deleting the agent row from the MIC* formal structure representing the DE.

3.4.2 The Perception/Deliberation/Influence Agent's Loop

The software agents interact through the DE by exchanging interaction objects within interaction spaces. Consequently, the DE should offer some services such as (cf. Figure 3.13): *(i)* the notification of a reception of a new interaction object (optional), *(ii)* retrieval of the inbox, *(iii)* and emission of interaction objects in the DE. The perception/deliberation/influence loop is a pattern of the agents' behavior. MIC* defines the inbox and outbox of the agent within the DE. A software agent has to proactively request its inbox (perception); compute internally its reaction (deliberation); and emit new interaction objects in some interaction spaces (influ-

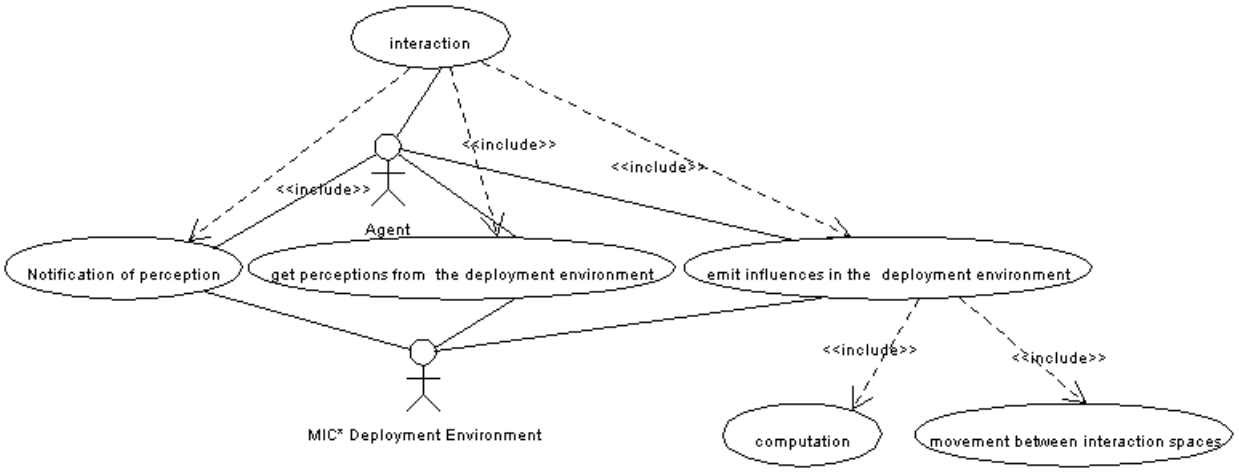


Figure 3.13: The agent interaction use case.

ence). When the software agent is temporary disconnected from the DE, it is still interacting with other agents and may retrieve its inbox whenever possible.

Notification of Interactions

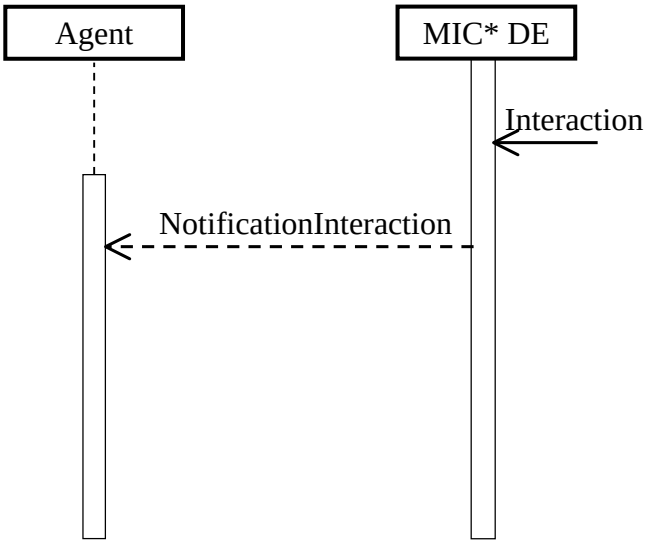


Figure 3.14: The DE sends a notification to an agent when its inbox is modified.

The notification process, described in Figure 3.14, informs the software agent about the modification of its inbox. This mechanism is optional and suitable for asynchronous agents. It is activated only when the asynchronous agents explicitly desire to react immediately to incoming interaction objects. When the agent is not reachable the DE simply ignores this process.

Retrieving the Perceptions from the Deployment Environment

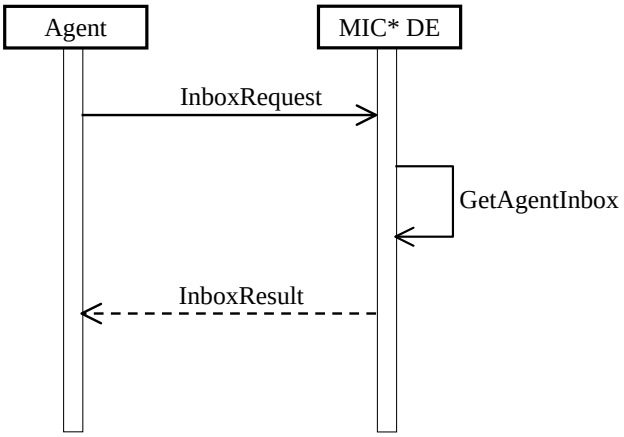


Figure 3.15: The software agent retrieves its inbox from the deployment environment by sending an inbox request message.

The software agent retrieves its inbox by sending a request message as described in Figure 3.15. The inbox is a mapping of ISs to interaction objects. Only the non-empty inboxes are delivered to the agent. The retrieved inboxes are set to the empty interaction object in the DE.

Emitting Interaction Objects in the Deployment Environment

After the deliberation, the software agent emits interaction objects in the DE. This is done by sending a mapping that links the ISs to the new outboxes (cf. Figure 3.16). This mechanism is described by the activity diagram of Figure 3.17.

For asynchronous agents, the computation is immediately followed by an interaction evolution inside each modified interaction space as presented by Figure 3.18. In the synchronous mode, the interaction evolution is performed only when all the required synchronous agents have computed (cf. Figure 3.18). This guarantees that all the synchronous agents perceive and react correctly at the same logical time even if they are executed sequentially [Michel, 2004].

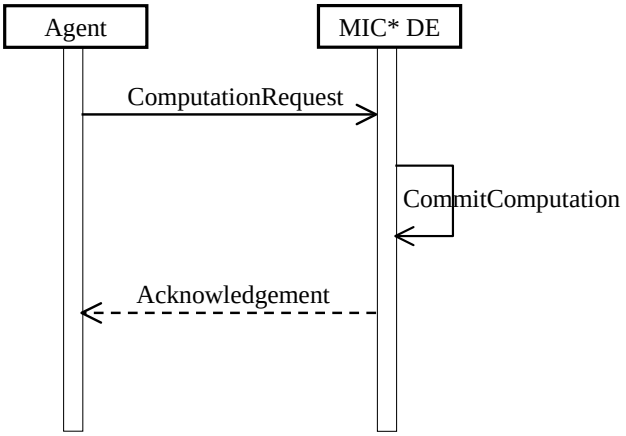


Figure 3.16: The software agent sets its outbox in the deployment environment by sending a computation request.

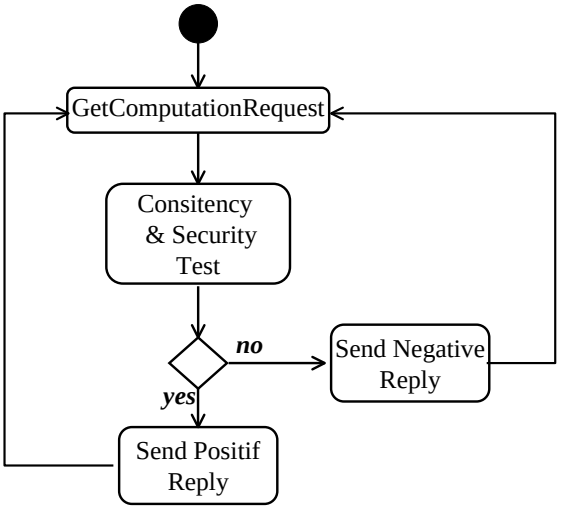


Figure 3.17: The computation activity diagram.

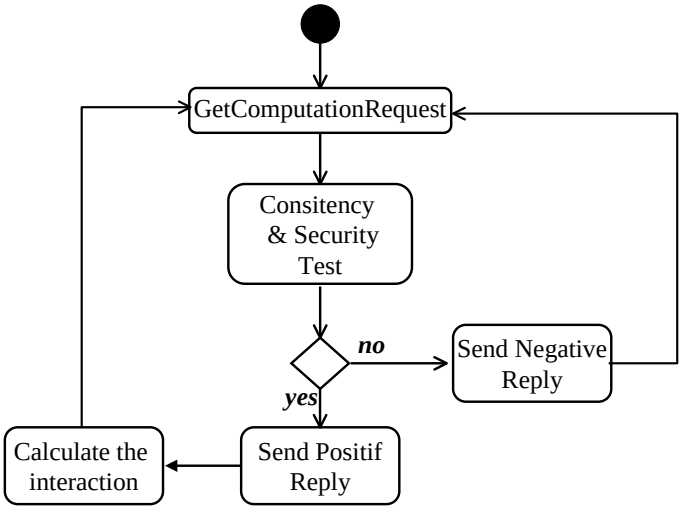


Figure 3.18: The computation-interaction activity diagram for asynchronous agents.

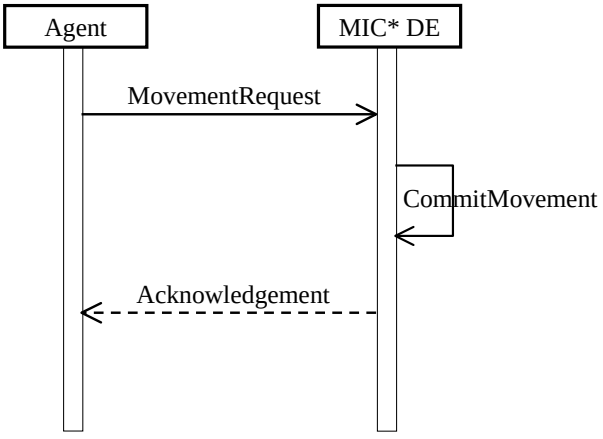


Figure 3.19: The software agent moves between interaction spaces by sending a movement request.

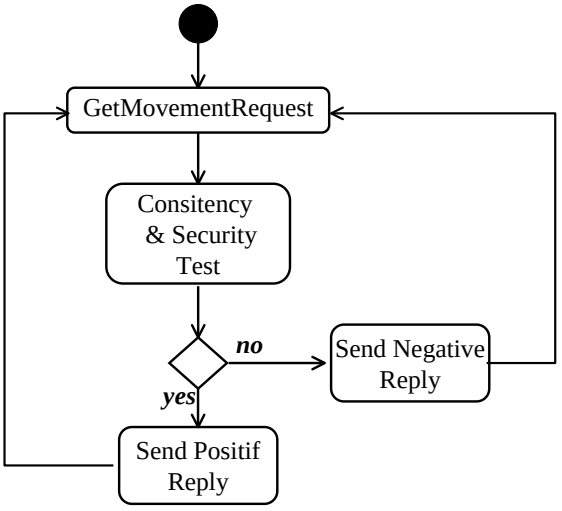


Figure 3.20: The movement activity diagram.

The Movement Among Interaction Spaces

The movement between the ISs is explicitly requested by the software agent by sending a movement request (cf. Figure 3.19). After processing the movement, the DE sends back an acknowledgement to the agent (cf. Figure 3.20).

3.4.3 The Composition of MIC* Deployment Environments

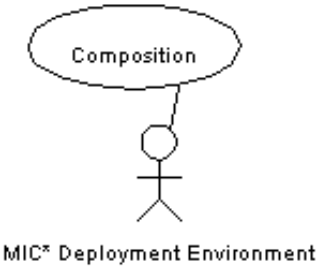


Figure 3.21: On the fly composition of MIC* DEs use case.

Thanks to the algebraic modeling of MIC*, the on-the-fly composition is specified formally and the software structures should implement correctly and efficiently this composition feature.

When a point-to-point communication link can be established among two DEs, their composition is initiated by one of the participants by sending a connect request as described in Figure 3.22. The connect request includes the description of public interaction spaces that may be shared. The receiver of the connect message replies positively when the composition request is agreed and negatively otherwise.

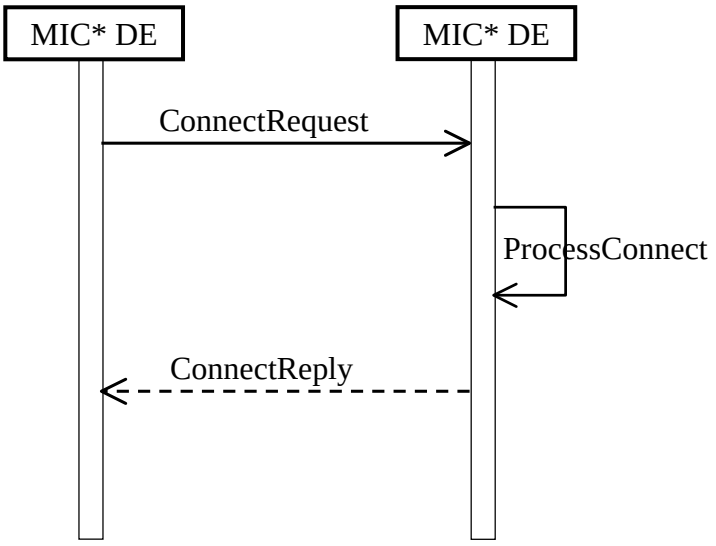


Figure 3.22: The connect activity diagram.

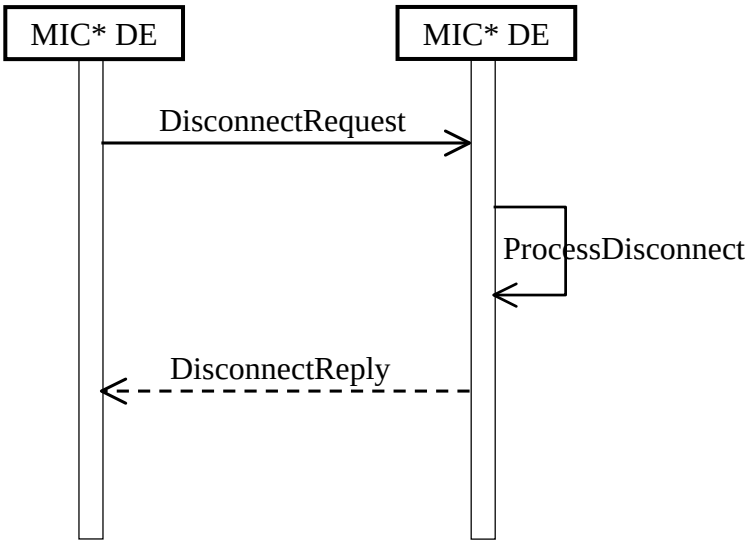


Figure 3.23: The disconnect activity diagram.

The decomposition of the DEs is performed by sending a disconnect request as described in Figure 3.23. The disconnected DEs are independent and autonomous and do not share any IS.

Despite the distribution, the connected DEs behave coherently and emulate a single global DE. However some restrictions at the implementation level have been made in order to implement the composition of the DEs:

- restriction on ISs: the DEs share their own public ISs. Consequently, the composition does not create locally new ISs that were unknown before the composition;
- restriction on the movement of IOs: as a consequence of the previous restriction on ISs, the movement of the IOs is defined only between local ISs. In other words, the movement evolution cannot move an IO from a DE to another.

These restrictions solve some problems related to the resources of the devices running the software system. In fat, the resources of the devices may be restricted and cannot handle new ISs discovered at the run-time. Furthermore, there is not a clear strategy to manage the incoming ISs and IOs when the DEs are no longer composed.

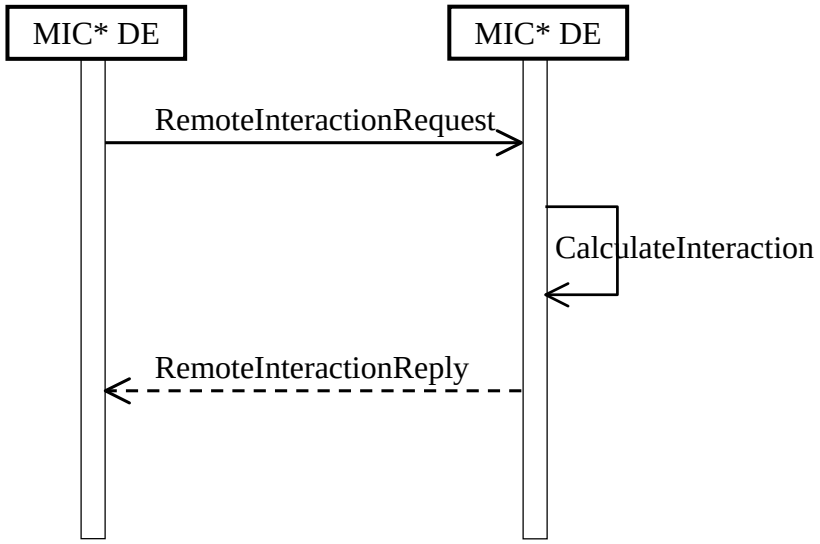


Figure 3.24: The synchronization of the interactions between the composed DEs.

By assuming the presented restrictions which are coherent with the EAC assumptions, the composed DEs have to synchronize only the interaction operations that occur on the shared ISs in order to behave as a global coherent DE. Consequently, when an interaction is calculated for a particular agent, this operation is calculated locally and forwarded to the remote DE along with the value of the source agent's outbox. The result of the interaction of the agent's

outbox with the remote agents is then returned and added to the source agent's inbox. Since, the addition of IOs is commutative, the order of these operations does not influence the final result.

3.4.4 MIC* Packages and Classes

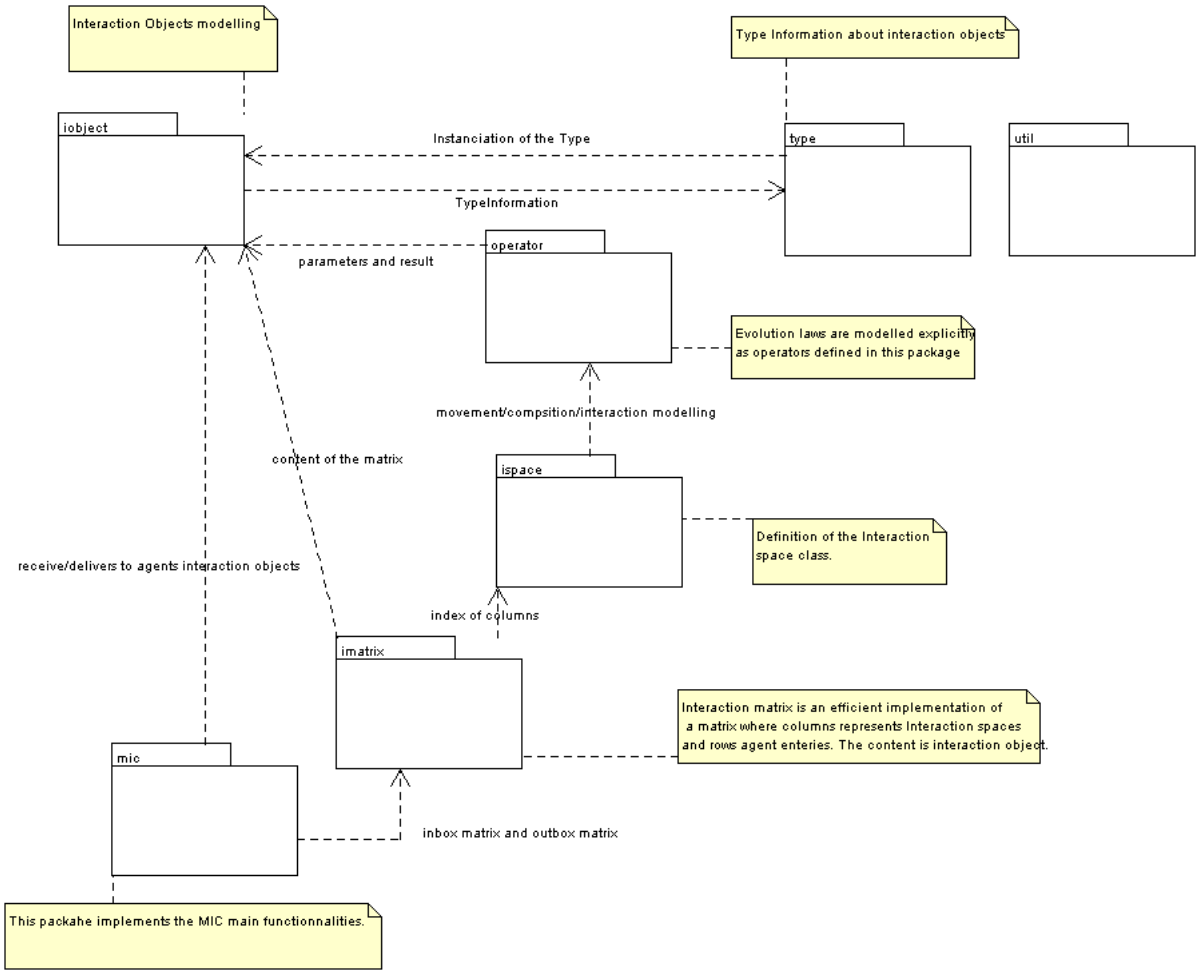


Figure 3.25: The packages of the MIC* implementation.

The MIC* packages are organized as follows (cf. Figure 3.25):

1. `iobject` : this package defines the classes implementing the IO concept.
2. `type` : in order to classify IOs a type mechanism is introduced by this package. Thanks to the type information, the performances are improved significantly. In fact, the operators that are the means to define the dynamics of MIC* uses the type information in order to be executed only when the matching IOs are available.
3. `operator`: this package represents the operators that are defined to implement the dynamics of MIC*. The operators are classified in three categories:
   - interaction category: two kind of operators are included in this category: the interaction operators and composition operators. The interaction operators represent

the interaction functions (cf. §3.3.4, page 68). The interaction operators take a couple of IOs as argument and the returned result is also an IO: $(x, y) \mapsto z$. By convention, x is considered as the sensor, y is considered as the effector, and z as the interaction result. The composition operators are introduced to define some simplification rules among the IOs. In fact, by default MIC* implements the formal composition operator, which builds formal sums, or multisets, of IOs. More specific composition operators may be defined to calculate the result of the composition of IOs. For instance, let us suppose that the integers are used as IOs. By default, the composition of two integers gives a multiset of integers (formal sum): $2 + 3 = ((2), (3))$. One may introduce another composition operator, $+_{\mathbb{Z}}$, to calculate the natural addition among the integers: $2 +_{\mathbb{Z}} 3 = 5$. The actual definition of the composition operators depends on the semantics of each MAS.

- movement category: the movement among ISs is modeled as the composition of two operators: the *movement-out* operator and the *movement-in* operator. The movement-out operator allows IOs to leave an IS. This operator takes as argument an IO, x, and returns an IO y. x is the IO that wants to move outside the IS and y is what actually goes outside the IO. On the other hand, the movement-in operator is an operator that allows IOs entering inside an IS. The movement-in takes as argument an IO and returns an IO: $x \mapsto y$. x is what wants to enter inside the IS and y is what actually enters inside the IS.
- control category: this category of operators allows the ISs to control the IOs that are sent by the agents. For instance, the *filter operator* gives the IS the ability to modify an IO sent by an agent. This operator takes as argument an IO and returns another IO: $x \mapsto y$. x is as the original IO and y is the alteration of x by the IS.

4. `ispace`: this package holds the classes that implement ISs. An IS holds the agents' IOs. The ISs define also some operators. For instance, both the interaction and composition operators define the interaction among the IOs. Similarly, the movement-out and movement-in operators describe the movements of IOs between the ISs. Finally, the filter operator gives the IS the ability to control the IOs of the agents.

5. `imatrix`: This package defines the matrices of MIC*.

6. `mic`: This package defines a single class representing the MIC* DE.

3.4.5 Classes Description

`iobject` package:

This package defines four classes (cf. Figure 3.26) that are described as follows:

1. `InteractionObject`: this is the base class of all IOs subclasses. This class is abstract and consequently cannot be instantiated.

2. `AtomicInteractionObject`: the atomic IO represents a single IO. The atomic IO is organized as a container of named and typed fields. The types that are supported by the current implementation of MIC* are:

   integer: representing the set of integers;

   string : representing the set of sequence of characters;

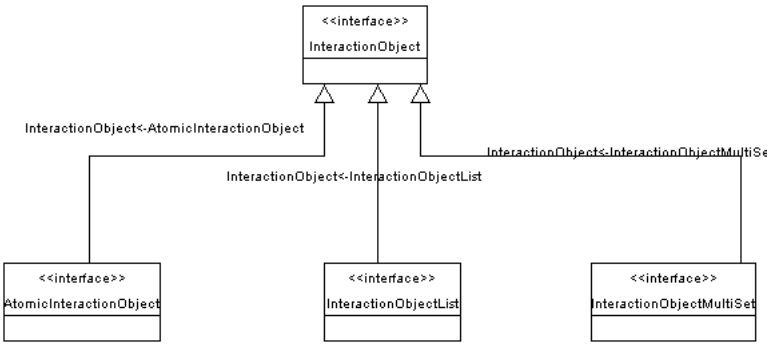


Figure 3.26: The IOs hierarchy of classes.

boolean: representing binary logical values;

byte: representing a 8-bits data structure;

float: representing real numbers;

The arrays, that represent ordered sequences of values, are also supported for each of the described simple types. Each atomic IO is associated with a type that describes its structure.

3. `InteractionObjectMultiSet`: this class represents the composition of the atomic IOs under the + law. The elements of the multi-set are unordered and may be duplicated. The empty multi-sets and the multi-sets containing only empty atomic interaction objects are considered equivalent to the zero 0.

4. `InteractionObjectList`: this class represents ordered list of atomic IOs. The empty lists and lists containing only empty atomic interaction objects are considered equivalent to the zero 0.

Package `type`

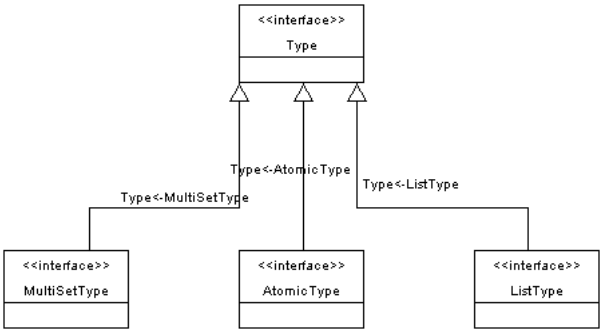


Figure 3.27: The type package hierarchy of classes.

The type is used in order to describe the internal structure of the IOs and to classify them. The type package includes four classes (cf. Figure 3.27):

1. `Type`: this is the base class of all the other classes included in this package.

2. `AtomicType`: this class represents the type of an atomic IO. The atomic type can be considered as the template from which the atomic IOs are instantiated. The atomic type can inherit from different atomic types. All the fields of the super-types are defined within the subtypes.

3. `MultiSetType`: this class represents the type of an `InteractionObjectMultiSet` as a multi-set of `AtomicType`. Hence, the atomic types may be duplicated in the multi-set and the elements are not ordered.

4. `ListType`: this class represents the type of an `InteractionObjectList` as an ordered sequence of `AtomicType`. Hence, the atomic types may be duplicated in the sequence and are ordered.

Package `operator`**:**

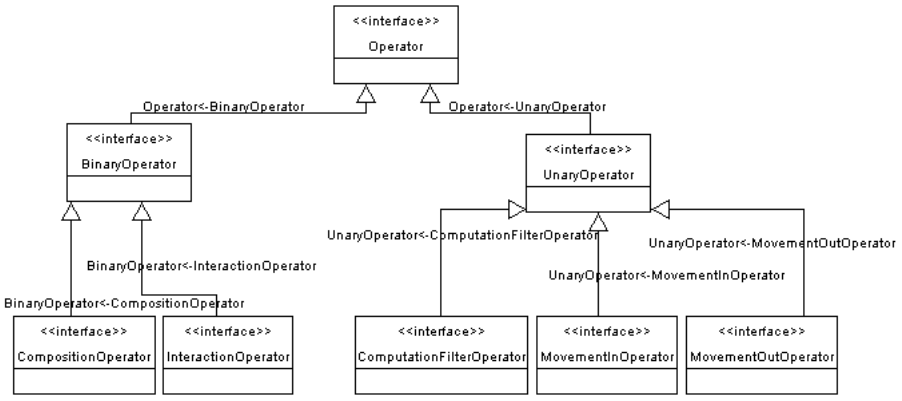


Figure 3.28: The class hierarchy of the operator package.

1. class `Operator`: the operator is a function taking its arguments from a *domain* and returning a result defined on a *co-domain*.

2. class `UnaryOperator`: this class is a specialization of the `Operator` class and represents the functions that take only one argument. This class is specialized by `MovementOut-Operator`, `MovementInOperator` and `FilterOperator` classes.

3. class `MovementOutOperator`: this class represents the functions that move atomic IOs outside ISs.

4. class `MovementInOperator`: this class represents the functions that move atomic IOs inside ISs.

5. class `FilterOperator`: this class represents the functions that alter the atomic IOs emitted by the agents.

6. class `BinaryOperator`: this class is a specialization of the `Operator` class and represents the functions that take two arguments. This class is specialized by the `Composition-Operator` and `InteractionOperator`.

7. class `CompositionOperator`: the atomic IOs may be composed to produce another atomic IO.

8. class `InteractionOperator`: This class represents the base-class of all the interaction operators.

`ispace` **package:**

- class `InteractionSpace`: this class allows the creation of new interaction spaces. Each interaction space is identifiable by a unique identifier, *UID*, represented as sequence of characters. Two interaction spaces with the same UID are supposed to be equivalent. This means that they are defined with the same set of operators.

`imatrix` **package:**

- class `InteractionMatrix`: this class implements efficiently the matrix representation of MIC*. The elements of the interaction matrix are `InteractionObjectMultiSet` instances.

`mic` **package:**

- class `MIC`: This class represents the DE. The DE holds two `InteractionMatrix` instances representing the `outboxMatrix` and `inboxMatrix`.

The implementation of the MIC* model is available as an open source project at sourceforge. The source code and binaries of MIC* are freely available at these pages: mic.sourceforge.net and sourceforge.net/projects/mic.

3.5 Building a Social Framework upon MIC*

MIC* only offers a generic and low level abstractions of a DE. To build real world applications, one has to provide a higher-level engineering framework. This section presents a social framework. The idea is that MAS designers only deal with social concepts which are automatically translated to MIC* concepts.

3.5.1 Presentation of the Social Framework

The presented social framework is deeply inspired by the AGR model [Ferber & Gutknecht, 1998]. The MadKit [Gutknecht & Ferber, 2000a] platform already implements the AGR model; here we explore another implementation using only MIC* primitives.

The social abstractions presented by AGR are briefly described as follows:

- (A)gent: an agent may play one or several roles and may be member of one or several groups;
- (G)roup: a group is a collection of roles and consequently a collection of agents that play these roles. The interaction among the agents can occur only when they are located within the same group;

- (R)ole: a role is an abstraction that represents a function or a service within the society; agents playing the role fulfill the desired service.

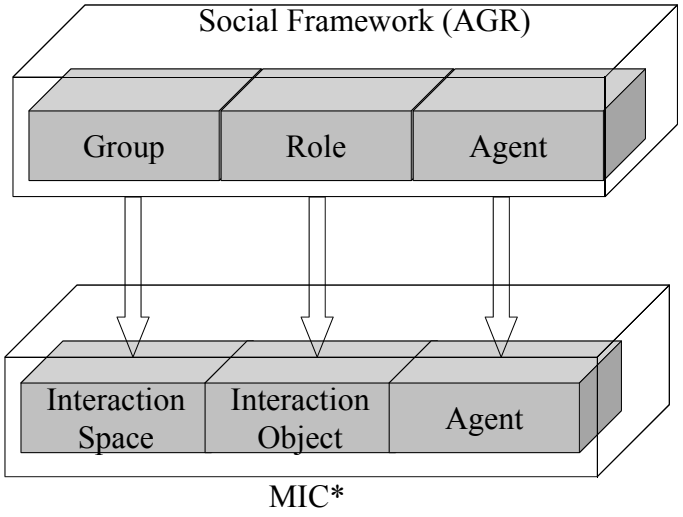


Figure 3.29: Mapping between AGR concepts and MIC* concepts.

At this stage, let us sketch a preliminary mapping between AGR and MIC*. As shown in Figure 3.29, the group concept may be modeled as an IS: the IS concept may be seen as a logical location where a collection of agents interact. Besides, the agents may move across groups; this is similar to moving across ISs. The agent concept of AGR naturally corresponds to MIC* agents. Still, there is not a one to one mapping between these concepts. The role concept is considered as an IO within MIC*. In fact, when an agent plays a certain role, it publishes an IO that describes itself as playing this role. Consequently, other agents can identify its social function and interact with it. The implementation of the AGR model using MIC* is explained in more detail in Table 3.2.

Two interaction schemes are considered for the social framework:

1. The role-level interaction schema: the messages are delivered to the agents only by knowing their roles. This mechanism allows implementing one-to-many communications and the discovery of agents' identities by knowing only their roles.

2. The agent-level interaction schema: the messages are delivered to the agents by knowing their exact identity. This mechanism implements one-to-one communications.

3.5.2 Implementation of the Social Framework

Interaction Objects

Figure 3.30 presents the types of IOs used in the social framework:

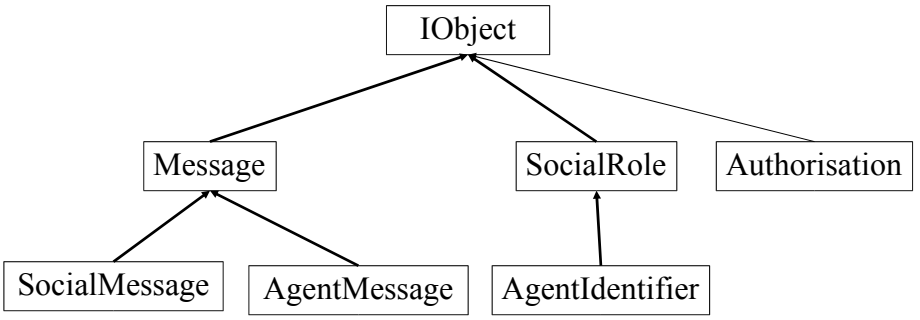


Figure 3.30: Type hierarchy of IOs used in the social framework.

- `Message`: the `Message` type represents IOs used to exchange information encoded as a content. This is the base-type of all other interaction related types; it contains a single field, `content`, that represents the exchanged information.
- `SocialMessage`: the `SocialMessage` type represents exchanged messages for the role-level interaction schema. The fields of this type are: `sender-role` that represents the role of the sender and `receiver-role` that represents the role of the receiver. This type inherits the `content` field from the `Message` type.
- `AgentMessage`: the `AgentMessage` type represents exchanged messages at the agent-level interaction schema. This type fields are: `sender-agent-id` that represents the identity of the sender; `receiver-agent-id` that represents the identity of the receiver; `sender-role` that represents the role of the sender; `receiver-role` that represents the role of the receiver. This type inherits the `content` field from the `Message` type.
- `SocialRole`: the `SocialRole` type represents roles which are played by the agents. This type defines only a single field `role-id` that represents the unique identifier of the role.
- `AgentIdentifier`: the `AgentIdentifier` type represents the identity of an agent. In fact, since agents do not have access to the structure of others, they have to explicitly publish their identity. This type defines only a single field `agent-id` that represents the unique identifier of the agent; it also inherits the `role-id` field from the `SocialRole` type.
- `Authorisation`: the `Authorisation` type is used to control group access using movement operators. An agent is allowed to enter an IS by presenting the correct `Authorisation` instance. The `Authorisation` contains the name of the played role, namely the

`played-role` field; and the `certificate` field that represents a signature confirming that the agent is allowed to play this role.

Interaction Operators

Two interaction operators are defined in order to model the interaction schemes:

- Role-level interaction operator: this operator is defined among `SocialRole` and `SocialMessage`. A `SocialRole` interacts with a `SocialMessage` only and only if the receiver role of the `SocialMessage` is the same as the role-id field of the `SocialRole`. This is expressed algorithmically as:

```
1: function RoleLevelIOP::interaction(sensor,effector)
Require: sensor is instance of the SocialRole type
Require: effector is instance of the SocialMessage type
2:   if sensor['role-id'] == effector['receiver-role'] then
3:     return effector
4:   else
5:     return 0                                   ▷ No interaction.
6:   end if
7: end function
```

- Agent-level Interaction Operator: the agent-level interaction is defined between `AgentIdentifier` and `AgentMessage`. An `AgentIdentifier` interacts with an `AgentMessage` only and only if the id of the receiver is the same as the id of the agent. This is expressed algorithmically as follows:

```
1: function AgentLevelIOP::interaction(sensor,effector)
Require: sensor is instance of the AgentIdentifier type
Require: effector is instance of the AgentMessage type
2:   if sensor['agent-id'] == effector['receiver-agent-id'] then
3:     return effector
4:   else
5:     return 0                                   ▷ No interaction.
6:   end if
7: end function
```

Movement Operators

The groups are modeled as ISs. Consequently, each IS is associated with a set of roles. To enter the IS, an agent has to play a role that belongs to this set. To realize these movements, the group-entrance operator allows agents to enter inside an IS. The agents have to present an `Authorisation` that describes the played role. On the other hand, the group-leaving operator allows agents to leave the IS.

Interaction Spaces

Besides the default IS defined by MIC*, each group is represented by an extension of MIC* IS, namely the *social interaction space*. Each social IS is defined with a set of authorized roles; the role-level and agent-level interaction operators; and the group-entrance and group-leaving operators.

The Autonomous Agents

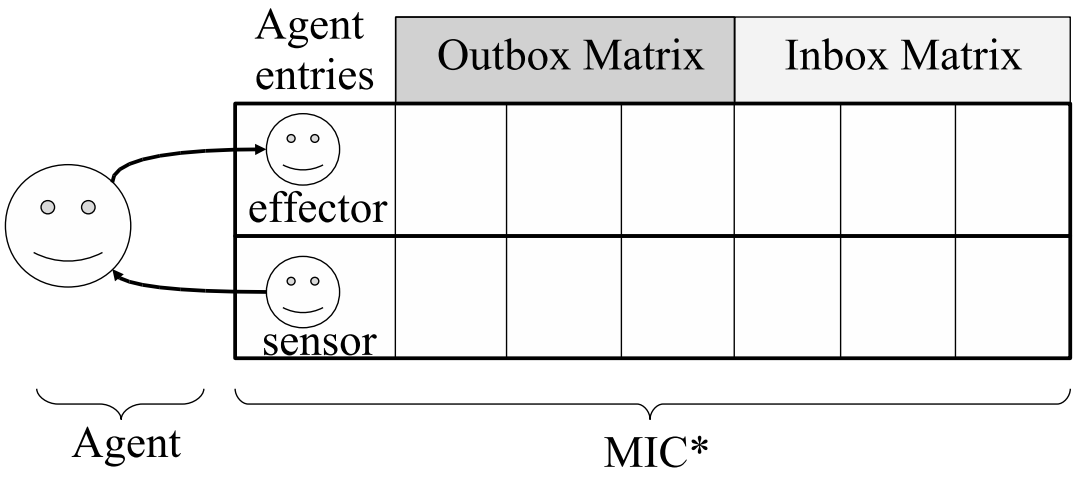


Figure 3.31: A single agent have several entries within the MIC* deployment environment: an entry dedicated to sense the universe, i.e. the sensor entry; and an entry dedicated to affect the universe, i.e. the effector entry.

Within MIC*, the agents may have simultaneous activities. For instance, a single agent can sense its surrounding environment and affect it simultaneously. To realize this simultaneity, several MIC* agent entries are used. For instance, Figure 3.31 shows this schema where two MIC* agent entries are associated to a single agent: the *sensor* entry and *effector* entry.

From the agent perspective, the sensor entry is dedicated for sensing the universe. Consequently, IOs that perceive the universe are placed in the outbox matrix, and the result of their interaction is placed in the inbox matrix. On the other hand, the effector entry is dedicated to affect the universe; consequently, IOs to be perceived by other agents are placed in the outbox matrix and, in this case, the inbox matrix is not used (marked with X in Figure 3.32). For the MIC* environment, the agent's sensor and effector entries are considered as independent agents; the agent is responsible for making this couple of agents behaving as a single entity. This seems similar to the Holonic approach that considers a set of agents as an single agent [Parunak & Odell, 2001]; still, here we argue that a set of agents that have been conceived to behave as a single entity can build a global agent.

To represent the fact that an agent plays several roles in groups, the mechanism presented above is extended such that each agent is associated to an effector entry and zero or more sensor entries. Each sensor entry represents a played role. Figure 3.32 gives an example of an agent that plays three roles R_1, R_2 and R_3. This agent has a single effector to send messages, for instance this agent is sending three messages simultaneously m_1, m_2 and m_3 in three groups G_1, G_2 and G_3. This agent plays simultaneously several roles within the same

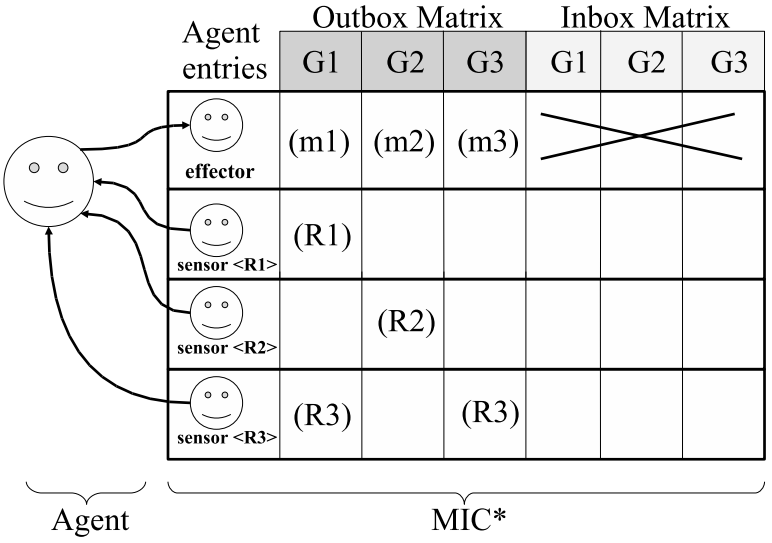


Figure 3.32: A social autonomous agent owns an effector entry to affect the universe, and several sensor entries representing its roles.

group: R_1 and R_3 in the group G_1; and plays the same role in several groups: R_3 in G_1 and G_3.

3.5.3 Mapping Table between AGR and MIC*

Finally, by considering the presented concepts, the complete mapping among the AGR concepts and MIC* is described by Table 3.2

AGR	MIC*
Creation of a group named *x*	This is performed by creating an IS identified as *x*
Deletion of the group *x*	This is performed by deleting the IS identified as *x*
The agent *a* joins the group *x*	The agent *a*'s entries (effector and sensors) enter the IS *x* using the *Authorisation* IO. The involved movement operator is the group-entrance operator.
The agent *a* leaves the group *x*	The agent *a*'s entries (effector and sensors) leave the IS *x* using the group-leaving movement operator.
The agent *a* acquires the role *r*	The agent *a* acquires the *Authorisation* IO with a valid certificate.
The agent *a* plays the role *r* within the group *g*	The agent *a* moves inside the IS *g* using the *Authorisation* IO; when this agent is inside the IS, he changes its outbox to the *SocialRole* IO; now this agent is perceived by the others as playing the role *r*.
The agent *a* stops from playing the role *r* within the group *g* for a short period	The agent *a* has only to change how the others perceive him; so it changes its *SocialRole* IO to another one; when the agent wants to resume playing the role *r*, he puts back its *SocialRole* IO.
The agent *a* drops the role *r*	The agent *a* leaves all ISs where he was perceived as playing the role *r*.
The agent a_1 playing the role r_1 sends a message to the agent a_2 playing the role r_2 within the group *g*	The message to be sent is an instance of *AgentMessage* type. The agent a_1 puts this message as a computation of its effector in the *g* IS; then the interaction operator performs the actual interaction among the agent a_1 effector outbox and the agent a_2 sensor corresponding to the role r_2. The result of the interaction is found in the inbox of the agent a_2. In this case, the agent-level interaction operator is used.
The agent a_1 playing the role r_1 sends a message to any agent playing the role r_2 inside the group *g*	The message to be sent is an instance of *SocialMessage* type. The agent a_1 puts this message as a computation of its effector in the *g* IS; then the interaction operator performs the actual interaction among the agent a_1 effector outbox and all sensors corresponding to the role r_2. In this case, the role-level interaction operator is used.

Table 3.2: Mapping between AGR concepts and commands to MIC*.

3.6 Conclusion

This chapter has emphasizes on the important role played by the DE within MASs at the implementation level. In fact, the DE guarantees the autonomy of the agents while it defines their interactions. As an example of such DE, MIC* has been proposed.

The notion of DE clearly separates the concerns of MAS engineering. In fact, the engineering of agents is completely separated from the engineering of DEs (SCMAS approach). The DE is the common structure offered to different developers and authorities to deploy autonomous agents and to make a global system which functions emerge from the interactions of the autonomous individuals.

MIC* offers some interesting features such as the implementation of the internal integrity for the agents; the generative interaction and the on-the-fly composition. These features have provided the basis for engineering open software systems in complex and unpredictable environments such as the EAC context.

The MIC* model has been implemented as a development library to help developers to design and implement DEs based on MIC* principles. Besides, an additional library has also been implemented to offer agents' developers an easy way to implement autonomous agents and to connect them to MIC* DEs.

The source code and binaries of these libraries are freely available at these pages: mic.sourceforge.net and sourceforge.net/projects/mic.

By experimenting the engineering of software systems using the MIC* model, we have noticed that the designers and developers have some difficulties in order to abstract their systems directly using MIC*. In order to hide the apparent complexity of the MIC* model, we have proposed to build higher-level layers that are translated automatically to MIC* concepts. The social framework is an example of this simplification effort. This framework is extended and developed in chapter 5 to offer an entire integrated development environment (IDE) to build software systems for the EAC context.

To meet the requirements of the EAC context, MIC* has to provide more elaborated control and trust functions. Chapter 4 presents a coordination framework that is integrated to MIC* in order to monitor the coordination protocols by only observing the conversations of the agents. Hence, the agents that do not follow the coordination rules are immediately identified as responsible for breaking the MAS consistency.

3.6.1 Acknowledgement

The initial reflections and algebraic models of the MIC* structure have been conducted in conjunction with Dr.Yves Guiraud from the Laboratoire Géometrie, Topologie, Algèbre, Université Montpellier 2. The formal structure of MIC* that has been presented in this chapter is a personal improvement of the initial models.

Chapter 4

Coordinating the Autonomous Agents

4.1 Introduction

The interoperability of open and autonomous systems is one of the fundamental features of the EAC (cf. §2.3.3, page 30). In fact, the potential innovation of the EAC depends on the ability of open and autonomous systems to interact in order to make emerge a new generation of services and functionalities.

The interoperability does not concern only the sharing of information but concerns also the execution of the joint activities that are distributed among autonomous systems. In fact, the solutions that are presented nowadays in order to share knowledge and information are incomplete if the context of the information exchange is not specified. For instance, the use of a particular formalism, like RDF or XML, to represent information is not sufficient if some important questions are not answered like the following: what communication protocol to use in order to exchange the content of the messages? when to send the messages? how to handle the errors and exceptions?

These questions are not related to the representation of the information, but to the activity of exchanging the information that is considered as a joint activity between the interlocutors.

This chapter presents our proposition for the interoperability of the autonomous systems within the EAC context. Our proposition suggests tackling the interoperability from the coordination point of view.

Etymologically, the word 'coordinate' is composed of two main words: *(i)* 'co', which is used to indicate that something is common and shared between different entities; and *(ii) ordinate* which indicates the order of things. So, the coordination approach for the interoperability suggests making a common and shared order of things between the entities in order to achieve consistently a joint activity in which each participating entity tries to achieve its own goals.

From this general etymological analysis, we can already present informally our approach for the interoperability. We suggest to make the autonomous systems sharing a common and predefined order of the joint activities. This is known as establishing a *coordination protocol*. Besides, we suggest that a special entity checks the consistency of the ongoing joint-activities according to the specified coordination protocol. This entity is not allowed to challenge the

autonomy of the software entities. Consequently, it has to adopt an external observer posture without interfering with the decisional processes and behaviors of the software entities.

The assumptions of the work presented in this chapter are as follows:

1. the considered system is a software system composed of either software or hardware autonomous and interacting entities.
2. the communication among the software entities is represented as an explicit exchange of data, or messages, through a communication medium.
3. the communication medium is assumed to have a FIFO (first in first out) model of communication. This means that the messages emitted by an entity arrive with the same order to destination. However, there is not a global order constraint on messages that are emitted by two different entities (see the next point)
4. the communications and actions of the entities are not assumed to be synchronous. In fact, several entities may be able to act at the same time and the communication medium may not guarantee the order of messages that are emitted by different entities. These are similar to the WAN assumptions as stated by Luca Cardelli [Cardelli, 1999].
5. the autonomy implies that the internal structure of the entities is never accessible nor modifiable by external entities (cf. §2.3.2, page 28). The autonomy prevents also from knowing how the software entities actually behave and achieve their goals. Only the observable aspects of their computation are studied (SCMAS approach as presented in section [2.5.3, page 43])

4.2 Backgrounds

Several works have addressed the coordination. To simplify the presentation of these works four non-exclusive and non-exhaustive categories are sketched:

1. generalization efforts: the works of this category develop a general theory to unify all the aspects of the coordination in a single framework. Malone and Crowston's work presented in section [4.2.1, page 93] is an example of this category.
2. activity centric: the works of this category are focused on the 'activity' aspects of the coordination. The leading question is to know what activity to execute and when executing it. This category is represented by the models and formalisms presented in section [4.2.2, page 94].
3. implementation centric: the works of this category offer middlewares that implement some generic coordination primitives. So, these primitives are implemented only once and reused as patterns of coordination in the final applications. This category is represented by the works presented in section [4.2.3, page 95].
4. conversation centric: the works of this category are focused on the conversational aspect of the coordination. The leading question is to know what message to send and when to send it in order to coordinate the joint activities. This category is represented by the works presented in section [4.2.4, page 96].

4.2.1 Generalized Study of the Coordination

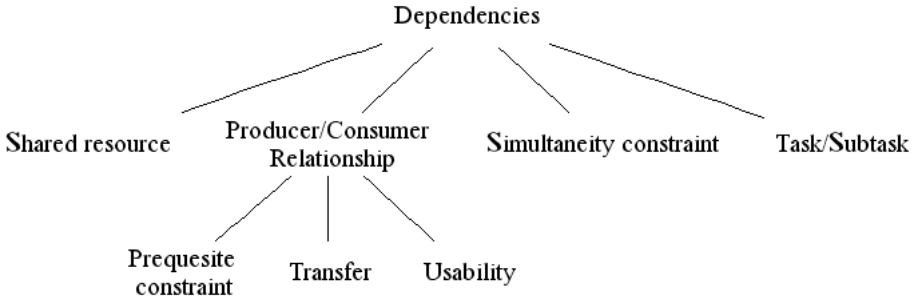


Figure 4.1: Malone and Crowston categorization of the dependencies.

Malone and Crowston in [Malone & Crowston, 1994] have noticed that the coordination has been addressed by several research fields such as informatics, economics, operational research and organizational theory. All these fields have developed some similar ideas about coordination. Starting from this fact, Malone and Crowston tried to develop a single unified framework called the *coordination theory*. The coordination is then defined as managing the *dependencies* between *activities*. In other words, the coordination offers the means to solve the dependencies that inherently exist between different activities. To understand what is coordination, one has to understand what are the dependencies that may exist among the activities. Figure 4.1 shows the categorization of the dependencies proposed by Malone and Crowston:

- shared resource: when several activities share some limited resources, a particular *resource allocation* strategy is needed in order to manage the conflicts. For instance, to share the same execution unit between different processes, a particular scheduling strategy is adopted by the operating system.
- producer/consumer relationship: this dependency represents the situations when an activity produces something, let us call it a *resource*, that is used by another activity. This relation implies necessarily the following sub-dependencies:
  - prerequisite constraint: this dependency expresses the order that inherently exists between the producing and the consuming activities. In fact, the consuming activity cannot start until the producing activity has completed.
  - transfer: the produced resource has to be transferred from the production point to the consumption point. This dependency defines how the resource is *transported* between the production and consumption points. For instance in informatics, when the produced resources are information, the transfer dependency is resolved using the communication protocols as a transportation mean.
  - usability: the third dependency implied by the producer/consumer relationship is the usability. In fact, the consumer should be able to correctly use the produced resource. *Standardization* is a common way to solve this dependency. Hence, both of the producer and the consumer agree on what are the requirements that make the resource consumable. For instance, the syntax of messages and the semantics of the commands are examples of standardization when transferring data between software entities.

- simultaneity constraint: some activities are timely constrained, so they have to be executed at the same time (or cannot occur at the same time). This dependency expresses these temporal situations.
- task/subtask: this relation expresses the hierarchical decomposition of an activity in several sub-activities. So, the main activity depends on its sub-activities and finishes only when the sub-tasks are completed.

4.2.2 Formalisms to Express the Coordination Protocols

Several models and formalisms have been proposed to express coordination protocols. This section gives an overview of these models:

- Software Engineering Methodologies: usually the software engineering methodologies such as MERISE [Kettani *et al.* , 1998] and UML [OMG, 2003] include some graphical and textual formalisms to specify the coordination protocols among the activities that are executed by either concurrent or sequential programs. Generally, these methodologies do not provide their proper formalisms and models, but reuse other known formalisms such as the State Transition Diagrams, Flowchart, and Statecharts.
- State Transition Diagram: The state transition diagrams are simple finite state transition automata that specify the states of a system and the transitions among these states. The transitions are enabled by either external events, such as the reception of a message, or tasks completion.
- Statecharts: the Statecharts formalism [Harel, 1987] is an extension of state transition diagrams with three elements dealing with the hierarchy, concurrency and communication of the automata. This formalism has been used for the specification and design of discrete-event systems such as distributed software systems and communication protocols.
- Flowchart: the Flowchart shows the overall structure of the software system by tracing the flow of information, activities and by highlighting the key processing and decisional points. Besides, the physical media on which the data are inputted, outputted and stored are represented explicitly.
- Process Algebra: this family of formal models (CSP [Hoare, 1985], CCS [Milner, 1989], Pi calculus [Robin *et al.* , 1992], Ambient [Cardelli, 1999], Join calculus [Fournet, 1998]) implicitly expresses the coordination among the concurrent activities by synchronising the communications among the processes. In fact, the order between the activities is implicitly defined by the semantics of the algebraic operators and rewriting rules. Consequently, the observed behavior is still an ordered execution of the concurrent tasks.
- Language of Temporal Ordering Specifications (LOTOS): this formalism is standardised by the International Organization of Standardization (ISO) under the reference `ISO/IEC8807`. LOTOS has been used to develop the Open Systems Interconnections (OSI) specifications. In fact, the OSI specifications have to be understood and implemented correctly by all the different software engineering actors around the world. The LOTOS formalism is based on process algebra to express the implicit order and dependencies among the activities. LOTOS allows also the definition of abstract data type

using the `Act One` formalism. The `Act One` formalism can be viewed as a way to solve the usability dependency presented by Malone and Crowston.

- Petri Nets: this is a formal and graphical oriented formalism for the design, specification, simulation and verification of concurrent systems. This formalism is presented in more detail in Annex B and section [4.6.1, page 109].
- Specification and Description Language (SDL): the SDL is standardized by the International Telecommunication Union (ITU). The SDL coordination protocols are expressed as extended finite state automata; they include also abstract data types described using `Act One`.
- Extended State Transition language (ESTELLE): this is a formal description technique standardized by the ISO. It is suitable for the specification of coordination protocols among distributed software systems and was used to describe OSI services and protocols. The ESTELLE specification is a hierarchy of communicating non-deterministic state automata. By specifying the type of the communication, synchronous or asynchronous, both of the dependencies and execution order are implicitly defined between the automata in ESTELLE.
- Message Sequence Charts (MSCs): The MSCs [ITU, 1993] are graphical and textual formalisms normalized by the ITU for the description and specification of interaction protocols between different components. They are usually included within the SDL specifications or used as sequence diagrams within the UML specifications. The MSCs denotational semantics is expressed using an adaptation of the process algebras. This helps in defining the partial order relations between the events occurring within the MSC.
- High-Level Message Sequence Charts (HMSC): the HMSC is used to compose several MSCs automata using sequence, parallel and alternative operators. These compositions are given two different semantics within the HMSC: strong composition, where all the events of a MSC must hold before executing the next MSC; and weak composition where the events are executed whenever possible. The semantics of the HMSC is an extension of the partial order defied by the MSCs and depends on the composition operators that are used.

4.2.3 Architectures and Middlewares for Coordination

The (software systems) coordination community[1] has also defined some coordination architectures implementing the generic coordination primitives such as time barriers (or 'rendez-vous') and the producer/consumer relationships. The entities of the coordination architecture are: the *coordinables*, representing entities that coordinate their actions; the *coordination medium*, representing the medium used in order to coordinate the coordinables; and finally the coordination laws that define how the coordination media reacts in response to the coordinables' actions. A survey of the coordination architectures can be found in [Papadopoulos, 2000]; it categorizes the coordination architectures in two main categories:

[1]The main conference of this community is the 'International Conference on Coordination Models and Languages', http://music.dsi.unifi.it/coordination.

- data driven: within the data-driven coordination models, the coordination medium is represented as an addressable storage space shared between the coordinables. The coordinables interact by storing and retrieving data structures from the shared data space. The coordination laws define how the data structures are represented, stored and consumed by the coordinables.
- control driven: within the control-driven coordination models, the coordination medium is represented as a set of input/output communication ports that link the coordinables. The coordination medium considers the coordinables as blackboxes and checks the events occurring on their communication ports. These events are then propagated to other coordinables following some specific coordination rules.

4.2.4 The Conversational Aspect of the Coordination

When the coordinables are distributed, the coordination necessarily implies communication. So, the entities communicate by exchanging messages in order to resolve the dependencies between their activities. When observing the exchanged messages among coordinables, an external observer notices that the structure of the conversation – defined as an ordered sequence of messages – is steered by the underlying coordination protocol. Specifying this dialogue structure is known as establishing a *conversation protocol*. Among the formalisms that are used to specify conversation protocols, one can cite the followings: finite state machines (FSM), as in works of [Barbuceanu & Fox, 1995][Barbuceanu & Lo, 2000][Martial, 1992]; state transition diagrams (STD), as in works of [König, 2003]; colored Petri Nets (CPN), as in works of [Cost *et al.* , 2000][Fallah-Seghrouchni *et al.* , 1999].

4.2.5 Discussion

By observing the literature, it seems that the activity centric, conversational centric and architecture centric works constitute blocks with some synergy among them. Still, there is no established formal link between these categories and especially between the activity and conversational aspects of the coordination. Malone and Crowston coordination theory is a first step towards establishing such a unification. This chapter starts from their work in order to define the Dependency-Based Coordination Model (DBCM). The DBCM includes some concepts such as the *roles* and *roles-cut* in order to fit the MAS paradigm. The conversation protocols are then defined formally as a sequence of messages that respect the constraint of order imposed by the coordination protocol.

4.3 Intuitive Introduction to the Dependency-Based Coordination Model

The main goal of the coordination is to make autonomous software systems working together in order to achieve some joint activities. We suggest that the autonomous software systems share explicitly a common specification of joint activities. These specifications are expressed as *coordination protocols*.

For instance in Figure 4.2, agents 'A' and 'B' agree to conform a specific coordination protocol in order to build a house. At the execution time, each agent is expected to conform its commitments and respects the order of the tasks described by the coordination protocol.

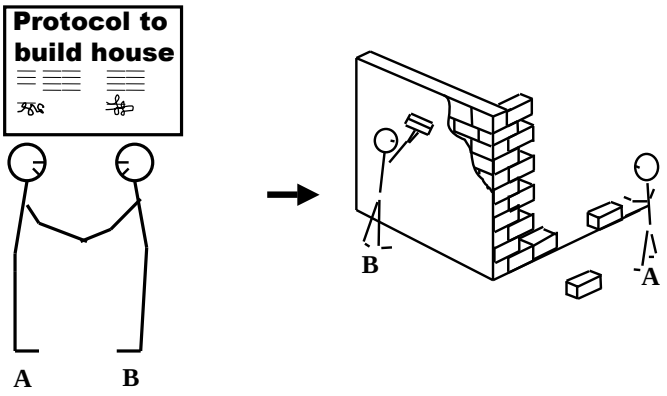


Figure 4.2: Simple example of a joint activity that is conducted after agreeing on a common coordination protocol.

For instance, the agent 'B' has to wait until that the agent 'A' builds a wall to start the painting activity. One can notice that on one hand the termination of a task may produce resources that were not initially available. On the other hand, some tasks wait the availability of some resources in order to start. In the presented example, the wall is produced by the task of agent 'A' and this resource is needed by agent 'B' to start its task of painting.

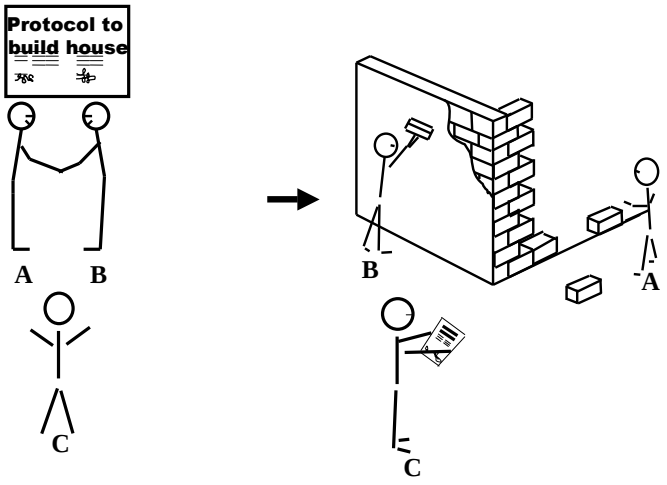


Figure 4.3: Introduction of a monitoring entity in order to check the consistency of the joint activity according to the coordination protocol.

Since agents 'A' and 'B' are autonomous, their actions at the execution time may not be consistent with the agreed coordination protocol. Consequently, the scheme of Figure 4.2 is extended by the one presented in Figure 4.3 such that a particular monitoring entity 'C' is introduced. The role of this entity is to check if the actions of the autonomous agents are consistent with their commitments.

In order to express coordination protocols, we introduce the dependency-based coordination model (DBCM). The DBCM is a formalism used to express coordination protocols *only* with the producer/consumer relationships. The main concepts of this formalism are described as follows:

- activities: an activity represents an atomic and indivisible task that starts when all the consumable resources are present and completes in a limited time by producing other resources.
- resources: a resource represents anything that may be consumed or produced by the activities. Resources follow also a coherency constraint stating that any resource that has been consumed disappears and any resource that has been produced appears within the system.
- usability dependency: according to Malone and Crowston, each producer/consumer dependency induces a usability dependency. The usability guarantees that the resource is usable by the consuming activity after being produced by the producing activity. Consequently, for each resource a control function is provided.
- roles: the functions of the agents are abstracted as *roles*. The roles are considered in the DBCM in order to situate where the activities are executed during the coordination process.

By using the DBCMs, the coordination protocols are graphically represented as directed graphs (digraphs): activities are represented as rectangles, resources as ovals and roles are represented as areas that contain tasks.

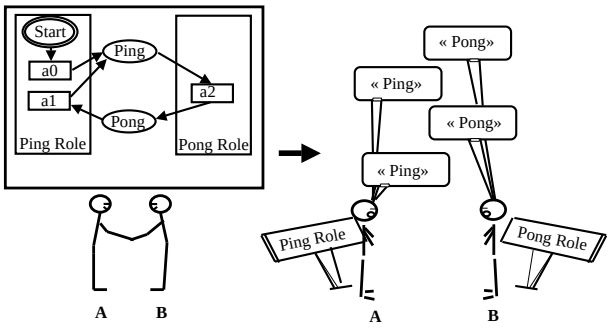


Figure 4.4: The Ping-Pong joint activity.

For instance, Figure 4.4 presents the coordination protocol that is shared among the agents 'A' and 'B' in order to play ping-pong. The coordination protocol is expressed as a digraph: a_0, a_1 and a_2 are the activities and `Start`, `Ping` and `Pong` are the resources. Activities are linked to the resources by two kinds of arrow: the production arrow, that starts from an activity and ends with a resource; the consumption arrow, that starts from a resource and ends with an activity. The production arrow is interpreted as the production of a resource by an activity and the consumption arrow is interpreted as a consumption of a resource by the activity. When an activity produces several resources, this is interpreted as the simultaneous production of all the resources. When an activity consumes several resources, all these resources have to be available to start this activity. On the other hand, when a resource is produced by several activities, every task that terminates produces this resource. When a resource is consumed by several activities, this is interpreted as an exclusive choice and only one activity consumes the resource.

The order relationship among the activities and resources is expressed by the coordination protocol. For instance, $a_0 < a_1$, $a_1 < a_2$ and $a_2 < a_1$; `Start`<`Ping`, `Ping`<`Pong` and `Pong`<`Ping`. Activities are contained within roles. For instance, $\{a_0, a_1\} \in$ `PingRole` and $\{a_2\} \in$ `PongRole`. The resource `Start` is drawn using a double circle to indicate that this resource is initially available within the system. It is considered as an *axiom* of the coordination protocol.

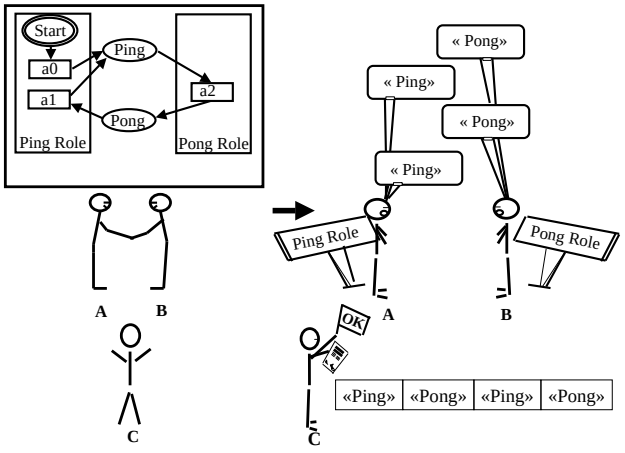


Figure 4.5: Introducing the monitoring entity within the Ping-Pong example.

Now let us introduce the monitoring entity 'C' in the presented example (cf. Figure 4.5). The entity 'C' is an external observer of the coordination process. Consequently, this entity can observe only resources that are exchanged among agents 'A' and 'B'. These resources appear as an ordered sequence that represents the *conversation* among agents 'A' and 'B'. The monitoring entity 'C' has to check the consistency of the conversation with regards to the predefined coordination protocol. Consequently, the autonomy of agents 'A' and 'B' is not challenged and as soon as a particular agent does not follow the coordination protocol, this agent is identified and made responsible for breaking the coordination rules.

In order to check the consistency of conversations according to coordination protocols, a formal link is made between the structure of the coordination protocol and the generated

conversations (cf. §4.5.2, page 103). In fact, as shown by Figure 4.5, one can notice intuitively that the structure of the conversation is constrained and structurally defined by the coordination protocol. After establishing formally this link, the second point offers some practical means in order to check the consistency of conversations in an incremental manner (cf. §4.6.1, page 109).

4.4 Formal Definition of the Dependency-based Coordination Model

4.4.1 Definition of the Dependency-based Coordination Model

The Dependency-Based Coordination Model (DBCM) is defined formally as follows:

Definition 4.4.1. The dependency-based coordination model is a four tuple $G = (A, R, C, P)$, $A = \{a_0, a_1, ..., a_n\}$ is a finite set of activity nodes ($n \geq 0$), $R = \{r_0, r_1, ..., r_m\}$ is a finite set of resource nodes ($m \geq 0$). $R \cap A = \emptyset$. $C : A \rightarrow \mathcal{P}(R)$ is defined as the consumption function, a mapping from activity nodes to a set of resource nodes; $P : A \rightarrow \mathcal{P}(R)$ is defined as the production function, a mapping from activities nodes to a set of resources nodes.

When a resource node is consumed by several activities, this is interpreted as an exclusive choice: only one activity may consume this resource. On the other hand, when an activity produces several resources, this is interpreted as simultaneous and non-ordered production of resources.

4.4.2 The Digraph Associated to the Dependency-Based Coordination Model

For each DBCM, $G = (A, R, C, P)$, one can associate a digraph $DG = (V', A')$ that is built as follows:

Definition 4.4.2. The digraph $DG = (V', A')$ associated to a dependency-based coordination graph $G = (A, R, C, P)$ is defined as follows:

$$\begin{aligned} V' &= A \cup R \\ \forall \quad & a \in A, \forall r \in C(a) : (r, a) \in A' \\ \forall \quad & a \in A, \forall r \in P(a) : (a, r) \in A' \end{aligned}$$

The digraph is the graphical representation of the DBCM. The activities are represented as rectangles and resources as ovals. The production and consumption relationships are represented as directed arrows.

Example 4.4.1. This example represents the coordination protocol of the *ping-pong* joint activity. First, an initial activity, a_0, starts the process by producing a 'ping' resource; this resource is consumed by the pong-activity (a_2). a_2 produces a 'pong' resource that is consumed by the ping-activity (a_1). a_1 produces a 'ping' resource. The ping-pong coordination protocol is formally defined as follows:

$$
\begin{aligned}
G_{\text{ping-pong}} &= (A, R, P, C), \text{such that:} \\
A &= \{a_0, a_1, a_2\} \\
R &= \{\text{'start'}, \text{'ping'}, \text{'pong'}\} \\
P &: \begin{cases} P(a_0) &= \{\text{'ping'}\} \\ P(a_1) &= \{\text{'pong'}\} \\ P(a_2) &= \{\text{'ping'}\} \end{cases} \\
C &: \begin{cases} C(a_0) &= \{\text{'start'}\} \\ C(a_1) &= \{\text{'ping'}\} \\ C(a_2) &= \{\text{'pong'}\} \end{cases}
\end{aligned}
$$

The digraph associated to $G_{\text{ping-pong}}$ is:

$$
\begin{aligned}
DG_{\text{ping-pong}} &= (V', A'), \text{such that:} \\
V' &= A \cup R \\
A' &= \{(a_0, \text{'ping'}), (a_1, \text{'pong'}), (a_2, \text{'ping'}), \\
&\quad (\text{'start'}, a_0), (\text{'ping'}, a_1), (\text{'pong'}, a_2)\}
\end{aligned}
$$

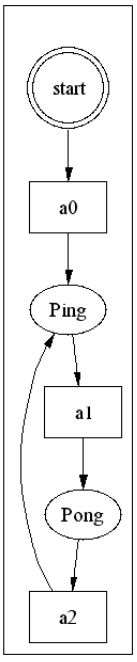


Figure 4.6: The digraph associated to the ping-pong coordination protocol.

The digraph is graphically represented by Figure 4.6. The 'start' resource is an *axiom* and thus drawn with a double circle. This means that at the initial state, this resource is available within the system. This point is discussed in more detail in section [4.5.2, page 103].

4.4.3 Partitioning the Dependency-based Coordination Model

In a MAS the coordination process involves several distributed agents. To model this situation, one can decompose the DBCM into several components representing the different sites where the activities are executed. Each execution site represents a *role* because it delimits the set of activities that a participant has to execute. In a sense, this represents its responsibilities during the coordination process. The role decomposition in a DBCM is viewed as a particular partitioning of the digraph. The roles have to be independent and thus do not share the same activities.

Definition 4.4.3. Let $G = (A, R, C, P)$, be a coordination model, let $D = \{s_0, s_1, ..., s_j\}, j \geq 0$ be the finite set of roles. A role decomposition associated to the coordination model G is a mapping $d : D \rightarrow \mathcal{P}(A)$ that associates to each role $s \in D$ a set of activities such that:

$$\begin{aligned} \forall x, y \in D \times D : x &\neq y \implies d(x) \cap d(y) = \emptyset \\ \bigcup_{x \in D} d(x) &= A \end{aligned}$$

The resources that are situated between the roles express the dependencies on resources among the roles. This part of the digraph is defined as the *roles-cut*.

Definition 4.4.4. The roles-cut of a dependency-based coordination model $G = (A, R, P, C)$, is defined as the set of resources $c \subset R$ such that each ressource links at least two acivities that belongs to two different roles.

Example 4.4.2. As an example, let us consider the ping-pong coordination protocol $G_{\text{ping-pong}}$. The set of roles is $D = \{\text{'Ping Role'}, \text{'Pong Role'}\}$. The role decomposition d that is associated to $G_{\text{ping-pong}}$ is defined as follows:

$$d : \begin{cases} d(\text{'Ping Role'}) = \{a_0, a_1\} \\ d(\text{'Pong Role'}) = \{a_2\} \end{cases}$$

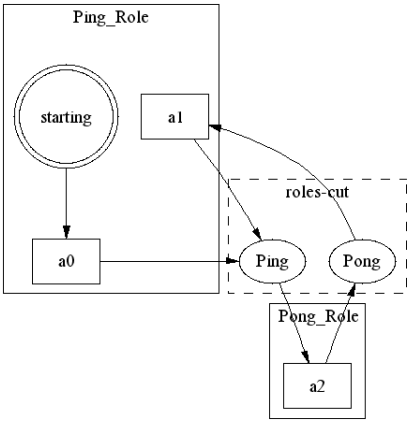


Figure 4.7: The roles and roles-cut of the 'ping-pong' coordination protocol.

The roles are graphically represented as areas that contain all the roles' activities along with the resources that are produced and consumed internally. The roles-cut is represented using dashed lines (cf. Figure 4.7).

4.5 The Generated Resource Patterns

4.5.1 Marked Resource Set

In order to define the structure of the resource patterns generated by a coordination protocol, we need to define the marked resources set. This set represents simply resources which are tagged with two values $\top$ (top) and $\bot$ (bottom).

Definition 4.5.1. Let R be a set; the market set R' associated to R is defined as follows:

$$R' = R \times \{\top, \bot\}$$

R' is partitioned in two disjoint sets $R' = R^{\top} \cup R^{\bot}$, where:

$$\begin{aligned} R^{\bot} &= \{(r, \bot), r \in R\} \\ R^{\top} &= \{(r, \top), r \in R\} \end{aligned}$$

To simplify the notations, the elements $(r, \top) \in R'$ are simply written as r and the elements $(r, \bot)$ are represented as $\bar{r}$.

$|| : R' \rightarrow R$ is a function that removes the tag from a tagged element: $|(r, \top)| \mapsto r$ and $|(r, \bot)| \mapsto r$.

4.5.2 The Structure of the Resource Patterns

The structure of the DBCM imposes some restrictions on the order and the number of the produced resources. In fact, having some initial resources called the *axioms*, one can list the set of activities that can be started. By executing these activities other resources are produced. Again another set of activities can be started. The sequence of the produced resources defines a *resource pattern*. Obviously, all the resource patterns always start by the axioms and then each time a resource is produced this resource is appended to the pattern.

The structure of the resource patterns produced by the coordination protocol is completely defined by a rewriting system where the simplification rules are extracted from the structure of the DBCM itself. To reach this point, let us first define the structure of the resource patterns generated by *any* set of simplification rules:

Definition 4.5.2. Let R be the set of resources, R' its associated marked set and $\sigma \subseteq \mathcal{P}(R) \times \mathcal{P}(R)$ the set of simplification rules. The structure of the resource patterns generated by σ is given by the rewriting relation $\xrightarrow{\sigma}: R'^{*} \times R'^{*}$ defined as follows:

$$\begin{aligned} U_0 r_1 U_1 ... r_i U_i r_{i+1} U_{i+1} ... r_n U_n \quad &\xrightarrow{\sigma} \quad U_0 \bar{r_1} U_1 \bar{r_2} U_2 ... \bar{r_i} U_i U_{i+1} ... U_n \\ &\Leftrightarrow \quad \begin{cases} \exists (\{|r_1|, .., |r_i|\}, M) \in \sigma \ : \{|r_{i+1}|, .., |r_n|\} \quad \subseteq M \\ \forall k \in [1..i], r_k \notin R^{\bot} \end{cases} \end{aligned}$$

For a set, R, the notation R^{*} designates the set of ordered sequences, that can be empty, of elements of R. $(r_i)_{i \in [1..n]} \in R'$ are tagged resources and $(U_j)_{j \in [0..n]} \in R'^{*}$ are sequences, that can be empty, of marked resources.

Interpretation 4.5.3. σ is the set of simplification rules. The interpretation of these rules is as follows: $(\{x_p\}_{p\in[0..n]}, \{x_p\}_{p\in[n+1..m]}) \in \sigma$ means that the resources contained in $\{x_p\}_{p\in[0..n]}$ can produce the resources contained in $\{x_p\}_{p\in[n+1..m]}$ (left-right interpretation). Or $(\{x_p\}_{p\in[0..n]}, \{x_p\}_{p\in[n+1..m]}) \in \sigma$ means that the resources contained in $\{x_p\}_{p\in[n+1..m]}$ consume the resources contained in $\{x_p\}_{p\in[0..n]}$ (right-left interpretation).

This models the implicit partial order relationship between the resources. Consequently, in a resource pattern the elements of the set $\{x_p\}_{p\in[0..n]}$ (no matter their order) appear *necessarily* before the elements of $\{x_p\}_{p\in[n+1..m]}$.

After the reduction:

- the resources that have been used in the left part of the reduction relation are marked to avoid confusions. In fact, this ensures that these resources are used only once.
- the resources of the right part are removed from the sequence.

What is missing in Definition 4.1 is how the simplification rules set σ is built. This set is constructed from the coordination protocol itself $G = (A, R, P, C)$ as follows:

$$\forall a \in A, (C(a), P(a)) \in \sigma_G \tag{4.1}$$

Interpretation 4.5.4. Each activity of the DBCM defines a particular production rule of the rewriting system since it consumes some resources and generates other resources. What is intersting with the rewriting relation given in Definition 4.1 is the fact that the parallal production and consumption of the resources is fully captured. In fact, when several resources are produced or consumed by activities, it is not the absolute positions of the resources within a resource pattern that is important for the validiy of the resource pattern but the relative position of the resources.

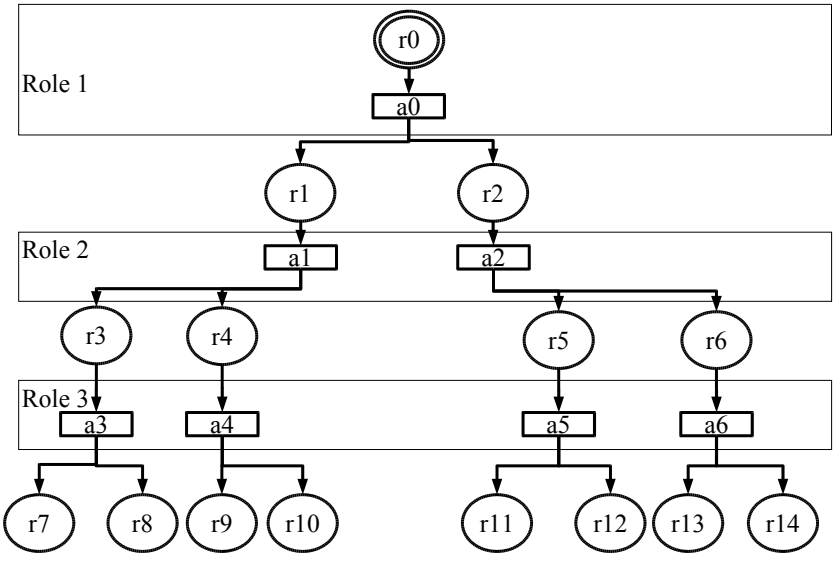


Figure 4.8: Example of a DBCM to express the relative order among the resources.

For instance, Figure 4.8 presents a DBCM that is organized as a binary tree. The resource pattern :

$$r_0.r_1.r_2.r_3.r_4.r_{10}.r_5.r_{11}.r_{12}$$

is a valid pattern since it can be reduced to the axiom as follows:

$$\begin{array}{lll} r_0.r_1.r_2.r_3.r_4.r_{10}.r_5.r_{11}.r_{12} & \stackrel{\sigma[a_5]}{\longrightarrow} & r_0.r_1.r_2.r_3.r_4.r_{10}.\bar{r_5} \\ & \stackrel{\sigma[a_2]}{\longrightarrow} & r_0.r_1.\bar{r_2}.r_3.r_4.r_{10} \\ & \stackrel{\sigma[a_4]}{\longrightarrow} & r_0.r_1.\bar{r_2}.r_3.\bar{r_4} \\ & \stackrel{\sigma[a_1]}{\longrightarrow} & r_0.\bar{r_1}.\bar{r_2} \\ & \stackrel{\sigma[a_0]}{\longrightarrow} & \bar{r_0} \end{array}$$

Now, if the positions of some resources are changed without challenging the relative order imposed by the structure of the DBCM, then the resource pattern is still valid. For instance, let us change the position of r_2, r_{11} and r_{12} in the previous pattern as follows:

$$r_0.r_1.r_3.r_4.r_{10}.r_2.r_5.r_{12}.r_{11}$$

This resource pattern is still valid and here is a reduction path:

$$\begin{array}{lll} r_0.r_1.r_3.r_4.r_{10}.r_2.r_5.r_{12}.r_{11} & \stackrel{\sigma[a_5]}{\longrightarrow} & r_0.r_1.r_3.r_4.r_{10}.r_2.\bar{r_5} \\ & \stackrel{\sigma[a_2]}{\longrightarrow} & r_0.r_1.r_3.r_4.r_{10}.\bar{r_2} \\ & \stackrel{\sigma[a_4]}{\longrightarrow} & r_0.r_1.r_3.\bar{r_4}.\bar{r_2} \\ & \stackrel{\sigma[a_1]}{\longrightarrow} & r_0.\bar{r_1}.\bar{r_2} \\ & \stackrel{\sigma[a_0]}{\longrightarrow} & \bar{r_0} \end{array}$$

However, by changing the position of r_{11} and r_5 as follows:

$$r_0.r_1.r_2.r_3.r_4.r_{10}.r_{11}.r_5.r_{12}$$

This resource pattern is no more valid since one cannot observe the resource r_{11} before r_5. So, the resource r_{11} is never consumed by any simplification rule. For instance, this is an attempt to reduce the previous pattern:

$$\begin{array}{lll} r_0.r_1.r_2.r_3.r_4.r_{10}.r_{11}.r_5.r_{12} & \stackrel{\sigma[a_5]}{\longrightarrow} & r_0.r_1.r_2.r_3.r_4.r_{10}.r_{11}.\bar{r_5} \\ & \stackrel{\sigma[a_2]}{\longrightarrow} & r_0.r_1.\bar{r_2}.r_3.r_4.r_{10}.r_{11} \\ & \stackrel{\sigma[a_4]}{\longrightarrow} & r_0.r_1.\bar{r_2}.r_3.\bar{r_4}.r_{11} \\ & \stackrel{\sigma[a_1]}{\longrightarrow} & r_0.\bar{r_1}.\bar{r_2}.r_{11} \\ & \stackrel{\sigma[a_0]}{\longrightarrow} & \bar{r_0}.r_{11} \end{array}$$

This pattern is not valid since $r_0.r_{11}$ is not the axiom of this DBCM.

It is worth noting that some resources were not considered in the presented patterns. But, these patterns were considered as valid patterns. For instance, the resources: $r_7, r_8, r_9, r_6, r_{12}, r_{13}, r_{14}$ did not appear within the valid patterns. This is due to our assumptions of work and our objective. In fact, the objective is to check the validity of the coordination process during its excution. So, we have not to wait until the termination of the coordination process to decide if the patterns are valid or not. But as soon as some resources appear we have to decide, as observers, if the coordination process is invalid.

Finally, the set of all resource patterns generated by a particular DBCM given an axiom is defined as:

Definition 4.5.5. Let $G = (A, R, P, C)$ be the dependency-based coordination model. The resource patterns set $\text{r\_patterns}_{(G,x_0)}$ generated by a dependency-based coordination model G and the axiom $x_0 \in R^*$ is defined as follows:

$$\forall u \in R^*, u \in \text{r\_patterns}_{(G,x_0)} \iff \exists u' \in R'^* : u' \overset{\sigma_G}{\rightsquigarrow} x_0 \wedge |u'| = u$$

So, any sequence of resources that can be simplified to the axiom after a certain application of the simplification rules is member of the generated resource patterns (in the right-left interpretation). One can have a more natural interpretation. Any sequence of resources that can be produced using the production rules starting from the axiom belongs to the set of generated resource patterns (in the left-right interpretation).

Example 4.5.1. By applying the definition given in Equation 4.1, the set of simplification rules of the ping-pong coordination protocol $G_{\text{ping-pong}}$ is defined as follows:

$$\sigma_{G_{\text{ping-pong}}} = \{(\{\text{'start'}\}, \{\text{'ping'}\}), (\{\text{'ping'}\}, \{\text{'pong'}\}), (\{\text{'pong'}\}, \{\text{'ping'}\})\}$$

The set of generated resource patterns $\text{r\_patterns}_{(G_{\text{ping-pong}},\text{start})}$ is completely known and Table 4.1 gives some examples of resource patterns and how they are reduced to the axiom 'start'.

Pattern	? $\in \text{r\_patterns}_{(G_{\text{ping-pong}},\text{start})}$	Reduction path
start.ping	yes	start.ping $\xrightarrow{\sigma}$ st$\bar{\text{a}}$rt
start.ping.pong	yes	start.ping.pong $\xrightarrow{\sigma}$ start.pi$\bar{\text{n}}$g $\xrightarrow{\sigma}$ st$\bar{\text{a}}$rt
start.ping.pong.ping	yes	start.ping.pong.ping $\xrightarrow{\sigma}$ start.ping.po$\bar{\text{n}}$g $\xrightarrow{\sigma}$ start.pi$\bar{\text{n}}$g $\xrightarrow{\sigma}$ st$\bar{\text{a}}$rt
start.pong	no	start.pong (cannot be simplified)
start.ping.ping	no	start.ping.ping $\xrightarrow{\sigma}$ st$\bar{\text{a}}$rt.ping (cannot be simplified)

Table 4.1: Example of resource patterns of the ping-pong coordination model.

4.5.3 Recognizing Actual Resources by Using the Control Functions

First, let us distinguish between the resource nodes that are defined by the DBCM and the actual resources that are exchanged among the coordinating entities. The resource nodes are

abstractions defined in the DBCM to 'talk about' the actual resources. The actual resources are the actual 'things' that are exchanged between the coordinating entities. In the context of this work, the interaction objects set, $\mathcal{O}$, is considered as the actual resources set.

The link between the resource nodes and actual resources is made by the *control functions*[2]. A control function is a Boolean function that returns true when the actual resource conforms the standard that has been established between the producer and consumer, and false otherwise.

So, for each resource node a control function is associated in order to control if the actual resource corresponds to the resource node. Still, some resource nodes do not require an exchange of actual resources. In fact, some resource nodes model a prerequisite relationship among the activities without a concrete exchange of an actual resource. For these particular resource nodes, a particular value (null or zero) is associated as a control function. The resource nodes that belong to the roles-cut must always be mapped to non-null control functions. In fact, in a distributed system the communication between the distributed activities always implies the explicit exchange of interaction object.

The resource recognition mapping, κ, is introduced as an assignment of control functions to resource nodes in a DBCM:

Definition 4.5.6. Let $G = (A, R, P, C)$ be a coordination model and $\mathcal{O}$ the set of actual resources. The resource control mapping κ of the coordination model G is a function from the set of resource nodes to the set of control functions or a null value:

$$\kappa : R \rightarrow (\mathcal{O} \rightarrow \{\top, \bot\}) \cup \{0\}$$

The roles-cut structure imposes that only the resources belonging to the role-cut are observable by an external observer. So, the mapping κ is said to *satisfy* the constraints of the roles-cut c iff:

$$\forall r \in R \begin{cases} \kappa(r) = 0 & \text{if } r \notin c \\ \kappa(r) \neq 0 & \text{if } r \in c \end{cases}$$

The recognition of an actual resource sequence $m = m_0 m_1 ... m_n \in \mathcal{O}^*$ according to a resource nodes pattern $p = r_0 r_1 ... r_m \in R^*$ and a mapping κ is performed as follows:

1. each resource node in the resource pattern p is replaced by its associated control function defined by κ. This produces a sequence of control functions $f = f_0 f_1 ... f_j \in ((\mathcal{O} \rightarrow \{\top, \bot\}) \cup \{0\})^*$.
2. the sequence of functions is normalized by removing all the zero (or null) elements. The normalized sequence is noted $f' = ||f||$.
3. the sequence of functions 'is applied' to the sequence of actual resources m. This is done by replacing each i-th element of m (m_i) by the application of the i-th function of f' to m_i. So, the result of this operation is a sequence of Boolean values.

[2]Malone and Crowston works have already introduced the concept of *control function* when they were speaking about the usability dependency.

4. if the resulting sequence contains only $\top$ values, this means that the actual resources sequence has been *validated* by the resource pattern and the recognition mapping κ; otherwise, it has been invalided.

To simply the notations, the fact that a resource pattern p recognizes an actual resource sequence m according to the mapping κ is noticed by a logical predicate: $\texttt{validate}(p, c, \kappa)$.

For each pattern of resource nodes (that is element of R^*) and a mapping κ, we can define the set of all actual resources (elements of $\mathcal{O}^*$) which are recognized. Within the MAS paradigm the $\mathcal{O}$ set is interpreted as interaction objects. So, the sequence of actual resources is defined as a *conversation* and the set of all recognized conversations is defined as the set of *valid conversations*.

Definition 4.5.7. Let $\mathcal{O}$ be the set of interaction objects; $G = (A, R, P, C)$ a DBCM associated to the resource recognition mapping κ and a roles-cut d such that the mapping κ satisfies the constraints of the roles-cut d. The set of valid conversations generated by G and the axiom $x_0 \in R^*$ is defined as the set of all the sequences of interaction objects that are validated by at least one resource pattern of G according to the axiom x_0:

$$\forall c \in \mathcal{O}^*, c \in \text{conversations}_{(G,\kappa,d,x_0)} \iff \exists p \in \text{r\_patterns}_{(G,x_0)} : \texttt{validate}(p, c, \kappa)$$

Example 4.5.2. For this example, we assume that the set of actual resources contains two binary values $\mathcal{O} = \{0, 1\}$. The resources of $G_{\text{ping-pong}}$ are linked to this set by the following mapping[3] κ:

$$\kappa : \begin{cases} \kappa(\text{start}) & \mapsto 0 \\ \kappa(\text{ping}) & \mapsto \lambda\texttt{(x):(x = 0)} \\ \kappa(\text{pong}) & \mapsto \lambda\texttt{(x):(x = 1)} \end{cases}$$

Now, let us follow the presented steps to show that the actual resources sequence, $s =$ 0.1.0.1 is validated by the resource pattern $r =$ 'start.ping.pong.ping.pong'.

1. each element of r is replaced by its associated recognition function:

$$\begin{aligned} f &= \kappa(start).\kappa(ping).\kappa(pong).\kappa(ping).\kappa(pong) \\ &= 0.\lambda(x) : (x = 0).\lambda(x) : (x = 1).\lambda(x) : (x = 0).\lambda(x) : (x = 1) \end{aligned}$$

2. normalization of the recognition functions sequence: this is done by removing all the zero elements from the sequence of functions:

$$||f|| = \lambda(x) : (x = 0).\lambda(x) : (x = 1).\lambda(x) : (x = 0).\lambda(x) : (x = 1)$$

3. apply the sequence of recognition functions to the sequence of the actual resources:

$$\begin{aligned} ||f||(s) &= \lambda(x) : (x = 0)(0).\lambda(x) : (x = 1)(1).\lambda(x) : (x = 0)(0).\lambda(x) : (x = 1)(1) \\ &= (0 = 0).(1 = 1).(0 = 0).(1 = 1) \\ &= \top.\top.\top.\top \end{aligned}$$

4. we can conclude that the sequence s have been validated by the pattern r. Furthermore, since the pattern r belongs to $\text{r\_patterns}_{(G_{\text{ping-pong}},\text{start})}$, then the sequence of actual resources s belongs to the valid conversations set defined by this coordination model.

[3] The sign = has to be interpreted as the logical comparison.

4.6 Validating the Conversations According to a Coordination Protocol

The previous section has presented a formal link between the coordination model and the generated conversations. Still, the presented results show only that this link exists and do not give the means in order to decide if a conversation belongs or not to the valid conversations of a coordination protocol. This section presents a method that uses the Queue Petri Nets (QPN) in order to validate the conversations in an incremental manner.

4.6.1 Informal Presentation of Petri Nets

The Petri Net is a well-known formalism, invented by Carl Adam Petri, to express concurrent activities. This formalism has already been used to express *directly* the conversation protocols in several works (cf. §4.2.4, page 96). First, let us present informally the structure of a Petri Net and give an extension used to validate the conversations.

The Petri Net is a digraph composed of *transitions* (graphically represented as rectangles); *places* (graphically represented as circles); and *tokens*. The places contain zero or more tokens. The transitions are linked to places by two kinds of arrows:

- production arrow: this arrow is directed from a transition to a place. When a transition is fireable, the tokens are produced in each place linked to this transition by the production arrows.
- consumption arrow: this arrow is directed from a transition to a place. A transition is said to be fireable when all the places linked to this transition by a consumption arrow contains at least one token. When the transition is fired, the tokens of consumption places are decremented by one.

A more complete and detailed presentation of the Petri Net theory is given in Annex B.

4.6.2 The Queue Petri Net Theory

In order to validate the conversations among the agents, we have extended the original Petri Net theory that is presented Annex B to define the *Queue Petri Nets* (QPN). The QPN is a structure composed of a Petri Net and a queue of interaction objects. To be fireable, the transitions of the QPN do not depend only on the tokens but also on the queue. When a transition is fired, it consumes the tokens and one object found on the front of the queue. The QPN are formally built by following the same steps as for simple Petri Nets.

Queue Petri-Net Structure

Definition 4.6.1. Let $\mathcal{O}$ be the set of interaction objects; $\mathcal{O}^*$ the set of ordered sequences of interaction objects. The Queue Petri Net structure is a six tuple $C = \{P, T, I, O, F, q\}$, $P = \{p_1, p_2, ..., p_n\}$ is a finite set of places $n \geq 0$ and $T = \{t_1, t_2, ..., t_m\}$ is a finite set of transitions $m \geq 0$. $P \cap T = \emptyset$. $I : T \rightarrow P^\infty$ is defined as the input function, a mapping from transitions to bags of places and $O : T \rightarrow P^\infty$ is defined as the output function, a mapping from transitions to bags of places; $F : T \rightarrow \{0\} \cup (\mathcal{O} \rightarrow \mathbb{B})$, is defined as the message control

function, a mapping from transitions to control functions or zero; $q \in \mathcal{O}^*$ is defined as the message queue. That is an ordered sequence of elements belonging to $\mathcal{O}$.

Queue Petri Net Marking

The marking of the QPN is similar to simple Petri Nets (see Annex B).

Execution Rules for QPN

The execution of the QPN is totally controlled by the tokens distributed over places and the objects of the queue. The QPN executes by *firing* transitions. For the QPN, two kinds of transitions are identified:

1. transitions that are associated to zero (or null) as control function: these transitions fire as normal Petri Net transitions by consuming the tokens from their input places and producing new tokens distributed over there output places.
2. transitions that are associated to non-null control functions: these transitions are enabled only if there are enough tokens in their input places and the associated control function validates the first object found in the queue. The firing of these transitions consumes the tokens of the input places and produces new tokens in the output places. Besides, the first object of the queue is removed.

Definition 4.6.2. Transition $t_j \in T$ in a marked Queue Petri Net $C = (P, T, I, O, F, q)$ with marking μ is enabled iff:

1. for all $p_i \in P$:
$$\mu(p_i) \geq \#(p_i, I(t_j))$$
2. if $F(t_j) \neq 0$ then:
$$F(t_j)(front(q)) = \top$$

Definition 4.6.3. Let t_j be a fireable transition in a marked Queue Petri Net $C = (P, T, I, O, F, q)$ with the marking μ. The firing of this transition results in a new marking μ' defined as follows:
$$\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j))$$

And a new message queue q' defined as follows:

$$q' = \begin{cases} pop(q) & \text{, if } F(t_j) \neq 0 \\ q & \text{, if } F(t_j) = 0 \end{cases}$$

When there are no fireable transitions, the QPN is said to be *dead*.

Definition 4.6.4. The Queue Petri Net $C = (P, T, I, O, F, q)$ with the marking μ is said to have recognized its queue q if there is an execution path that leads to C': $C \rightsquigarrow C' = (P, T, I, O, F, q')$ with a marking μ' and where the queue q' is empty.

4.6.3 Building the QPN from the Coordination Protocol

The goal of this section is to present an algorithm that builds the QPN from a given DBCM. Before presenting this algorithm, let us first define the *block* structure.

The Block Structure

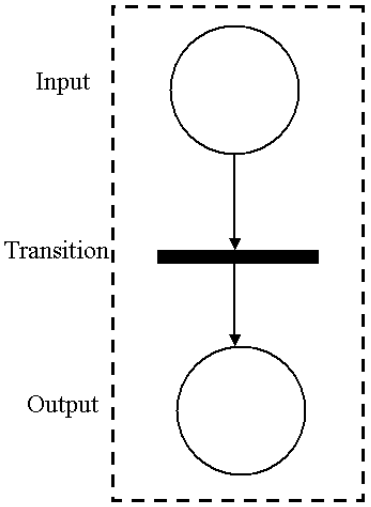


Figure 4.9: The block structure composed of: an input place, an output place and a transition.

The block structure in a Petri Net is composed of two places and a transition: namely the input and output place and the transition of the block. The input place is linked by a production arrow to the block's transition, and the output place is linked by a consumption arrow to the transition of the block. Graphically, the block corresponds to the structure illustrated in Figure 4.9.

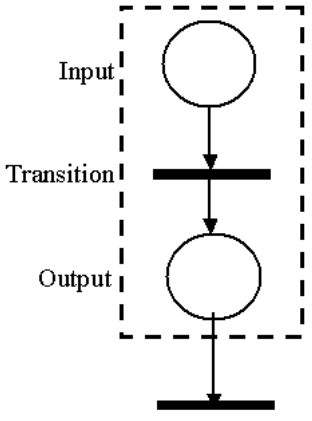


Figure 4.10: Production relationship between a block and a transition.

Linking a block to a transition by a production relation means that the output place of the block is linked with a production arrow to this transition (cf. Figure 4.10).

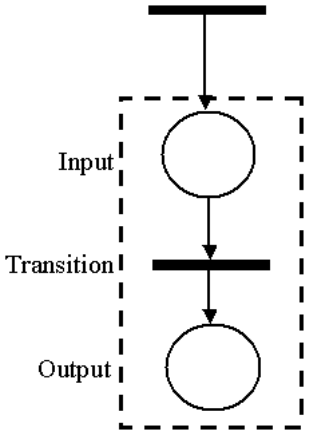


Figure 4.11: Consumption relationship between a block and a transition.

Similarly, linking a block to a transition by a consumption relation means that the input place of the block is linked by a consumption arrow to this transition as presented in (cf. Figure 4.11).

Algorithm to Build the QPN from the DBCM

Algorithm 1 presents how the QPN is built from the DBCM. This algorithm is described as follows:

1. the coordination model $G(A, R, P, C)$; the axiom x_0; the roles-cut d; the control function mapping κ are considered as the inputs of the algorithm. Besides, κ is assumed to satisfy the constraints of the roles-cut d;
2. for each resource $r \in R$ one has to :
   (a) build its associated block identified as b_r;
   (b) if the resource is an axiom then marks the input place of b_r with 1, otherwise the marking of the block places is 0;
   (c) if $r \in d$ then:
      i. associate to the transition of the block b_r the control function $\kappa(r)$;
3. for each activity $a \in A$ do:
   (a) make a transition identified as t_a;
   (b) make a production links between t_a and the blocks of resource found in $P(a)$;
   (c) make a consumption links between t_a and the blocks of resource found in $P(a)$;
4. return the built QPN as result;

Example 4.6.1. Algorithm 1 is applied to the ping-pong coordination protocol in order to build the associated QPN. The parameters, $G = (A, R, P, C)$, d and κ are similar to section [4.5.2, page 108].

Algorithm 1 Building the structure of the QPN from the DBCM coordination protocol

1: **function** FROMCOORDINATIONMODEL2QPN(cm,x_0,d,κ) ▷ cm represents the coordination model structure, x_0 the axiom of the coordination model, d is the set of resources found in the roles-cut and κ represents a given recognition functions mapping.
Require: satisfies_roles-cut_constraints(κ,d)
2: $resource2block \leftarrow$ new_map() ▷ This is a mapping that links each resource node to its block.
3: $qpn \leftarrow$ new_QPN() ▷ This is the QPN that is returned by this function.
4: **for all** $r \in cm$.getResourceNodes() **do**
5: $b \leftarrow$ new_block(r)
6: **if** $r \in x_0$ **then**
7: b.getInputPlace().setMarking(1)
8: **else**
9: b.getInputPlace().setMarking(0)
10: **end if**
11: **if** $r \in d$ **then**
12: b.getTransition().setCrtlFunction($\kappa(r)$)
13: **else**
14: b.getInputPlace().setCrtlFunction(0)
15: **end if**
16: $resource2block[r] \leftarrow b$
17: qpn.addBlock(b)
18: **end for**
19: **for all** $a \in cm$.getActivityNodes() **do**
20: $t \leftarrow$ new_transition(a)
21: qpn.addSimpleTransition(t)
22: **for all** $r \in cm$.getProducedResources(a) **do** qpn.makeBlockProductionLink($resource2block[r]$,t)
23: **end for**
24: **for all** $r \in cm$.getConsumedResources(a) **do** qpn.makeBlockConsumptionLink($resource2block[r]$,t)
25: **end for**
26: **end for**
27: **return** qpn
28: **end function**

The resulting QPN is:

$$
\begin{aligned}
QPN &= (P, T, I, O, F, q) \\
P &= \{p_0, p_1, p_2, p_3, p_4, p_5\} \\
T &= \{t_0, t_1, t_2, t_3, t_4, t_5\} \\
I &= \begin{cases} I(t_0) &= \{p_0\} \\ I(t_1) &= \{p_2\} \\ I(t_2) &= \{p_4\} \\ I(t_3) &= \{p_1\} \\ I(t_4) &= \{p_3\} \\ I(t_5) &= \{p_5\} \end{cases} \\
O &= \begin{cases} O(t_0) &= \{p_1\} \\ O(t_1) &= \{p_3\} \\ O(t_2) &= \{p_5\} \\ O(t_3) &= \{p_2\} \\ O(t_4) &= \{p_4\} \\ O(t_5) &= \{p_2\} \end{cases} \\
F &= \begin{cases} F(t_0) &= 0 \\ F(t_1) &= \lambda(x) : (x = 0); \\ F(t_2) &= \lambda(x) : (x = 1); \\ F(t_3) &= 0 \\ F(t_4) &= 0 \\ F(t_5) &= 0 \end{cases} \\
q &= \text{empty sequence}
\end{aligned}
$$

The resulting QPN is presented in Figure 4.12. The figure shows also that all the axiom elements have been initialized with a marking equals to 1. In fact, these resources are present initially within the system. So, the QPN starts by firing the transitions that consume the axiom places and then follows the execution rules presented in section [4.6.2, page 110].

4.6.4 Algorithm to Validate Conversations using the QPNs

The previous section has presented how the structure and the marking of the QPN are generated from a DBCM coordination protocol. The QPN implements the same order constraints on the actual resources as in the coordination model. To recognize a conversation between agents, the sequence of the exchanged messages are considered as the queue of the QPN. If there is an execution path that empties the queue of the QPN, then the conversation is valid.

The control of the validity is performed incrementally. This means that the validity of the conversation is checked each time a new message is exchanged among the coordinating entities. Consequently, the invalid conversations that do not follow the order constraints of the coordination protocol are detected since the occurrence of the first error.

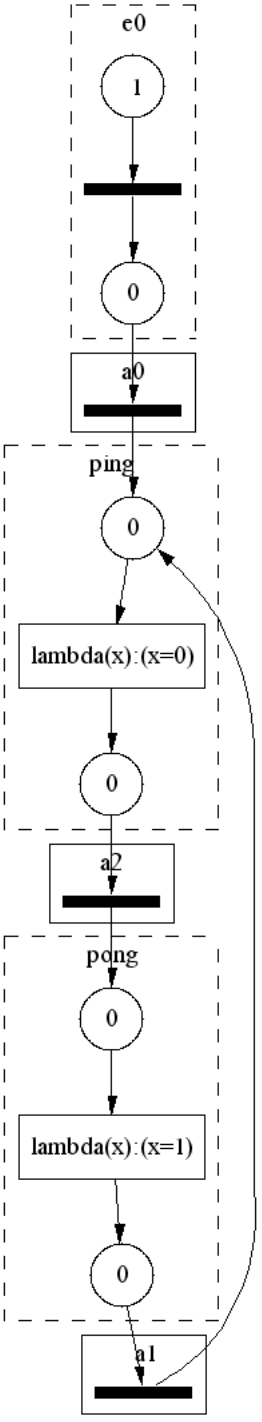


Figure 4.12: The QPN associated to the ping-pong coordination protocol.

This section presents a set of algorithms to validate incrementally the conversations. Several QPNs may be used in this process. In fact, by following an execution path, a QPN may offer different transitions that are fireable. This represents the different alternatives of the QPN. Some of these transitions are not in a conflict. So, no matter which transition is fired first, the final result is still the same. However, some of the transitions are conflicting and the order of execution changes the final result. There are two conflicting situations:

- conflict on tokens: within classical Petri Nets transitions are conflicting when they consume tokens from the same places;
- conflict on messages: within QPN another conflict is introduced. It concerns the message queue. In fact, when the control functions associated to different transitions recognizes the same message of the queue.

Each conflict situation represents an alternative of the execution path of the QPN. To explore all the possible execution paths offered by a single QPN, different QPNs are used in order to represent the alternatives resulting from the conflicts either on tokens or messages. Consequently, when a QPN offers different alternatives, let us say n different alternatives, then $n-1$ QPN are created by 'cloning' the current QPN and each of these QPNs fires a particular transition.

Algorithm 2 Algorithm to validate incrementally a conversation according to a coordination model

```
1:  procedure ControlConversation(cm,x0,d,κ)
2:      init_qpn ← CoordinationModel2QPN(cm,x0,d,κ)        ▷ see Algorithm 1
3:      qpn_set ← { init_qpn }
4:      while m ← GetNewExchangedMessage() do   ▷ Gets from the system the
            observed message exchanged during the coordination process.
5:          test ← IsTheConversationValid(m,qpn_set)        ▷ see Algorithm 3.
6:          if ¬test then
7:              RaiseException("Invalid Conversation")
8:          end if
9:      end while
10: end procedure
```

The inputs of Algorithm 2 are the coordination protocol, the axiom, the roles-cut and the control function mapping. The first step, line 2, builds the QPN associated to the coordination protocol using Algorithm 1. When a message has been observed (line 4) a call is made to the function defined in Algorithm 3 in order to check if the conversation is still valid or not. Notice that this function takes as argument the newly observed message and the set of QPNs representing the state of the conversation. If the test fails, an exception is raised informing that the conversation is no longer consistent with its related coordination protocol.

The function presented in Algorithm 3 takes as argument the newly observed message and the set of QPNs. First, a copy is made of the QPNs set, line 3, to avoid confusion with the loop of line 4. Each QPN is updated by appending its queue by the incoming message (line 5). The QPN is then asked to validate the conversation using the function defined in Algorithm 4. The QPN returns a Boolean value informing if the queue of the messages has been recognized or not. Besides, the set of the QPNs is updated by inserting the new alternatives that have been produced during the execution path. If at least one QPN has recognized its message

Algorithm 3 IsTheConversationValid function used in Algorithm 2

```
1: function IsTheConversationValid(m,qpnset)
2:     global_result ← ⊥
3:     qpnset_copy ← qpnset                          ▷ Make a copy of the set.
4:     while (n ← qpnset_copy.getNextElement()) ≠ 0 do
5:         n.getQueue().append(m)   ▷ The queues of the QPNs are updated by appending
               the newly observed message.
6:         local_result ←ValidateByThisQPN(n,qpnset)          ▷ see Algorithm 4.
7:         global_result ← global_result ∨ local_result
8:     end while
9:     if global_result then
10:        qpnset ← qpnset_copy                  ▷ Update the state of the conversation
11:    end if
12:    return global_result
13: end function
```

queue (line 7), the conversation is valid and a true Boolean value is returned. Otherwise a false Boolean value is returned (line 2).

The function presented in Algorithm 4 checks the validity of the conversation according to a given QPN and its alternatives. The initial QPN and the set of QPNs are taken as argument of the function. The function asks the initial QPN to follow an execution path until no fireable transitions are available (line 3). If the queue of the QPN has been emptied during this process, this means that the sequence of messages has been recognized by the QPN and consequently the conversation is valid (the true Boolean value is returned). On the other hand, if the queue is not empty this means that the sequence of messages has not been recognized and consequently the conversation is invalid (the false Boolean value is returned). The alternatives that have been generated during the execution process are stored in the set passed as parameter. If the QPN that was given as parameter of the function validates the conversation and is not member of the global QPNs set, it is added to this set. On the other hand, if it does not recognize the conversation and is member of the global QPNs set, it is removed from this set. The function is then recursively applied to the alternatives (line 17). When the initial QPN or at least one of its alternatives validate the conversation, a true Boolean value is returned. Otherwise a false Boolean value is returned.

Validation and Limitations of the Presented Algorithms

The function presented in Algorithm 2 terminates only under some assumptions made on the structure of the QPN generated from the coordination protocol. In fact, the generated QPN should not contain any *free cycle*. A free cycle is a cyclic path found in the QPN where all the transitions within this path do not depend on the queue. This means that all the transitions of the cyclic path are associated with control functions equal to 0. This creates a problem since the QPN, as for classical Petri Nets, can be always alive and does not reach a 'dead' state. So, the statement of line 3 of Algorithm 4 does not terminate and consequently the whole algorithm does not terminate. Therefore, the presented algorithms are well defined only on coordination protocols that generate QPN without free cycles.

This restriction does not restraint the expressiveness of the DBCM coordination protocols. In fact, the main goal of the DBCM is not to model the internal activities of roles but the

Algorithm 4 ValidateByThisQPN function used in Algorithm 3

```
1:  function ValidateByThisQPN(n,qpns_pool)
2:      sub_qpns ← {}
3:      result ← n.validate(sub_qpns) ▷ The alternatives that are found during the execution
            of the QPN are stored in the set 'sub_qpns'.
4:      if result then
5:                                          ▷ This QPN has recognized the conversation.
6:          if n ∉ qpnset then
7:              qpns_pool.addElement(n)
8:          end if
9:      else
10:         ▷ This QPN has not recognized the conversation. It should be removed from the
                pool of QPNs
11:         if n ∈ qpnset then
12:             qpns_pool.removeElement(n)
13:         end if
14:     end if
15:     for x ∈ sub_qpns do
16:         if x ∉ qpns_pool then      ▷ [optimization]Avoids to call the function on the same
                QPN structure and the same state several times.
17:             sub_result ← validateByThisQPN(x,qpns_pool)      ▷ Recursive call to the
                    function.
18:         end if
19:         result = result ∨ sub_result
20:     end for
21:     return result
22: end function
```

relationships among the activities of the different roles. Since these relationships are expressed by an exchange of resources through the roles-cut, this implies that the blocks corresponding to the resources of the roles-cut are always associated to a non-zero recognition function. So, the generated QPN does not contain free cycles.

Having this assumption on the structure of the generated QPN, one can prove that the defined algorithms always terminate in a finite amount of time.

Proof. An algorithm that emulates the execution of a Petri Net (called also an exploration algorithm) terminates in a limited time $\iff$ the graph of reachable states for this Petri Net, given an initial marking, contains only finite paths. The graph of reachable states is a simple digraph, where the nodes represent the different states of the Petri Net. The state of the Petri Net is defined by the number of tokens found in each place. A directed arrow, labelled by a transition, is made between two state nodes, if one can reach the state at the end of the arrow from the state at the beginning of the arrow by firing the transition.

For the QPN, the state is not only defined by the marking of the places but also by the queue of the actual resources. So, the nodes of the reachable states graph include the marking and the state of the queue.

For any QPN, one can notice that, by following the paths in the reachable states graph, the size of the queue remains either unchanged: this means that a standard transition has been fired or decrease by one: this means that a transition associated to a non-null control function has been fired.

So, the only way to find an infinite path in the reachable states graph of a QPN is to find a path that does not modify the queue. But this is simply impossible in QPNs that do not contain free cycles. In fact, each cycle in the QPN must have at least one transition that modifies the state of the queue. This implies that the reachable states graph does not contain infinite paths.

For instance, let us suppose that such infinite path, $s = s_0 \stackrel{t_0}{\rightarrow} s_1 \rightsquigarrow s_n ...$, can be found in the states graph associated to a QPN that does not contain free cycles. The fact that the QPN does not contain free cycles means that there is not an infinite path that does not contain a message consuming transition:

$$\forall n \in \mathbb{N}, \exists m \in \mathbb{N}, m > n : \begin{cases} s = s_0 \rightsquigarrow s_n \rightsquigarrow s_{m-1} \stackrel{t_{m-1}}{\rightarrow} s_m ... \\ F(t_{m-1}) \neq 0 \end{cases} \tag{4.2}$$

Since, the queue of the QPN is a finite sequence, there is surely a message consuming transition, let say t_k, that will empty the queue:

$$\exists k \in \mathbb{N} \begin{cases} s = s_0 \rightsquigarrow s_k \stackrel{t_k}{\rightarrow} s_{k+1} ... \\ \text{The queue in the state } s_{k+1} \text{ is empty.} \end{cases}$$

By applying the result of equation 4.2, there is a fireable transition $t_{k'}, k' > k$ that belongs to the states path and that consumes a message from the queue. However, $t_{k'}$ may not be fireable since the queue remains empty after the state s_k: this contradicts the fact that the path is infinite.

Finally, since all the paths of the QPN state graph that does not contain free cycles is finite, any exploration algorithm (such as algorithm 2), that follows these paths terminates in a limited time. □

4.7 Link between the Generated Resource Patterns and the QPNs

[I have to complete this section with the proof to demonstrate that the Rewriting System and the QPN are equivalent.]

4.8 The XML Representation of DBCM Coordination Protocols

A specific DTD, presented in Annex D.1.1, defines an XML-based formalism to express the coordination protocols. The elements of this DTD are described as follows:

- CoordinationGraph: this is the root element of any coordination protocol. This element is composed of a sequence of ResourceNode; ActivityNode; ProductionLink; ConsumptionLink and ControlFunction. Only one attribute is associated with this element:
  - `name:` this represents the name that is given to the coordination protocol.
- ResourceNode: This element represents the resource node. The attributes of this element are:
  - `id:` this attribute represents a unique identifier of the resource node;
  - `label:` this attribute represents a textual description of the resource node;
  - `group:` this attribute represents the role that is associated to the resource. If the resource node belongs to the roles-cut, this field should contain 'roles-cut' as value;
  - `axiom:` when this attribute is set to 'true' the resource node is considered as an axiom of the coordination protocol;
  - `epsilon:` when this attribute is set to 'true' value, this means that a zero control function is associated to the resource node; otherwise a control function is associated to the resource node;
  - `control-function:` this attribute represents a reference to the ControlFunction element that defines the resource control function associated with this resource node.
- ActivityNode: this element represents an activity node of the coordination protocol. The attributes that are associated with this element are the following:
  - `id:` this attribute represents a unique identifier of the activity node;
  - `label:` this attribute represents a textual description of the activity node;
  - `group:` this attribute represents the role that is associated to the activity. If the activity node belongs to the 'roles-cut', this field should contain 'roles-cut' as value.
- ControlFunction: this element represents the control function that is associated to a resource node in order to recognize the actual resources. This node encapsulates an unparsed text section that includes the algorithmic definition of the control function using a standard programming language [4]. The attributes that are associated with this element are the following:

[4] C++ has been considered for this work.

  - `id:` this attribute represents a unique identifier of the control function.

- ProductionLink: this element represents a production relationship among an ActivityNode and a ResourceNode. The attributes that are associated with this element are the following:
  - `resource:` reference to ResourceNode element.
  - `activity:` reference to ActivityNode element.

- ConsumptionLink: this element represents a consumption relationship among an ActivityNode and a ResourceNode. The attributes that are associated with this element are the following:
  - `resource:` reference to ResourceNode element.
  - `activity:` reference to ActivityNode element.

To design the coordination protocols, an intuitive graphical editor has been developed. So, the users have not to manipulate directly the XML descriptions of the coordination protocols, but use simple graphical components. The editor offers some other tools such as free-cycles detection and export coordination graphs to the XML format.

Some python scripts, `cg2dot.py` and `cg2petrinetdot.py`, have been developed in order to transform the XML description of the coordination protocols into the coordination digraph and the Petri Net graphical representation using the Graphviz graph drawing tool[5].

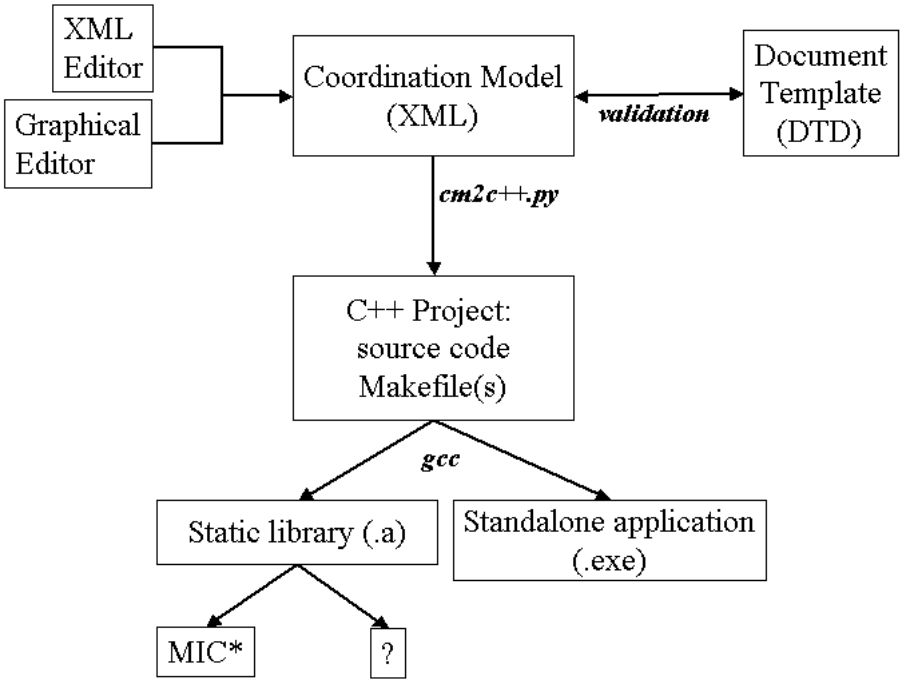


Figure 4.13: The code generation process starting from the XML description of a coordination model.

[5] Graphviz is an open source graph drawing software developed by AT&T Labs. The web page of this tool is www.research.att.com/sw/tools/graphviz

The XML description of the coordination protocols has also been used as an input for a script that generates automatically the source code of the conversation controller written in the C++ programming language. This process is illustrated by Figure 4.13. The user has only to design the coordination protocol and to compile the generated source code into either an executable program for test purposes or a library to be included as a component in other systems. For instance as shown by section [4.9, page 122], the conversation controllers are used within the MIC* DE in order to monitor the conversations of the agents according to predefined coordination protocols.

4.9 Integrating the Coordination Framework within MIC*

Chapter 3 has presented the MIC* DE. Within this model, the interaction is seen as a particular evolution law that modifies the inboxes of the agents according to the outboxes of other agents found within a particular interaction space.

The results on coordination presented in this chapter are integrated to the MIC* DE by considering the environmental dynamics. In fact, one can define an interaction law among the interaction objects that validates the ongoing conversation before modifying the inboxes of the perceiving agents. Consequently, the interaction among the agents is performed only and only if the conversation is consistent with its coordination protocol. When the conversation is invalid, the agent does not perceive any interaction object in its inbox. Furthermore, the DE can establish the responsibility of the agent that has violated the established norm on coordination.

The integration of the coordination into the MIC* framework can be done following two approaches:

1. global control of the coordination process: within this approach the conversation is monitored globally. So, the DE as an observer is assumed to maintain a global coherent state of the conversation in order to check its validity. Section [4.9.1, page 122] explores the advantages and limits of this approach.

2. local control of the coordination process: within this approach the DBCM coordination graph is subdivided into several subparts representing the roles. The DE controls the overall coordination process by controlling, independently, the coordination process of each role. This approach is discussed in section [4.9.2, page 123].

4.9.1 Global and Centralized Control of the Coordination Process

This approach implies that the DE as an observer is able to observe the whole coordination process and to maintain its state coherently. Conceptually, the composed DEs as shown in Figure 4.14 share a common data space where the states of the ongoing conversations are stored and retrieved. For instance, when the agent 'A' that is deployed on the DE (1) sends a message; the DE (1) retrieves the sate of the conversation; rebuild the set of QPNs using the retrieved state and executes Algorithm 3. When the interaction is remotely performed in the DE (2); this environment is also able to retrieve the state of the conversation and thus to control its consistency. If the conversation is valid then the interaction can be performed (for instance with agent 'B'). The state of the conversation is then updated in the shared data space for future controls.

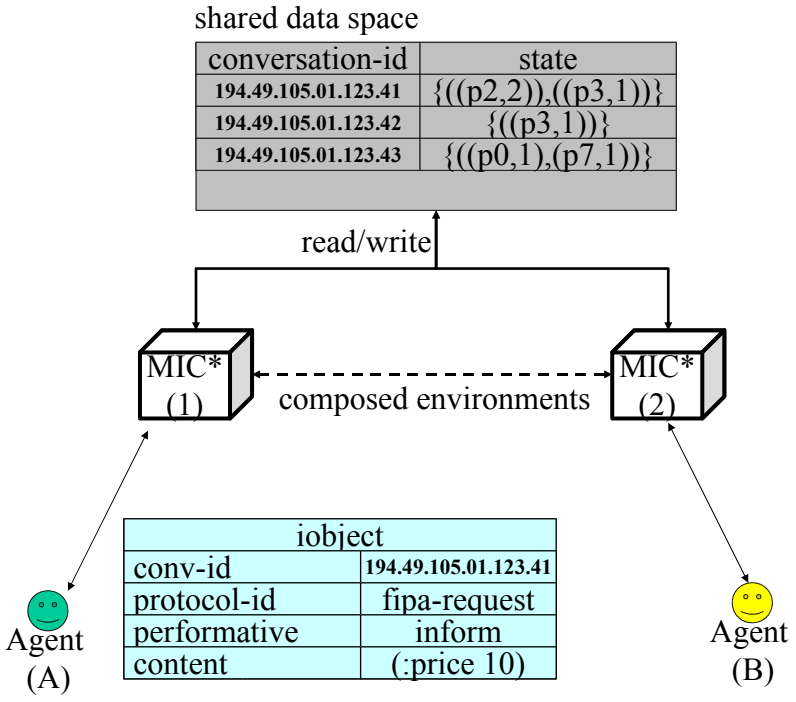


Figure 4.14: The global control of the coordination process implies a shared-memory to manage the states of the conversations. This approach contradicts the assumptions made on the properties of targeted distributed systems.

However this approach contradicts the assumptions of our work. In fact, the assumptions that were previously made on the targeted software systems such as openness, dynamical composition and locality of interactions made the maintenance of a globally shared memory or data space impossible. So, this approach is usable only when there is a single closed DE that cannot be composed with other environments.

4.9.2 Local and Decentralized Monitoring of the Coordination Process

To be consistent with our work assumptions, one has to provide a solution that does not require sharing memories among the DEs (cf. Figure 4.15). So, the DE controls the validity of the conversations of its local agents.

The global coordination graph is subdivided into several coordination sub-graphs. Each sub-graph represents the coordination graph of a single role. The role sub-graph includes all the activities of the role and their related produced and consumed resources. Besides, for each produced resource found in the roles-cut the role coordination sub-graph models the fact that this resource is consumed by another role's activity by introducing a virtual activity. Without these virtual activities, the generated QPN would have some problems since the token of the produced resources are never decremented. Figure 4.16 shows how the ping-pong coordination protocol is divided into coordination sub-graphs representing the 'ping' and 'pong' roles. Figure 4.17 shows the corresponding QPNs.

Obviously, the DEs have to implement some association mechanisms to join the distributed parts of the coordination graph. For instance in Figure 4.16, the resource node 'Ping' of the coordination sub-graph (b) should be linked to the resource node 'Ping' of the sub-graph

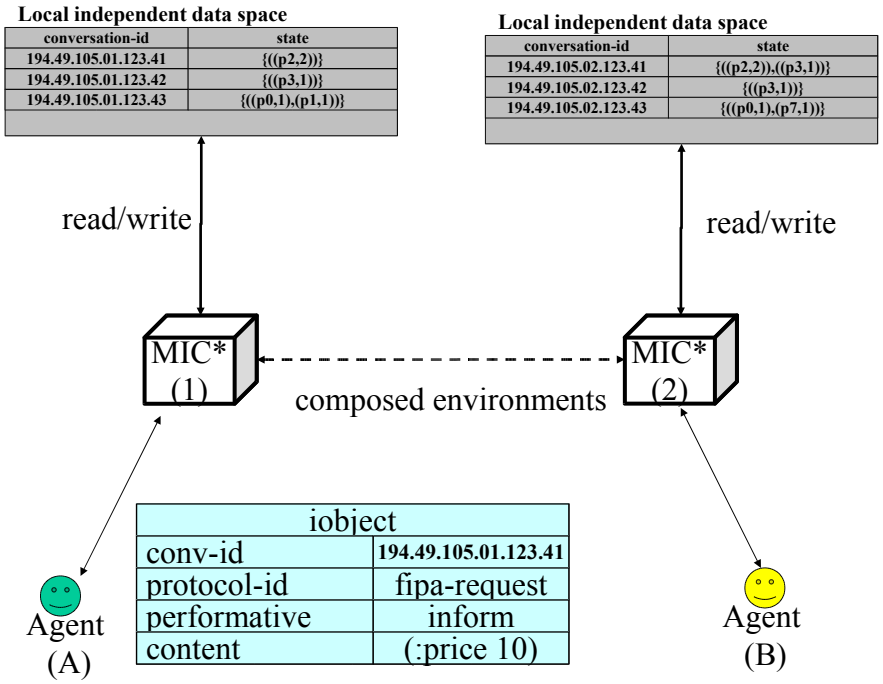


Figure 4.15: Decentralized control of the coordination process implies only the management of local independent memories.

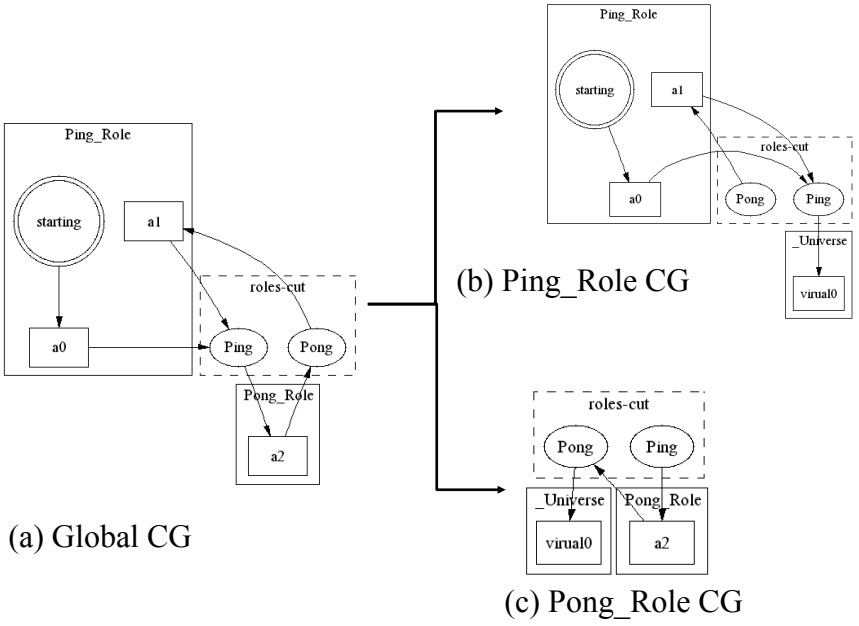


Figure 4.16: Example of the decomposition of the global coordination graph into several coordination sub-graphs.

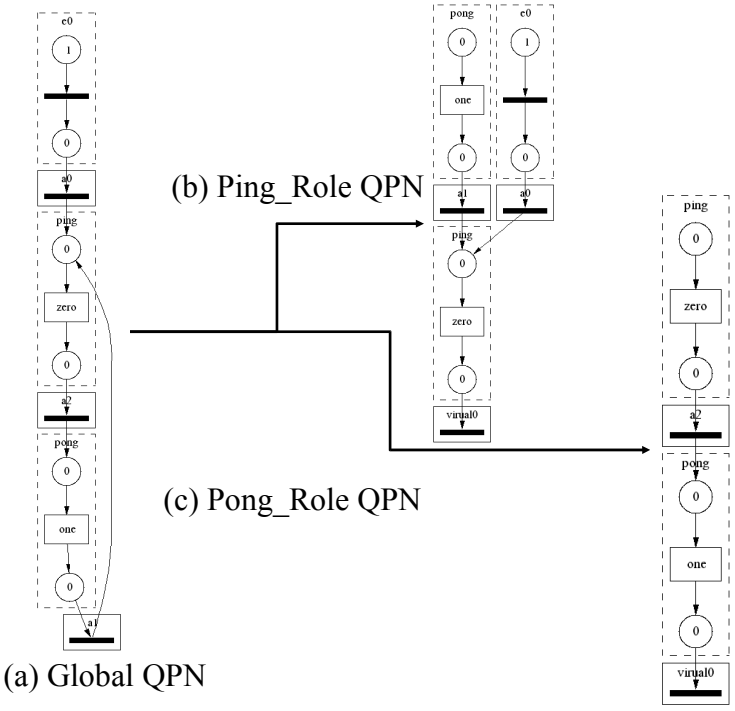


Figure 4.17: The QPN of the global coordination graph of Figure 4.16 and its decomposition on roles.

(c). Without this association mechanism, the coordination sub-graphs that do not contain the axiom resources are always 'dead' as it is the case of the sub-graph (c). This association is introduced later in the definition of the interaction operators that handles the interaction among the interaction objects.

The `CoordinationMessage` type is introduced in order to include the fields describing the used coordination protocol, namely the `protocol-id` field, and the unique identifier of the conversation, namely the `conversation-id` field. Other fields used for routing the interaction objects among the agents are also used such as the `sender-role` and `receiver-role` for social-level routing and `receiver-id`, `sender-id` for agent-level routing (cf. §3.5.1, page 84).

The monitoring of the coordination process is performed in two phases. The first phase concerns the emission of the interaction objects by agents. The question here is to know whether the agent is allowed or not, depending on the state of the conversation, to emit an interaction object. The second phase concerns the reception of the interaction objects. The question here is to know whether an agent is allowed or not, depending on the state of the conversation, to receive an interaction object.

Within the MIC* model, the control of interaction objects emitted by the agents is performed using the filter operators (cf. §3.4.5, page 82). The filter operator is a unary function that alters the interaction objects emitted by the agent. A special filter operator named `CoordinationFilter` is defined on the following domain: `CoordinationMessage` → `CoordinationMessage`. The algorithmic definition of this operator is presented in Algorithm 5.

This coordination filter takes as argument the `CoordinationMessage` interaction object. The first step (lines 2 to 4) retrieves some useful information from the interaction object such as: the conversation unique identifier, the identifier of the coordination protocol, and

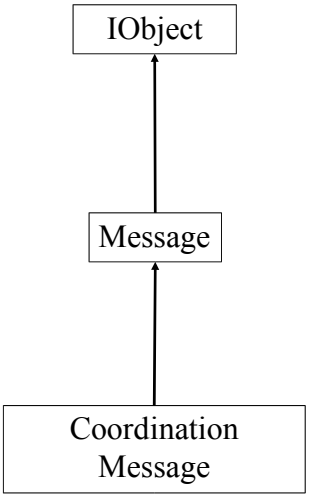


Figure 4.18: Introduction of the `CoordinationMessage` type in the type hierarchy of the social framework.

Algorithm 5 Algorithmic definition of the filter function to control the emitted messages from an agent during a coordination process.

```
1: function CoordinationComputationFilter(msg)
2:     convID ← msg['conversation-id']
3:     protoID ← msg['protocol-id']
4:     roleID ← msg['sender-role']
5:     if not knownProtocol(protoID) then
6:         return 0
7:     end if
8:     conversation_state ← getConvStateById(convID)
9:     if conversation_state = 0 then
10:        conversation_state ← getInitConvState(protoID, roleID)
11:    end if
12:    valid ← IsValidConversation(msg,conversation_state)
13:    if valid then
14:        updateConversationState(convID,conversation_state)
15:        return msg
16:    else
17:        return 0
18:    end if
19: end function
```

the played role of the emitter. After this initialization phase, the operator checks if the coordination protocol is registered within the interaction space (line 5); if not the operator returns 0. This means that the computation of the agent has been canceled.

Obviously, the implementation provides a mechanism in order to inform the agent about the reason of the failure. Line 8 retrieves the state of the conversation by its identifier. If the conversation was not present in the local data space of the DE, this function returns a null value. When a null value is returned, this is interpreted as the creation of a new coordination process. So, a new initial conversation state is created by line 20.

To create the initial state of the conversation the name of the coordination protocol and the role of the emitter are used. Notice that the state of the conversation is represented by the set of QPNs generated from the coordination graph. Initially, this set contains only one QPN.

Line 12 checks the validity of the conversation by considering its current state and the incoming interaction object. When the conversation is validated, the conversation state parameter is modified (call by reference). Otherwise, this parameter is not changed.

If the conversation is validated, then the entry of this conversation is updated in the local data space and the interaction object is returned. Otherwise an empty interaction object is returned to the MIC* DE. When an empty interaction object is returned, MIC* does not follow the computation evolution law by an interaction law. In other words, if an empty interaction object is returned by the filter operator nothing happens and the state of the DE is unchanged. On the other hand, if a non-zero interaction object is returned, the MIC* DE follows the computation of the agent by an interaction evolution law within the same interaction space.

The second step checks whether an agent is authorized to receive the interaction object. This is performed using the interaction operator: `CoordinationInteractionOperator`. This operator is defined algorithmically by Algorithm 6. It takes as argument the *sensor* interaction object an instance of `SocialRole` or `AgentIdentifier` type; and the $effector$ interaction object an instance of the `CoordinationMessage` type.

The initialization phase (line 2 to 7) retrieves some useful fields from the interaction objects. Lines 8 to 17 check if the receiver's *sensor* is able to perceive the sender's $effector$ using only the routing information. If the sent message cannot be perceived by the *sensor* at the routing level, the interaction function returns a zero. If the routing information are correct then the interaction function gives the control to the coordination level.

The state of the conversation is retrieved from the local data space using the conversation unique identifier (line 18). If the conversation was not present in the local data space, a new conversation state is created for the role of the receiver.

The initial conversation state is the QPN generated from the sub-coordination graph of the receiver's role (line 20). Since, the supervision of the coordination process is performed in a distributed manner, one needs to associate the sub-coordination graphs in order to rebuild the original coordination graph. The `joinRessources` function performs this operation: it takes as arguments the name of the coordination protocol, the emitter role, the receiver role and the emitted interaction object. The `joinRessources` function checks the list of resources that can be exchanged from the producer role to the consumer role and identifies the concerned resource using the control functions. When the resource node has been identified, the input place of the resource block is incremented by one. Consequently, the block of this resource

Algorithm 6 Algorithmic definition of the interaction function to control the reception of messages during a coordination process.

```
1: function CoordinationInteractionOperator(sensor,effector)
2:     convID ← effector['conversation-id']
3:     protoID ← effector['protocol-id']
4:     toRecvID ← effector['receiver-agent-id']
5:     toRoleID ← effector['receiver-role']
6:     senderRole ← effector['receiver-role']
7:     recvRole ← sensor['role-id']
8:     recvID ← sensor['agent-identifier']    ▷ If the interaction object is not an instance of
           the AgentIdentifier type this returns a null value.
9:     if toRoleID = recvRole then
10:        if toRecvID != 0 then
11:            if toRecvID != recvID then
12:                return 0
13:            end if
14:        end if
15:    else
16:        return 0
17:    end if
18:    conversation_state ← getConvStateById(convID)
19:    if conversation_state = 0 then
20:        conversation_state ← getInitConvState(protoID, roleID)
21:    end if
22:    conversation_state ← joinRessources(protoID,senderRole,recvRole,conversation_state,effector)
23:    valid ← IsValidConversation(effector,conversation_state)
24:    if valid then
25:        updateConversationState(convID,conversation_state)
26:        return effector
27:    else
28:        return 0
29:    end if
30: end function
```

becomes fireable and can pop the interaction object from the queue of the QPN. The next step is to check the validity of the conversation (line 23). If the conversation is valid then the conversation state is updated in the local data space and the $effector$ is returned as of the interaction result; otherwise a zero is returned.

4.9.3 Complete Example of the Distributed Monitoring of the Coordination Process

In order to show how the distributed monitoring of the coordination, we consider the ping-pong example. We assume that the system is composed by the following elements:

- m_1 is MIC* DE that contains an interaction space that represents the 'PingPong' group.
- the 'PingPong' interaction space contains all the operators described in the social framework (cf. §3.5, page 83), the coordination interaction operator and the coordination filter.
- m_2 is MIC* DE that contains also the social interaction space 'PingPong';
- m_1 and m_2 are composed.
- ping and pong agents: these agents are deployed respectively on m_1 and m_2 and play respectively the 'Ping' and 'Pong' social roles.

Initially, the ping agent starts the coordination process by emitting the following interaction object in the social interaction space 'PingPong':

CoordinationMessage	
`protocol-id`	PingPongCoordinationProtocol
`conversation-id`	1000000F
`sender-role`	PingRole
`sender-id`	agent_01
`receiver-role`	PongRole
`content`	'0'

Since the emitted interaction object is instance of the `CoordinationMessage` type, the m_1 DE applies the coordination filter presented in Algorithm 5. The 'PingPongCoordinationProtocol' is assumed to be a registered protocol within the interaction space, so the line 5 of Algorithm 5 returns true.

However, the result of the function call in line 8 of Algorithm 5 is a null value since the local data space of m_1 does not contain any information about the conversation identified as `1000000F`. So, the initial state of the sub-coordination graph corresponding to the 'PingRole' is built. This initial state is returned by the function call of line 20, Algorithm 5. The initial state contains only one QPN that corresponds to the coordination graph (b) in Figure 4.17. The queue of this QPN is appended with the interaction object and the validity of the conversation is checked using Algorithm 3.

The result of this algorithm is presented in Figure 4.19. The conversation is valid since the QPN has emptied the queue of messages. Consequently, the emitter agent has the right to emit the coordination message. The filter operator returns the initial message as result and stores the state of the conversation in m_1 's local data space.

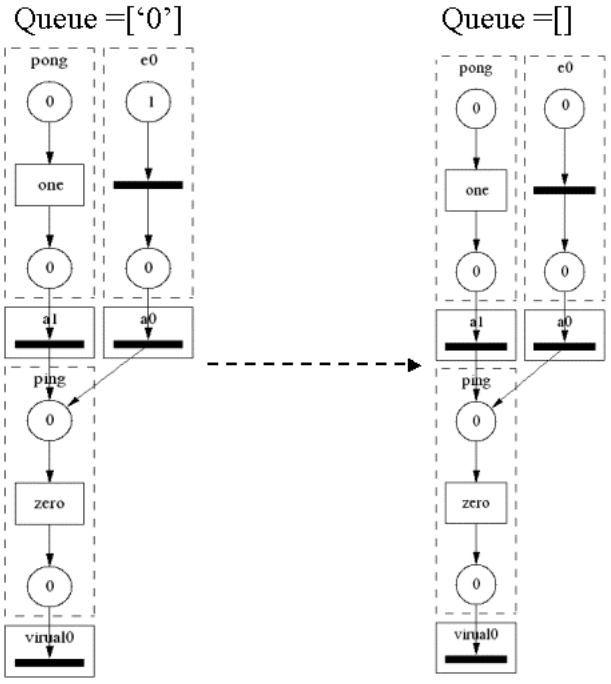


Figure 4.19: The initial and final state of the QPN associated to the 'PingRole' sub-coordination graph.

Since the filter operator has returned a non-zero interaction object, m_1 follows the computation evolution law by an interaction evolution law.

However, since no other agent is present locally in the interaction space, the interaction is forwarded remotely to the peer environment m_2. When m_2 receives the request for the remote interaction from m_1, it tries to find an interaction operator with the matching types.

The coordination interaction operator is a good candidate since it satisfies the constraints of types. So, a call is made to this operator to get the result of the interaction. The routing information of the coordination message are correct (lines 8–17 of Algorithm 6), so the interaction operator passes to the coordination level.

Once again, the operator asks to get the state of the conversation using its identifier. But, since the conversation has just started in m_2 a null value is returned.

The initial state of the conversation is requested but this time it concerns the receiver role, namely the 'PongRole'. The initial state corresponds to the graph (C) of Figure 4.17.

Before checking the validity of the conversation, the resources of the producer and consumer roles are joined (line 22 of Algorithm 6). Figure 4.20 shows the result of the resource joining mechanism for the QPN of the 'PingRole' sub-coordination graph. The input place of the 'ping' resource is incremented by one. In fact, the control function of the 'ping' resource recognizes the coordination interaction object with `'0'` as value of the `'content'` field.

The final step is to check the validity of the conversation (line 23 of Algorithm 6). The initial and final states of the QPN are presented in Figure 4.21. The ping role sub-coordination graph has validated the conversation, so the consumer role is able to receive the interaction

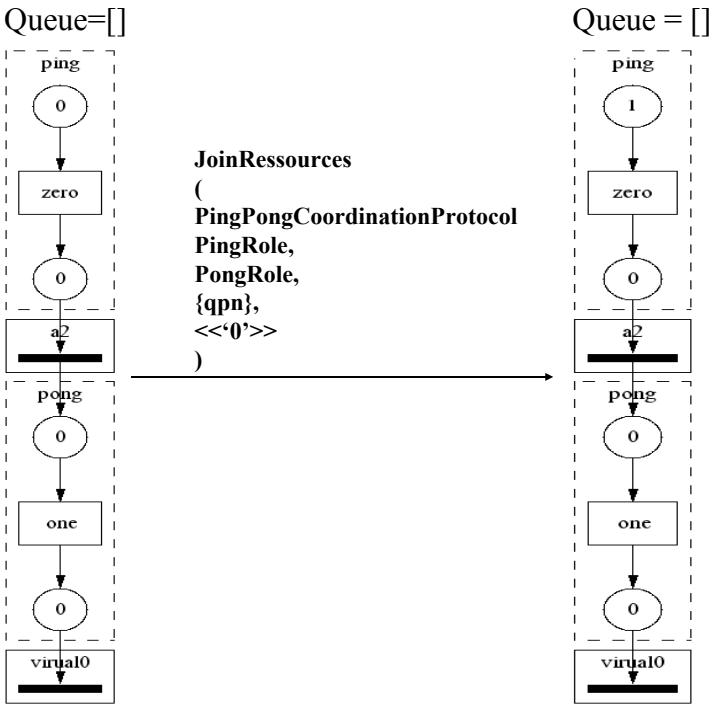


Figure 4.20: Application of the resource joining algorithm between the PingRole and the PongRole. The control function of the resource 'ping' has recognized the interaction object that contains '0' as content. Consequently, the input place of this resource has been incremented by one.

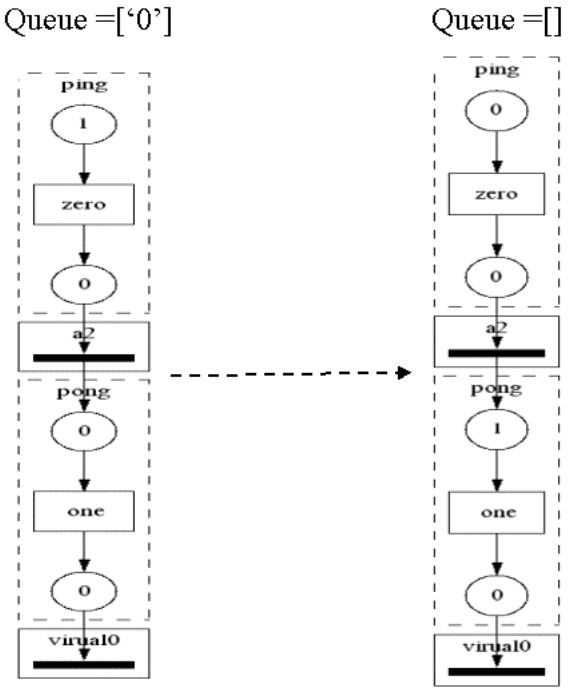


Figure 4.21: The initial and final state of the QPN corresponding to the 'PongRole' subcoordination graph.

object (line 23 of Algorithm 6). The state of the conversation is then stored in the local data space.

4.10 Conclusion

In this chapter we have presented an approach to design coordination protocols and to check their consistency by only monitoring the exchanged messages among the coordinating entities. The formalism of the coordination protocol is based on the dependency relationship between the activities and adapted to the MAS paradigm. This formalism is suitable to express parallel and simultaneous production or consumption of resources. The conversations that are generated by a coordination protocol are formally defined and practically validated using the QPNs. Consequently, by contrast to current MAS approaches, the structure of the conversations among the agents is not explicitly given but derived directly from the coordination protocol. The formalism to express the coordination protocols is independent on the number of the roles that are implied on the coordination process. In fact, currently the coordination protocols are generally expressed between exactly two roles. The DBCM can express the coordination process between more than two roles.

The DE, which is considered as the coordination medium, is responsible for monitoring the conversations according to the coordination protocols. Section [4.9, page 122] has presented the integration between MIC* and the coordination framework. Using the DE as a mean to control the coordination process has also been used in other works like in [Ricci *et al.* , 2004]. A. Ricci *et al.* suggest to define the DE as a set of *artifacts* that offer facilities to the agents. Hence to coordinate their joint activities the agents use a coordination artifact. The Agent Coordination Context (ACC) is then included in the coordination artifact to control the actions of the agents before their injection within the TuCSon tuple space. This principle is similar to the filter function that has been defined for the integration of the coordination framework and the MIC* DE.

In order to use to results of this chapter the next chapter presents a practical engineering framework aimed to design and build applications for the EAC context. This engineering framework uses the social metaphor and considers the applications of the EAC context as open and self-contained artificial societies of agents. The interoperability among the independent artificial societies is tackled from the coordination point of view. Hence, the interface of an artificial society is seen as the set of coordination protocols that this society defines to interact with the external world. The consistency between the agents' actions and the coordination protocols is a non-negligible feature to protect each artificial society from the actions that do not follow the established scheme of interoperability. Consequently, even if the artificial societies are defined in an open and dynamical context, such as the EAC context, they are still able to control and preserve the consistency of their design models at the run-time.

Chapter 5

Building Open Software Systems as Open Artificial Societies

"Il n'y a pas de recherche appliquée, mais des applications de la recherche"
Louis Pasteur.

5.1 Introduction

Since the early stages of DAI, distributed software systems have been intuitively considered as *artificial societies*. For instance, M. Fox in [Fox, 1981] compares distributed software systems to a human societies: in both of these systems autonomous individuals, that are rationally bounded (cf. §2.5.1, page 38), interact and coordinate their actions to achieve some personal or collective goals. To illustrate this sight, M. Fox describes the HEARSAY system as an organised artificial society of heterogeneous experts. These experts interact through a blackboard architecture in order to achieve the systemic function that is the speech recognition.

Carl Hewitt, who has introduced the Actor model, has also been inspired by the social metaphor. For instance, one can cite the following statement taken from Hewitt's paper [Hewitt, 1976b]:

> *"...we present an approach to modeling intelligence in terms of society of communicating knowledge-based problem solving experts."*

So, the actors are viewed as individuals that populate an *artificial society* and coordinate their activities by exchanging messages. The accomplishment of the systemic functions is a side effect of the interactions that hold between the actors.

These pioneer works have only introduced the idea of viewing distributed software systems as artificial societies. Still, the organizational structure of artificial societies remains implicit. In fact, there is not a clear separation of concern between the abstract level that organizes the society and the concrete level that defines the individuals.

To illustrate this decomposition, let us consider the example of a factory. The factory is a society that owns some systemic functions such as transforming some resources into goods while respecting the environment and offering good conditions of work to the employees. To achieve these functions, each individual owns its proper goals. An external observer of the factory can observe only the individuals that work in the factory. The behaviors of these

individuals can be explained only when considering the abstract level of the society that *organizes* the concrete level in a way that helps to achieve the societal functions. For instance, some individuals are organized in a supply chain in order to optimize the production of goods. The interaction of the individuals is also organized in order to optimize the flow of information and to protect some confidential data.

The MAS paradigm clearly distinguishes between the abstract and concrete levels. The concrete level is the observable part of the society. Obviously, the elements of the concrete level are steered and organised by the abstract level. The MAS paradigm has also provided organizational abstractions to specify the organisation of the concrete level such as roles, goals, groups, tasks, coordination and so on.

The discernment between the abstract and concrete levels has also separated the concerns for the engineering of MASs. On one hand, the abstract level specifies the organizational structure that organizes the artificial society. On the other hand, the concrete level is concerned with the design and the implementation of the individuals that populate the artificial society. The objective of this chapter is to present a practical engineering framework that follows this decomposition of concerns. In this engineering framework we target the EAC context by considering open software systems as open and interacting artificial societies. So, the openness of the artificial society is explicitly handled at the abstract level. In other words, each artificial society is aware about its openness and describes, at the organization level, how it can interact and coordinate its actions with other open societies. Besides, the organizational specifications of the abstract level are translated *automatically* into software structures that represents the *social deployment environment*. So, the consistency between the abstract level and the concrete level is continuously controlled. This is an essential feature for the EAC context to control that the actions of autonomous agents are consistent with the design specifications of the artificial society.

The rest of the chapter is organized as follows: section [5.2, page 134] reviews existing organizational models used to specify the organization level of MASs; in section [5.3, page 151] we present our proposition of an organizational model that targets the EAC context; section [5.4, page 158] presents the translation of the organizational specifications into the MIC* DE; section [5.5, page 160] presents a concrete application that has been entirely built using the presented engineering framework; section [5.6, page 170] presents the simulation platform that has been developed in order to experiment with EAC applications; section [5.7, page 174] presents briefly an open project that aims to gather different services that are deployed in the EAC simulation platform; finally, section [5.8, page 174] concludes the chapter.

5.2 State of the Art

Several organizational models have been proposed for the specification of MASs. Most of these models are provided within an engineering methodology[1]. This state of the art is concerned only with the organizational models and does not describe the methodological aspects. The organizational models are studied according to the following criteria:

- **organizational topology**: within an artificial society each individual is able to interact with a certain number of other individuals. This defines its *acquaintances* and the

[1] An engineering methodology is a predefined sequence of tasks and documents steering the engineering process.

acquaintances of all the individuals define the *topology* of the organization. The topology can be modeled as a directed graph where nodes represent the individuals and arrows the acquaintance relationship. This criterion studies how the topology of the artificial society is specified.

- **responsibilities/functions of the individuals**: the individuals populating the artificial society are autonomous agents. These agents are committed[2] to accomplish some functions within their society [Jennings, 1993]. Without this commitment, it is not possible to build a society that behaves coherently as a whole. This criterion studies how the agents' social functions are specified.
- **coordination of the individuals**: the individuals are aggregated in order to accomplish jointly functions that cannot be accomplished when the individuals are taken apart (cf. §2.5.1, page 38). This criterion studies how the coordination of joint activities is specified.
- **openness of the artificial society**: the term 'open' is overloaded and used to indicate different properties of a software system. To avoid ambiguities, in the rest of the chapter the openness of a system refers to its ability to interact with *external* systems. This interpretation contrasts with some interpretations found in the literature of MAS and that relates the openness of a system as its ability to change its own structure by either accepting or rejecting some elements (cf. §5.3.2, page 152). This criterion studies how the organizational model specifies the openness of the artificial society.

5.2.1 AAII

AAII is an engineering methodology proposed by D. Kinny *et al.* in [Kinny *et al.* , 1996] for building BDI MASs. AAII decomposes the MAS in two views:

1. external view: this view specifies the organizational structure of the artificial society in term of roles and conversation protocols.
2. internal view: this view studies the internal architecture of the agents following the BDI model.

Organizational Topology

The topology of the society is defined by the roles and their interdependencies. The conversation protocols specify how to access the functions offered by each role.

Responsibilities/Functions of the Individuals

The role and interaction models express the functions of the individuals. To obtain these models, AAII suggests the following steps:

1. identification of the main roles within the system.

[2]The commitment of an agent to implement a certain function does not imply necessary that this agent will implement it actually. This is due to the autonomous property of agents (cf. §2.3.2, page 28).

2. for each role one has to define the offered services and the interactions that are necessary to offer these services.

3. for each identified service one has to express explicitly the implied interactions by specifying the: *(i)* conversation protocols, *(ii)* activation events and *(iii)* some guard conditions that guarantee the consistency of the system.

Coordination of the Individuals

The coordination of the joint activities is addressed from a conversational point of view (cf. §4.2.4, page 96). In fact, the interaction model describes the conversation protocol used in order to access each function offered by a role.

Openness of the Artificial Society

The AAII is focused on building closed BDI MASs and does not consider the openness of MASs at the organizational level.

5.2.2 AGR/Aalaadin

Ferber and Gutknecht have proposed the Agent, Group, Role (AGR) model in [Ferber & Gutknecht, 1998]. The MadKit platform [Gutknecht & Ferber, 2000b] was then developed as a framework to implement the AGR concepts. The main organizational concepts of AGR are described as follows:

- group: the MAS is structured by the groups. A group is defined as a set of agents that play specific roles. It is important to notice that the agents can interact only when they belong to the same group.
- role: the role represents an abstraction of the function of an agent within a group.
- agent: the authors of the AGR model deliberately did not impose any constraint or assumption on the internal structure of the agents. Historically, this was a major step in order to separate the organizational aspects of MASs from the internal model of its constituents.

Organizational Topology

The topology of the MAS is structured by the set of groups and roles. Indeed, each agent that plays some roles is able to interact with the agents that play some roles that are member of the same group.

Responsibilities/Functions of the Individuals

The role concept specifies the functions and responsibilities of the agents.

Coordination of the Individuals

AGR adopts a conversational view of the coordination. The interaction is defined among the roles and conversation protocols describe the pattern of messages exchanged during a joint activity.

Openness of the Artificial Society

While AGR does not explicitly address this issue at the organizational level, the MadKit platform offers some services in order to distribute an artificial society among several sites or to merge distributed artificial societies that are called *communities*.

5.2.3 AUML

AUML (Agent Unified Modeling Language) is an adaptation of the UML notations to the MAS paradigm proposed by J. Odell *et al.* [Odell *et al.* , 2000]. The AUML notations and especially the one concerning the conversation protocols have gained some acceptance by the MAS community and are used in several agent-oriented software engineering methodologies.

Organizational Topology

The topology of the artificial society derives from the definition of the roles and their conversation protocols.

Responsibilities/Functions of the Individuals

The functions and responsibilities of the individuals derive from the definition of their roles and conversation protocols. Besides, the behavior of the agents can be specified using statecharts and activity diagrams.

Coordination of the Individuals

AUML provides both a conversation and activity view of the coordination. In fact, the coordination of a joint-activity can be specified by a conversation protocol either using the sequence diagrams or state charts. The coordination of a joint-activity can also be specified by the flow of its activities using the UML coordination diagrams.

Openness of the Artificial Society

The openness of the artificial society is not explicitly addressed by AUML. However, AUML conversation protocols can be used to define the interface of a MAS with the external world.

5.2.4 CoMoMas

CoMoMas has been proposed by Glaser in [Glaser, 1996] as an extension of the CommonKADS methodology [Schreiber *et al.* , 1994]. According to the state of the art presented by C. Iglesias *et al.* [Iglesias *et al.* , 1999], this methodology defines the following models:

- the agent model: this is the central model of the methodology that specifies the agents' architecture and knowledge.
- the expertise model: this model describes the cognitive and reactive competences of the agents.
- the task model: this model describes the functional decomposition of the system in term of a hierarchy of tasks.
- the cooperation model: this model describes the cooperation between the agents using conflict resolution methods and cooperation means. This model defines the communication primitives, the conversation protocols and the ontology of conversations.
- the system model: this model defines the organizational aspects of the agent society such as the roles.
- the design model: this model makes a synthesis of the specifications before the implementation of the system.

Organizational Topology

The system and cooperation models determine the topology of the artificial society since they describe the individuals and their interdependencies.

Responsibilities/Functions of the Individuals

The functions of the individuals are specified by:

1. the agent model
2. the task model
3. the expertise model.

The expertise model presents the task allocation that specifies for each agent what task to perform.

Coordination of the Individuals

The cooperation model uses the conversation protocols as a mean to resolve conflict among interdependent roles.

Openness of the Artificial Society

The openness of the MAS is not addressed at the organization level by CoMoMas.

5.2.5 GAIA

The GAIA [Wooldridge *et al.* , 2000][Zambonelli *et al.* , 2003] approach considers the MAS as an organized society of agents. Each agent plays a specific role and interacts with other agents using a predefined conversation protocol. The GAIA methodology concerns the engineering phases starting from the collection of requirement until the detailed design without addressing the implementation phase.

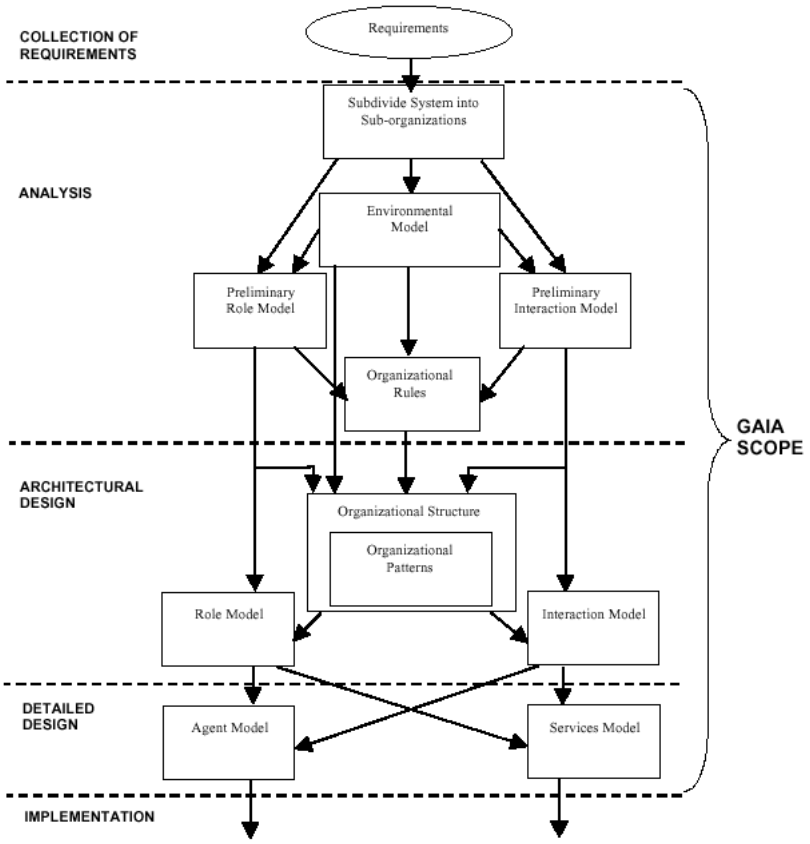


Figure 5.1: The GAIA engineering methodology phases [Zambonelli *et al.* , 2003].

The steps of the GAIA methodology are described as follows (cf. Figure 5.1):

- collection of requirements: during this phase all the necessary information concerning the functionalities of the desired software system and of its environment are collected and studied in order to prepare the analysis phase.
- analysis: during this phase the global MAS is subdivided into several sub-organizations and the preliminary roles, environment and interaction models are sketched.
- architectural design: the preliminary models of the previous phase are refined during this phase. The refinement process may imply the identification of new roles and interaction protocols. The refined models establish the final organizational structure.
- detailed design: this phase describes in detail the model of the agents that implement the described roles.

Organizational Topology

The topology of the artificial society is defined in GAIA by the roles and their associated conversation protocols. In fact, the conversation protocols are defined among couple of roles and give the acquaintances of the agents.

Functions/Responsibilities of the Individuals

Within GAIA the functions of the individuals are captured by the roles.

Coordination of the Individuals

GAIA handles the coordination of the agents from a conversational point of view. So, in order to coordinate their joint activities, the agents follow the conversation protocol that is defined between their roles. The conversation protocol specifies the structure of the dialog that holds among the agents. However, GAIA does not provide the formalism to describe the conversation protocols. Only a proposition was made in [Zambonelli *et al.* , 2003] to use the AUML sequence diagrams as a formalism to represent the conversation protocols.

Openness of the Artificial Society

The GAIA models do not express the openness of the artificial society at the organizational level. In fact, the interaction is defined only among internal agents and the organizational models do not explicitly specify how the artificial society interacts with the external world and especially with other artificial societies.

5.2.6 MACE

The Multi-Agent Computing Environment (MACE) [Gasser *et al.* , 1987] is a pioneer multi-agent platform developed by L. Gasser *et al.* as a testbed for DAI applications.

Organizational Topology

Within MACE the organizational model does not define the topology of the artificial society. The topology it is determined either statically by the class of the agent or dynamically by adding new acquaintances for the agent at runtime using an agent directory service.

Responsibilities/Functions of the Individuals

Each MACE agent owns a set of models about itself and other agents. The model of an agent contains:

- general information: such as the name, class and the address of the agent.
- the played roles: MACE defines a role as follows:

  "*a role is an accepted shorthand for a set of expectations.*"

- the skills: representing the capabilities of the modeled agent to achieve some goals.
- the goals: representing what are the state of affairs that the modeled agent want to reach.
- the plans: representing the sequence of actions that the agent has to follow in order to achieve its goals.

Coordination of Individuals

The coordination of the joint activities is not treated at the abstract level within MACE. The agents at the concrete level directly implement the coordination means.

Openness of the Artificial Society

MACE does not address the openness of the artificial society at the organizational level.

5.2.7 MaSE

The Multi-agent Systems Engineering (MaSE) is a software engineering methodology proposed by M. Wood and S. Deloach in [Deloach, 2001][Wood & DeLoach, 2000].

Organisational Topology

The role and interaction models define the set of roles, their interdependencies and the conversation protocols that regulate the conversations among the agents. Consequently, the acquaintances of the agents are completely defined by these models.

Responsibilities/Functions of the Individuals

MaSE uses the goals and roles to model the functions and responsibilities of the agents. The analysis phase sketches the goals that are expected from the software system. The goals are then organized hierarchically. The roles model is derived from the goals hierarchy and expresses also the independences among the roles. Finally, the tasks model associates each role with a set of goals to achieve.

Coordination of Individuals

The coordination of the individuals is addressed from a conversational point of view. Indeed, the interaction model specifies the conversation protocols among the interdependent roles using finite state automata.

Openness of the Artificial Society

The goal of MaSE is to entirely build a software system as an artificial society of agents. Still, the software system is closed and the interactions with the external world are not explicitly handled at the organizational level.

5.2.8 MASSIVE

MASSIVE [Lind, 2000] is an engineering methodology based on the concepts of *features* and *views* which are defined as follows:

- features: the features are divided in two categories:
  1. features of the model: these features concern the design decisions, the constraints that should be handled and the tests that should be performed;
  2. features of the implementation: these features concern the properties of the software components, particular equipments or a fragment of code.

  MASSIVE models the system to be produced and its implementation as a collection of features that are interconnected by labeled links. This is known as the *features graph*.
- views: the view is an abstraction of a set of features that are conceptually linked. It is a semantic projection of the system's features graph on a particular matter. The set of all views define the *system view* of the software system.

The concepts of features and views are the basis of the aspect-oriented programming (AOP). So, the MASSIVE approach gives a particular view using the AOP concepts that is suitable for the MAS paradigm.

The MAS system views proposed by MASSIVE are composed by the following views:

1. task view: this view handles the functional aspects of the system. The system functions are translated into a hierarchy of tasks. MASSIVE also models non-functional requirement of the system in this view.
2. environment view: this view is composed of two sub-views: the environment view from the developer perspective and the environment view from the agents perspective. In fact, while the developer has a global view of the environment of the MAS; the software agents have only a partial view of this environment.
3. roles view: MASSIVE defines the roles as the means that abstract the tasks. Consequently, this view translates the tasks that were defined in the task view in term of roles.
4. interaction view: this view analyses the communications that occur between the agents and specifies them in term of conversation protocols.
5. social view: MASSIVE defines the society as a structured aggregation of entities that have a common objective. This view studies the organizational aspect of the artificial society.
6. architectural view: this view studies the software architecture that is used to implement the MAS.
7. system view: this view captures the features that affect several views or the entire system.

Organizational Topology

The topology of the artificial society is specified in the interaction view. This view defines the conversation protocols that establish the acquaintances of the agents.

Functions/Responsibilities of the Individuals

MASSIVE defines two levels of functions: the functions that are expressed during the requirement phase; and the internal functions of the system expressed as tasks. These tasks are then mapped into roles in the role view.

Coordination of the Individuals

MASSIVE adopts a conversational point of view of the coordination. Indeed, the coordination of joint activities is performed through the conversation protocols which are defined in the interaction view.

Openness of the Artificial Society:

MASSIVE does not address the openness of the artificial society. This methodology is focused on building closed software systems without considering the interaction with other artificial societies.

5.2.9 MAS-CommonKADS

The MAS-CommonKADS [Iglesias *et al.* , 1996] is a software engineering methodology that extends the CommonKADS [Schreiber *et al.* , 1994] for MASs modeling and design. This MAS-CommonKADS defines the following models:

- agent model: this model specifies some characteristics of the agents such as their reasoning capabilities, services and hierarchy. This model also specifies the groups of agents.
- task model: this model describes the tasks carried out by the agents.
- organization model: this model describes both the organization where the MAS is integrated and the organization of the artificial society of agents.
- coordination model: this model specifies the interactions that occur between the agents and between the human users and the agents.
- design model: this model collects the previous models and defines three other submodels:
    1. network design: this model specifies the properties of the communication medium used by the agents.
    2. agent design: this model selects the most suitable architecture to design and implement the software agents.
    3. platform design: this model selects the most suitable agent development platform.

Organizational Topology

The organization model specifies the relationships among the agents. The coordination model precisely describes these relationships and specifies the interactions that occur internally between the agents and the interactions that occur between the human users and agents.

Responsibilities/Functions of the Individuals

Within MAS-CommonKADS the functions of the individuals are specified by the task and by the agent models. The tasks are decomposed following a top-down approach and described by an and/or tree. The agent model describes the services offered by each agent in order to achieve these tasks.

Coordination of Individuals

The coordination model specifies the conversation and coordination protocols using several notations such as the MSC, event flow diagrams or SDL state diagrams.

Openness of the Artificial Society

MAS-CommonKADS addresses some aspects of the openness of the artificial society. In fact, the interaction between the MAS and the human users are explicitly modeled in the organization model. Still, interactions with other artificial societies are not represented.

5.2.10 MESSAGE

MESSAGE [Caire *et al.* , 2001] is a software engineering methodology that extends the UML notations for MASs. The concepts of MESSAGE are classified in three categories: *ConcreteEntity*, *Activity*, and *MentalStateEntity*.

The concepts of the *ConcreteEntity* category are:

- Agent: an agent is defined as an autonomous entity capable of performing some functions. The functions of the agent are reified as *services*. MESSAGE uses the motivation based definition of the autonomy that was presented in section [2.5.2, page 41]. The agent's motivations are represented as *purposes*. The purposes steer the agents' behaviors to perform or not the services.
- Organization: the organization is defined as a group of agents that work together for a common purpose. The structure of the organization is expressed through the *power relationships*. The power relationship expresses a relationship between a superior and a subordinate.
- Role: the role describes the external characteristics of an agent. It is similar to the concept of the interface used within the OO framework.
- Resource: the resources represent all the non-autonomous and passive entities found in the system.

The *Activity* category concepts are as follows:

- Task: a task has a set of pairs of states describing the pre and post conditions. The task is performed when the pre-conditions are valid. When a task is performed and completed, the post-conditions are supposed to hold. The activities of the tasks are represented as finite state automata using the UML activity diagrams.
- Interaction and InteractionProtocol: the interaction is defined by MESSAGE as a collective action to: *(i)* reach a consistent view of some aspect of the problem domain; *(ii)* agree terms of a service; *(iii)* exchange the results of one or more services. The interaction protocol specifies the pattern of messages that are exchanged during the interaction process.

The *MentalStateEntity* concept is:

- Goal: The goal associates an agent with a situation or a state. If the goal instance is present in the agent's mental state, then the agent intends to reach the situation or state described by this goal.

Organizational Topology

The topology of the artificial society is expressed through the groups of agents, power relationships, and conversation protocols.

Responsibilities/Functions of the Individuals

The agents play one or several roles that determine the offered services. The goal of the agent is then to fulfill its associated services by performing the associated tasks.

Coordination of Individuals

MESSAGE addresses the coordination from a conversational perspective. Consequently, the coordination mechanisms are expressed as conversation protocols. Notice that MESSAGE differs from SODA on the definition of the resources. In fact, contrary to SODA (cf. §5.2.12, page 146), the interaction between the agents and resources is not described. The agents are simply assumed to access the resources directly as in OO architectures.

Openness of the Artificial Society

MESSAGE does not address the openness of the artificial society at the organizational level.

5.2.11 PASSI

PASSI (Process for Agent Societies Specification and Implementation) is an engineering methodology for designing and developing MASs using the concepts that derive from both the OO and MAS paradigms. PASSI is composed by five models that are: system requirements model, agent society model, agent implementation model, code model, and deployment model. Still, only the two first models are relevant for our study. These models are described as follows:

- system requirements model: this is an anthropomorphic model of the system requirements in terms of agency and purposes. This model is built according to the following steps:
  - domain description: this step presents a functional description of the system using conventional UML use-case diagrams.
  - agent identification: this step defines the functionalities of the agents as use-cases.
  - role identification: this step explores the responsibilities of the agents using UML sequence diagrams.
  - task specification: this step specifies the capabilities of the agents by using the UML activity diagrams.
- agent society model: this model specifies the interactions and dependencies among the agents. This model is composed of the following components:
  - ontology description: the knowledge ascribed to the agents and the pragmatics of the conversation are specified by UML class diagrams.
  - role description: the roles are specified by the list of tasks and conversation protocols they involve.
  - protocol description: the structure of each pragmatic conversation protocol is specified using UML sequence diagrams.

Organizational Topology

Within PASSI the topology of the artificial society is defined by the agent society model through the roles description that details the communication capabilities and expresses the dependencies among the roles.

Responsibilities/Functions of the Individuals

Within PASSI the functions of the individuals are specified by the roles and tasks.

Coordination of the Individuals

PASSI adopts a conversational view of the coordination. The conversation protocols are specified by the agent society model.

Openness of the Artificial Society

PASSI does not address the openness of the artificial society at the organizational level.

5.2.12 SODA

The SODA (Societies in Open and Distributed Agent spaces) [Omicini, 2001] is an engineering methodology complemented by a set of organizational models for the analysis and design of Internet-based applications. This approach argues that the notion of the society should play

a central role in agent-oriented software engineering and thus should be considered as a first-class abstraction around which complex software systems are designed and built. SODA distinguishes between the intra-agent aspects where the agents are studied as single entities aware about their environment and social abilities and the inter-agent aspects where the society and the environment on which it is embedded are designed and studied.

The SODA definition of the agent artificial societies: The agents are autonomous individuals with social abilities. The behavior of the agents is not understandable outside their social context. Similarly, the behavior of the overall society of the agents cannot be expressed in terms of the behaviors of its composing agents when taken apart. So, to fully understand a MAS, one has to consider both the individual aspects and the collective aspects of the MAS. Besides, the society of the agents is not a simple aggregation of the individuals. Some social rules and laws govern the agent interactions and drive the entire society towards the accomplishment of its design goals.

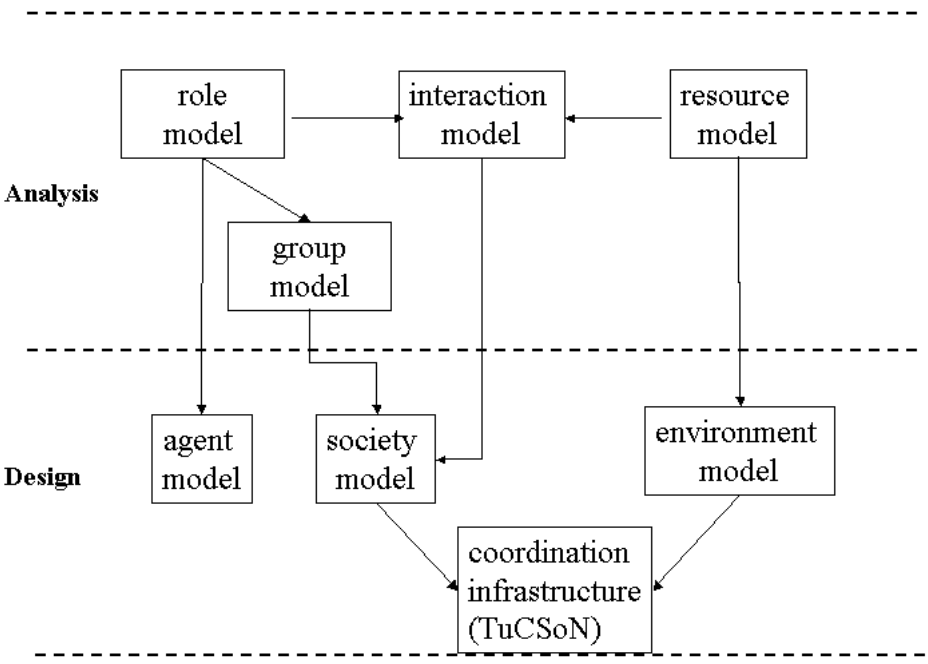


Figure 5.2: The SODA engineering methodology phases and models [Omicini, 2001].

The phases and models of the SODA methodology are described as follows (cf. Figure 5.2):

1. analysis phase: during this phase the application domain is analyzed, studied and expressed in terms of the following models:
   - role and group model: the elementary organizational concept used in SODA is the *task*. The tasks are expressed in terms of: responsibilities they involve, competences they require, and resources they depend upon. The responsibilities are defined as the states of affairs that result from the task accomplishment. The tasks are classified in two categories: individual tasks and social tasks. The individual tasks require the competences of a single individual agent while social tasks require the

competences and resources of a group of agents. Individual tasks are assigned to *roles*. On the other hand, the social tasks are assigned to the *groups*.

- resource model: within SODA the agents are situated in an environment that offers some resources. The concept of a resource is not formally defined, but it can be viewed as an abstract *service* offered by the environment to the agents. Each resource defines some *access modes* modeling the different ways in which the service of the resource may be exploited by the roles. To use the resources some *permissions* are granted to the roles and groups.
- interaction model: the interaction model of SODA is expressed by the conversation protocols and interaction rules. The conversation protocols define how to access both the services of roles and resources. The interaction rules regulate the conversations among the roles within a group.

2. design phase: this phase translates the previously presented models to design models that are:

   - the agent model: SODA does not define a detailed model of the agents since this is out of its scope. Only the observable behaviors are specified. This includes the interactions with other roles and resources. The agent class model specifies the tasks carried out by the agent, the set of permissions and the conversation protocols associated to the roles. Additional information are provided during this phase like the: *cardinality*, number of agents of the class; the *location*, either fixed for static agents or variable for mobile agents; and the *source* which indicates if the agent class is either local or imported in the case of a mobile agent.
   - the society model: each social group is mapped into a *society of agents*. The agent society is characterized by the social tasks it carries out, the set of permissions, the participating social roles, and the interaction rules associated to the groups. In order to implement the interaction rules, SODA societies are designed around a coordination medium such as TuCSoN (cf. §2.4.3, page 36). The coordination medium achieves the interactions among the entities.
   - the environment model: the resources are mapped into *infrastructure classes*. An infrastructure class is characterized by the services it offers, the resources access modes, the permissions granted to roles and groups and the conversation protocols defined to access the resources.

Organizational Topology

Within SODA the topology of the artificial society is determined by the set of roles, groups and the conversation protocols associated to each role and resource. The interaction among the roles is also governed by the interaction rules.

Responsibilities/Functions of the Individuals

Within SODA the functionalities of the individuals are modeled by the tasks. The tasks require competences and depend on resources. Two types of tasks are identified in SODA: individual tasks, which can be performed by a single agent; and social tasks that imply necessarily the coordination of a group of agents.

Coordination of Individuals

SODA adopts a conversational view of the coordination. In fact, a conversation protocol is associated with the roles in order to accomplish their assigned social tasks. The conversation protocols express the type and the order of the messages that should be sent when performing the social task.

Openness of the Artificial Society

The SODA approach does not define clearly what is meant by the systemic openness. In this chapter, we have given our own interpretation in section [5.3.2, page 152]. The organizational model of SODA does not define explicitly the interaction with other artificial societies. However, since the agents are also able to interact with the resources that may be found outside the artificial society this can be viewed as a property that make SODA societies open according to our interpretation of the openness.

5.2.13 TROPOS

The TROPOS [Giunchiglia *et al.* , 2001a] [Giunchiglia *et al.* , 2001b] [Perini *et al.* , 2001] is a software engineering methodology that considers the software system to be produced as an organized artificial society [Kolp *et al.* , 2002] [Fuxman *et al.* , 2001][Kolp & Mylopoulos, 2001] that is integrated in a pre-established organization [Giorgini *et al.* , 2003]. The TROPOS organizational concepts are derived from the *i** model presented by E. Yu [Yu, 1995]. The organizational concepts of TROPOS are as follows:

- actor: the actor represents a generalization of a physical or a software agent. The behavior of an actor is determined by its role.
- goal: the goal represents a strategic objective of an actor. Two kinds of goals are defined within TROPOS: concrete goals and informal goals. The concrete goals represent quantifiable objectives that are determined during the design phase and assigned to the actors. The informal goals represent non-functional recommendations that are expressed during the design phase and that should be taken into account when building the software system.
- dependence: an actor depends on another actor for the achievement of a goal, for the execution of a plan or for the acquiring of a resource. So, this relationship is composed of three attributes: the dependant, this is the actor that depends on the other actor; the dependum, this is the matter of the dependence; and finally the dependee, that is the actor that delivers the dependum to the dependant. The dependency relationship is the elementary building block used within TROPOS to specify the organization of the artificial society.
- plan: a plan represents a sequence of activities that should be executed in order to achieve a particular goal.
- capability: the capability reifies the ability of an actor to define and to choose a plan in order to fulfill a particular goal.
- belief: the belief reifies the knowledge of a particular actor about its surrounding environment.

Organizational Topology

The topology of the artificial society is implicitly defined by dependence relationships defined among the actors. In fact, the depender and dependee have to interact in order to exchange dependums. So, by collecting all the dependencies relationships defined among the actors, one can rebuilt the acquaintances network of the artificial society.

Responsibilities/Functions of the Individuals

The functions of the individuals are reified by the concepts of goals. The goals can be decomposed in sub-goals using or/and trees. The goals are finally assigned to the actors.

Coordination of Individuals

The TROPOS coordination is implicitly defined by the dependencies relationships. These dependencies are expressed among the actors and not among their activities as suggested by Malone and Crowston (cf. §4.2.1, page 93). Still, the TROPOS dependencies only give an indication about the coordination processes that will occur among the actors. TROPOS suggests the use of AUML sequential diagrams to express the conversation protocol that is used as a mean to solve dependencies on resources.

Openness of the Artificial Society

The idea of TROPOS is to build the software system as an organization that is integrated to a pre-existing organization. So, TROPOS can be considered as handling explicitly the openness of the artificial society at the organizational level.

5.2.14 Discussion

The state of the art review shows that there are several models for the specification of MASs as artificial societies. Most of these organizational models share the same organizational concepts with few differences on their definitions.

In most of the models the topology of the artificial society is expressed by the conversation protocols that are defined between the roles and that express the mean to solve the interdependencies among the roles. The notion of dependence among the roles is made explicit in the TROPOS model.

The concept of role is central to make the organisational specifications of the MAS independent from the agents. In fact, the role abstracts the functions of the agents. So, the role expresses *what* is expected from the individual and does specify *how* the function is actually implemented.

The coordination of the joint activities is expressed by using the conversation protocols.

The interesting point that is revealed by the state of the art is the fact that few organisational models express the openness of the artificial society using organisational abstractions. The openness of the artificial society is a fundamental property for the EAC context (cf. §2.3, page 24). Consequently, we need to develop an organisational model that provides some

organisational abstractions in order to specify MASs as self-contained[3] and open artificial societies.

5.3 Organizational Model for Open Artificial Societies

5.3.1 Proposition of an Organizational Model for (not yet open!) Artificial Societies

After reviewing the sate of the art of existing organisational model for artificial agent societies, we propose here a model that is suitable for self-contained and open MASs. This model is adjusted to fit the concepts presented in the previous chapters concerning the deployment environment (MIC*) and the coordination framework. This model is not a new proposition, but simply an adaptation of the concepts found in the state of art for the EAC context.

We follow the decomposition of the MAS artificial society in two levels: the abstract level and the concrete level. The abstract level concerns the organisational model that rules the individuals of the artificial society. The concrete level represents what is actually visible in the artificial society, namely the autonomous agents or *citizens* of the artificial society.

The Abstract Level

The abstract level is composed of four elements:

1. **organizational structure**: the organizational structure specifies the topology of the artificial society in terms of groups, roles and structural constraints.
   - roles: the roles are abstractions of functions that are implemented by the citizens. When playing a role, the citizen is committed to fulfill the associated functions and to follow the associated coordination protocols. The social environment continuously controls the consistency between the citizen commitments and its observed actions.
   - groups: the concept of group is introduced simply to create a context of interaction among the roles. So, a group is a set of roles that are able to interact to execute a joint activity.
   - constraints: the constraints express the cardinality of groups and roles.
2. **systemic functions**: this represents the functions that the artificial society offers to the external world. These functions are generally gathered in the requirements phase of the engineering process and are independent from the fact that the developed software system follows the MAS paradigm.
3. **mapping the systemic functions on the organizational structure**: the systemic functions are mapped into the organizational structure by the concept of *social task*. Each systemic function is translated into one or several social tasks. A social task implies the participation of one or more social roles to be accomplished. Besides, these roles must be defined in a social group in order to allow the agents playing these roles to interact.

[3]Here the expression 'self-contained MAS' means that the MAS defines a delimited system (cf. §2.2, page 19).

4. **coordination**: for each social task a coordination protocol is associated to explain clearly the coordination process of the joint activity. The used formalism to express the coordination protocols is the BDCM presented in chapter 4.

The Concrete Level

This level represents the autonomous agents that belongs to the artificial society. The agents are called the *intracives* or the internal citizens of the artificial society. It is important to identify clearly the intracives from other agents that belong to other societies. Without this property the artificial society cannot be considered as a delimited system since its frontiers cannot be clearly established. The artificial society has to be a delimited system in order to define its openness. Section [5.3.2, page 152] discusses this point more in detail. Each autonomous agent plays one or more social roles. When an agent plays a certain role, it is committed to fulfill all the implied social tasks and to follow their coordination protocols. If an agent does not fulfill this requirement, the social environment should be able to identify this as an inconsistent situation and to establish that the agent is responsible of challenging the social model.

5.3.2 Opening the Organizational Model

Basic Elements of Open Systems

First, let us introduce a general framework (inspired from Physics) modelling our systemic view. The first concept of this framework is the notion of *absolute universe*. The absolute universe is the overall container; everything is defined inside this container and nothing exists outside. Among the things that exist in the absolute universe, one may find *delimited systems*. A delimited system is defined as a any composite of things that can be *delimited* with a clear frontier. In other words, a delimited system is an aggregation of things that one may clearly define its *inside* and its *outside*. The frontier between the outside and the inside is called the *interface*. For a particular delimited system, the *relative universe* is defined as the set of elements of the absolute universe except the system itself.

The interaction among the systems is defined as the process of exchanging *influences* that change the internal state of the interacting systems. These influences pass necessarily through the interface.

As shown by Figure 5.3 a closed system owns an interface that prohibits the influences from getting inside or outside the system. Consequently, closed systems define interactions only between their internal components and no interaction is defined with the relative universe. The absolute universe can be seen as the biggest closed system and its relative universe is the void.

An open system is a delimited system where the interface allows (particular) influences to get in (get out from) the system (cf. Figure 5.4). Consequently, the open system can interact with the elements of its relative universe.

The openness of a system also depends on the definition of its containing universe. For instance, if we consider a simple universe composed of a human and of a pocket calculator (cf. Figure 5.5), then the pocket calculator is an open system since it interacts with the human. In

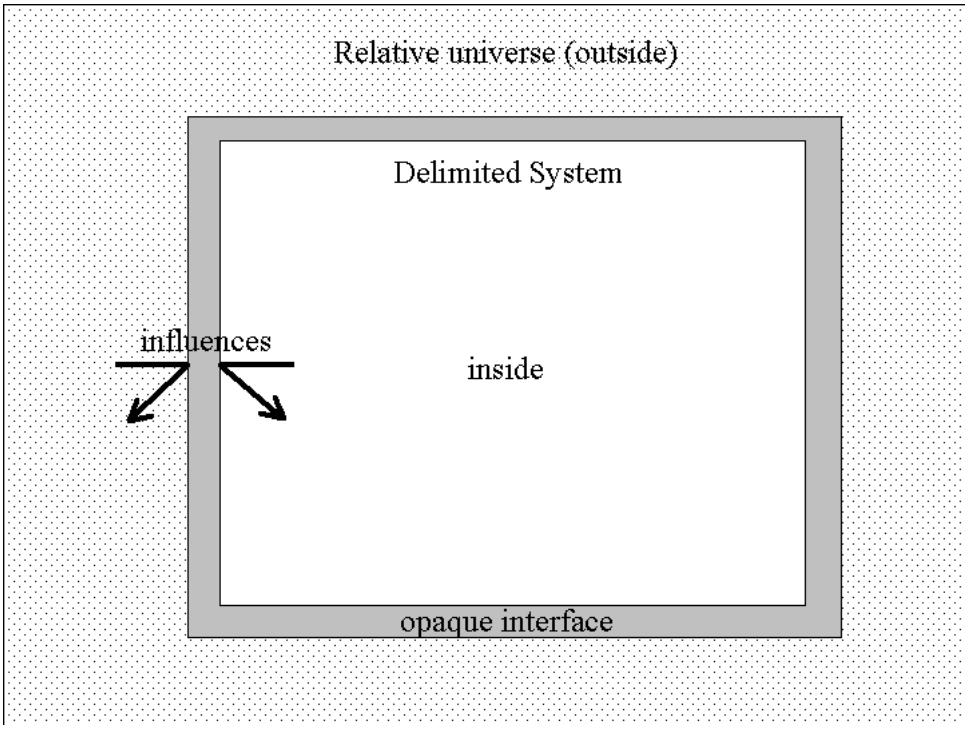


Figure 5.3: Representation of a closed delimited system: the interface prohibits influences from getting in/out the system. Closed systems do not interact with the elements of their relative universe.

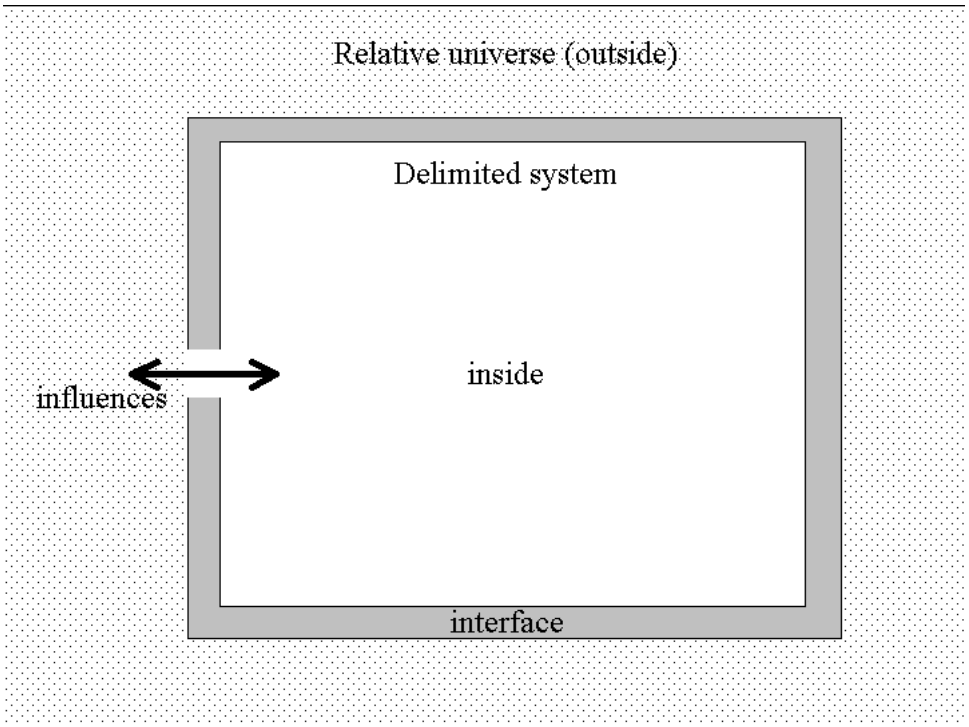


Figure 5.4: Representation of an open and delimited system: the interface allows influences from getting in/out the system. Open systems interact with the elements of their relative universe.

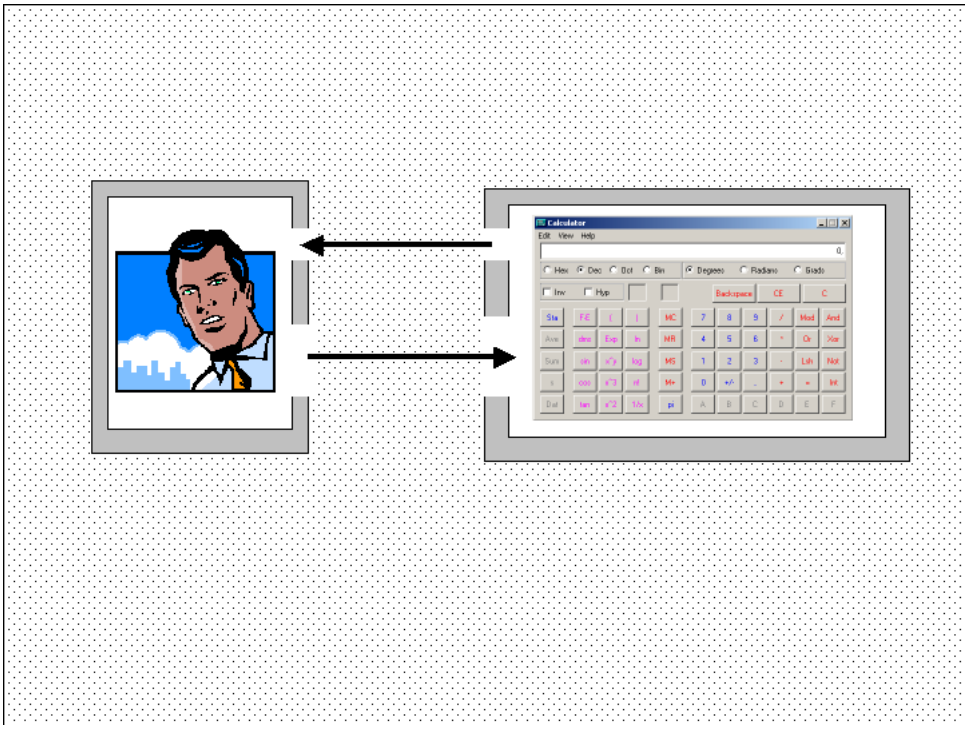


Figure 5.5: The systems containing the human and the calculator are open systems since their interaction is possible and passes across their respective interface.

fact, the human may change the values of the digital circuits of the calculator by pressing the keys and the calculator influences the perceptions of the human by displaying some results.

Now, let us take a universe composed of two calculators (cf. Figure 5.6): the calculators are closed systems since there is no interaction defined among them. So, when talking about an open system, one has to specify the exact context in which the openness property holds. For instance, the FIPA platforms are open systems only if we consider the absolute universe as the set of all platform following the FIPA specifications. But, when a FIPA platform is placed in an absolute universe with systems that do not follow the FIPA specifications it is considered as closed system since there is no interaction with the external world. This can be generalized to agents. For instance, it is not sufficient to consider that an agent defines an open system only when it is able to open a TCP/IP socket. One has to give more information on the requirements that make this agent able to interact with other agents or systems. For instance, by specifying the ontologies, languages, conversation protocols that are understood by this agent.

The openness property has also been given another interpretation within the literature of MASs. It is defined as the ability of a system to modify its internal structure dynamically by adding or removing agents. This is very different from our approach. For instance, let us consider an example of two systems 'A' and 'B'. The system 'A' is composed of two persons: Lylia and Fabien. Lylia and Fabien are two persons that both speak French; so they can interact. The system 'B' is composed of a person, Melinda, who speaks Romanian. These systems are closed since there is no interaction between them. Now, let us change dynamically the structure of the system 'A' by adding a new person, Christopher, who speaks also French in system 'A'. This dynamical modification of the structure of the system 'A' has not made it an open system! In order to make these systems open, one has to add a *bridge* person

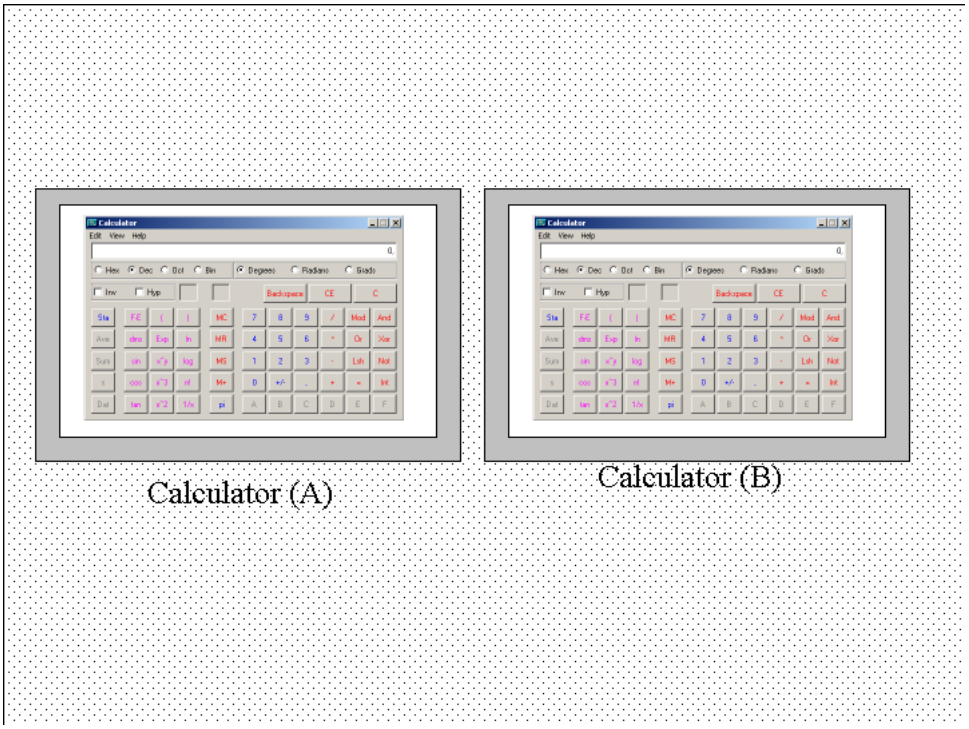


Figure 5.6: The calculators are no more open systems since there is not interaction defined among them .

that speaks both French and Romanian. So, the openness of a system is more related on the interaction capabilities of the elements composing it rather than its ability to dynamically modify its internal structure. Consequently, for MASs it is not the number of agents of the artificial society that determine the openness of the system but the interaction capabilities of the agents found in the society to influence the external world or to capture the influences from the external world. The next section shows how a 'bridge' mechanism is introduced in the artificial agent societies to define their openness.

Opening the Artificial Society

Section [5.3.1, page 151] has presented a proposition of an organizational model to specify software systems as artificial societies of agents. To make this model open, we have introduced the concept of *interface role*. The interface role is just like a classical role except the fact that it is reserved for the individuals that do not belong to the artificial society, namely the *extracives*. The extracives agents are individuals that may play interface roles within an artificial society and interact with the intracives agents of this society. Consequently, the interaction is defined between the inside and the outside of the artificial society.

The artificial society presented in Figure 5.7 follows an organizational structure composed of a single group, *'group 1'*, that is composed of two internal roles: *'role 1'* and *'role 2'*. At the concrete level two agents represent the intracives: *'agent 1'* and *'agent 2'*. These agents play respectively the *'role 1'* and *'role 2'*. Since these agents belong to the same group, they are able to interact. Obviously, this artificial society is a closed system since no interaction is possible between the intracives and the external world.

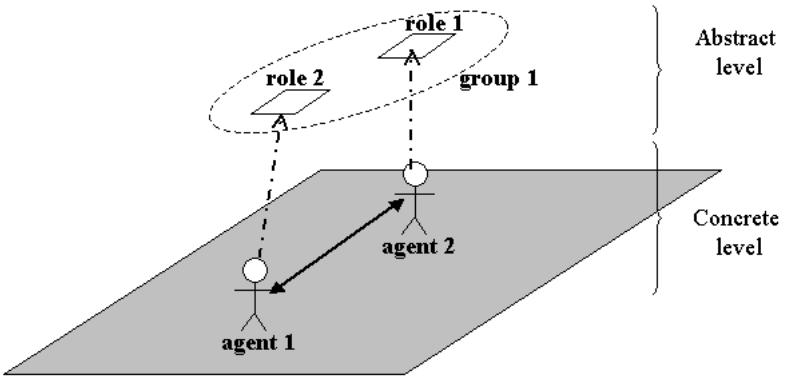


Figure 5.7: An example of a closed artificial society. The organizational structure is composed of a single group, *'group 1'*, and two internal roles: *'role 1'* and *'role 2'*. Following this organizational structure, *'agent 1'* and *'agent 2'* can interact internally within this system. However, no interaction is possible with the external world.

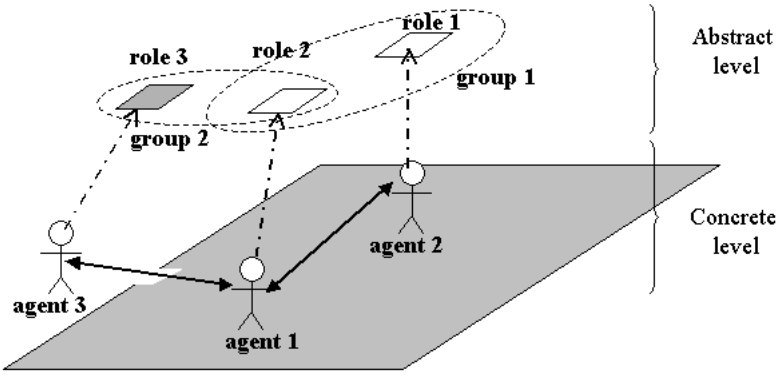


Figure 5.8: Example of an open artificial society: by adding an interface role *'role 3'* to the organizational structure defined in Figure 5.7, the intracives agents are able to interact with an extracives agent: *'agent 3'*. Consequently the artificial society is now open.

Now we add the interface role, *'role 3'*, and the group *'group 3'* in the organizational structure (cf. Figure 5.8). The interface role (represented by a grey rectangle) is defined for the extracives agents. For instance, the *'agent 3'* does not belong to the artificial society but this agent is able to play *'role 3'*. Consequently, this extracives agent is able to interact with *'agent 1'* and the artificial society is now considered as an open system.

5.3.3 Overall Picture of the Organizational Model

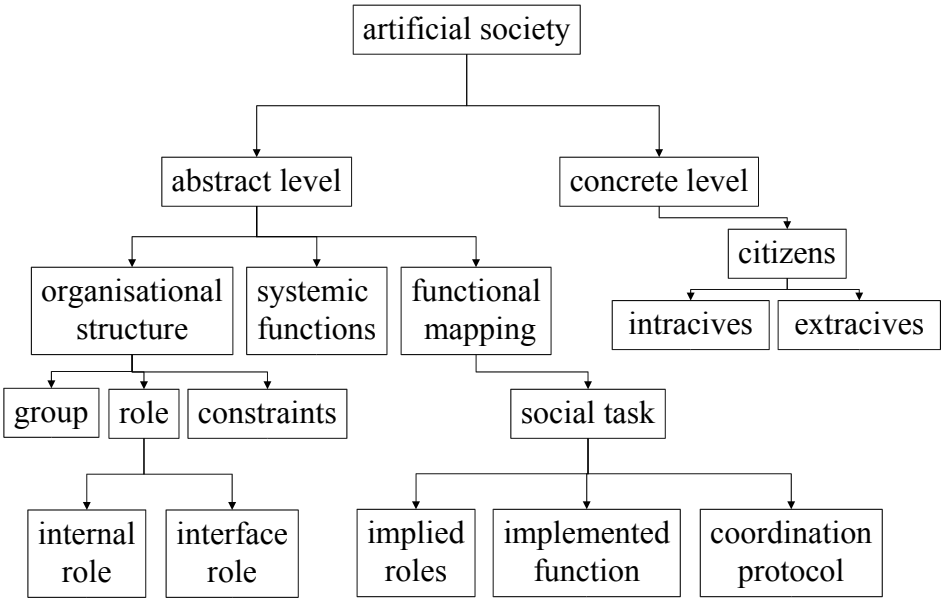


Figure 5.9: Concepts of the organizational model for the specification of open artificial societies for the EAC context.

The organizational model of open artificial societies is presented in Figure 5.9. The artificial society is divided in two parts: the concrete and abstract levels. The citizens define the concrete level. These citizens are placed in two exclusive categories: the intracives and extracives. The intracives agents are recognized as being part of the societal entity; while the extracives are the agents that may exist in the society but do not belong to the societal entity. The abstract level is the hidden part of the artificial society that regulates and organizes the citizens. The abstract level is composed of the organizational structure, the systemic functions and the mapping between the systemic functions and the organizational structure. The organizational structure is given by the set of roles, groups and structural constraints. The role defines the functions that are expected from an agent within its social environment. The social role may be either an internal or interface role. The internal roles can be played only by the intracives agents. The interface roles are played by the extracives agents. The groups create a context for the interaction between the agents playing roles. The groups that contain at least one interface role are considered as interface groups since they can contain extracives agents. The structural constraints express the cardinality of the roles and groups within the artificial society.

The systemic functions express what are the functions expected from the entire software system. These functions are then mapped to the organizational structure through the concept of social task. The proposed model is focused only on collective functions and the functions that do not imply the coordination of several individuals are not considered. The social tasks imply the participation of a set of roles. This set of roles has to be included within a group to allow the interaction of the agents. The coordination protocol describes how the participating agents coordinate their activities in order to achieve the social tasks.

Using MIC* as a DE and the coordination framework described in chapter 4, we can conclude that the abstract and concrete levels are consistent. In fact, the MIC* DE implements the organizational structure of the artificial society and the coordination framework checks the consistency of the conversations according to the established coordination protocols.

5.4 Translating the Organizational Model into MIC* Deployment Environment

The presented approach is similar to the SODA approach organisation:back:soda. The main idea is to build/generate a DE that implements the organizational structure. While SODA uses the tuple space coordination medium TuCSoN, we use the MIC* DE.

5.4.1 Translating the Organizational Structure

The organizational structure of the artificial society is defined by the set of groups and roles. The translation of these concepts to MIC* has already been presented in section [3.5, page 83]. The groups are considered as interaction spaces. The groups that contain at least one interface role are considered as public interaction spaces and can be shared between distributed MIC* DE when they are composed. The roles are considered as interaction objects. They represent a certification to play a certain role within the artificial society. The structural constraints are implemented by managing the number of certificates that are delivered to the agents.

5.4.2 Translating the Functional Mapping

The social framework presented in section [3.5, page 83] has not included the coordination of social tasks. Social tasks are assigned to the groups and carried out by the agents playing these roles. To check the consistency of the coordination protocols, we use the results presented in section [4.9.2, page 123]. Consequently, each interaction space is associated with a set of DBCMs that represent the coordination protocols of the social tasks assigned to the group. The MIC* DE controls in a decentralized manner the consistency of the conversations among the agents and the coordination protocol expressed for the social tasks.

5.4.3 Integrated Development Environment for the Specification of Open Artificial Societies

To help developers to design and develop applications for the EAC context an integrated development tool (IDE) has been developed[4]. This IDE is based on the organizational model

[4]The IDE is available for Windows platforms at: www.lirmm.fr/ gouaich/dev/ide/ide.zip

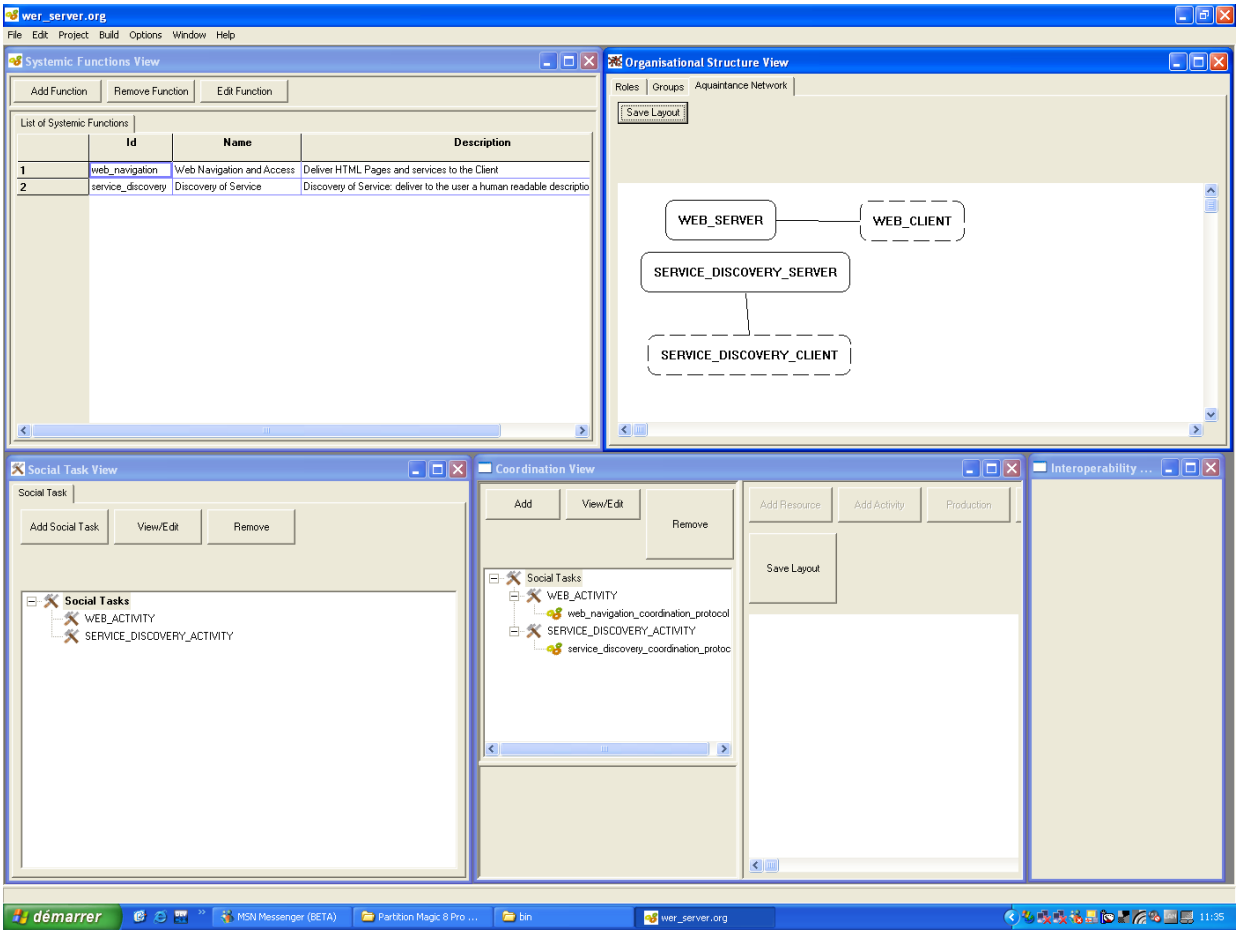


Figure 5.10: Screenshot of the IDE for the specifications of open artificial societies.

presented in section [5.3.3, page 157] and presents the development of an application for the EAC context as the development of an open and self-contained artificial society of agents. The designer is offered five design views which are (cf. Figure 5.10):

1. systemic functions: this view presents the main functions that are expected from the system. Generally, the development process starts by this view where a preliminary list of systemic functions is gathered.

2. organisational view: this view specifies the organisational structure of the artificial society in terms of roles, groups and structural constraints. The designer also indicates if the roles have to be considered as internal roles or interface roles. The groups contain a set of either internal or interface roles. The structural constraints are defined to indicate the cardinality of the roles and the groups.

3. mapping view: this view maps the systemic functions on the organisational structure by defining the social tasks. A social task is created to implement a certain systemic function and is associated to a subset of roles that are contained in a predefined group.

4. coordination view: this view specifies the DBCM coordination protocols that are associated with the social tasks. For each social task, the designer can graphically create a DBCM between the roles associated to the task. By using the result of chapter 4, we can conclude that the conversation protocol among the roles is completely defined.

5. interoperability view: this view manages the specification of the resources that were specified by the DBCM coordination protocols. In fact, for each DBCM a control function is associated to the resources which are defined within the roles-cut.

The abstract level models of the artificial society can be exported to an XML representation. A specific python script, `organisation2mic.py`, automatically generates the source code (C/C++) of the adequate MIC* DE. This code is ready for compilation and execution. The developer has only to develop the behavior of the intracives agents. The intracives agents are then deployed on the MIC* DE and the artificial society is ready to offer its functionalities and to interact with other artificial societies.

5.5 Application: Ubiquitous Web

The Ubiquitous Web application emulates the use of the web for the EAC context. The purpose of this section is to demonstrate how this application is developed using the proposed organizational model.

Two different types of sub-systems compose the ubiquitous web: the 'web server' and the 'web client'. This is similar to what can be observed in the Internet-based web. The 'web server' system is in charge of delivering HTML pages and replying to the client's requests. On the other hand, the 'web client' system is in charge of translating the users' action into commands which are comprehensible by the web server system and correctly displaying the HTML pages. The main difference between the Internet-web and the ubiquitous web is related to the communication medium between the web servers and clients. In fact, within the EAC context the web servers and web clients are no longer connected by a global communication medium but with a local communication medium.

The next section presents the organizational specifications of the web server. The organizational specifications are then used in order to specify the web client system. Obviously, the ubiquitous web application is defined only when both the web server and the web client are composed. The composition of these systems is steered by a simulation platform which is presented in section [5.6, page 170].

5.5.1 The 'Web Server' Specifications

Systemic functions

The developed software system is aimed to emulate the navigation and the access of services on the web for the EAC context. The systemic functions of the web server are as follows:

1. web navigation and the access to services: the 'web server' system delivers the HTML pages and forms.
2. discovery of services: the 'web server' system is also responsible of delivering a human readable description of the offered services and an access point where to find the actual service provider.

Organizational Structure

Roles

1. WebServerRole: this is an internal role that responds to the requests of the agents playing the role: WebClientRole. A WebClientRole requests either HTML pages or a service. When requesting a service, the WebClientRole agent delivers the parameters set by the user. The WebServerRole handles gracefully the following situations by sending error messages to the WebClientRole:
   - the requested html presentation does not exist.
   - the requested service does not exist or the parameters are incorrect or insufficient.
2. WebClientRole: this is an interface role that represents the intermediary function between the end user and the WebServerRole. The functions of this role are:
   (a) request a particular HTML presentation from the WebServerRole.
   (b) correctly layout the HTML presentation for the user.
   (c) request services from the WebServerRole by sending the parameters of the service as set by the end user.
3. ServiceDiscoveryServerRole: the function of this internal role is to deliver a human readable description of the offered service and to deliver an access point where to contact to the actual service provider.
4. ServiceDiscoveryClientRole: the main function of this interface role is to check the presence of the ServiceDiscoveryServerRole agents and to retrieve their description and access points.

Groups

1. WebGroup: this group holds agents that play the WebClientRole and WebServerRole roles.
2. ServiceDiscoveryGroup: this group holds agents that play ServiceDiscoveryServerRole and ServiceDiscoveryClientRole roles.

Structural Constraints The artificial society of the web server defines the roles and groups with the following cardinalities:

- exactly one WebGroup is defined within the artificial society.
- exactly one ServiceDiscoveryGroup is defined within the artificial society.
- exactly one intracives agent is able to play WebServerRole within the WebGroup;
- exactly one intracives agent is able to play the ServiceDiscoveryServerRole within the ServiceDiscoveryGroup group.

Mapping the Systemic Functions on the Organizational Structure

WebSocialActivity **:**

- The implemented function: the 'web navigation and service access' function is implemented by this social task.
- The roles and groups: this social task implies the participation of the WebClientRole and WebServerRole roles and is associated to the WebGroup.
- Description: the WebSocialActivity represents the web-like navigation in the EAC context. During this activity the WebServerRole delivers the *rooms* or error messages in response to the WebClientRole requests. The WebClientRole can either request a room by its identifier; or request for an action given the parameters. Usually, these parameters are the inputs of an HTML form set by the end user. A room is basically a complete directory containing HTML pages and images that is packed and compressed. The room mechanism is used to minimize the communications between the WebClientRole and WebServerRole. Thus, contrary to conventional Internet-web where each HTML page and each image is separately requested, within the Ubiquitous Web the WebServerRole sends a complete presentation. The HTML pages of a room may contain the following items:

  1. conventional HTML elements such as tables, images and so on.
  2. hyperlinks to pages found in the same room.
  3. hyperlinks to other rooms.
  4. forms to request services from the WebServerRole.

  The error messages are sent by the WebServerRole when the requests cannot be performed. By convention, the WebClientRole starts the activity by requesting the initial room identified by '0'. This is like the conventional 'index.html' in Internet based web.

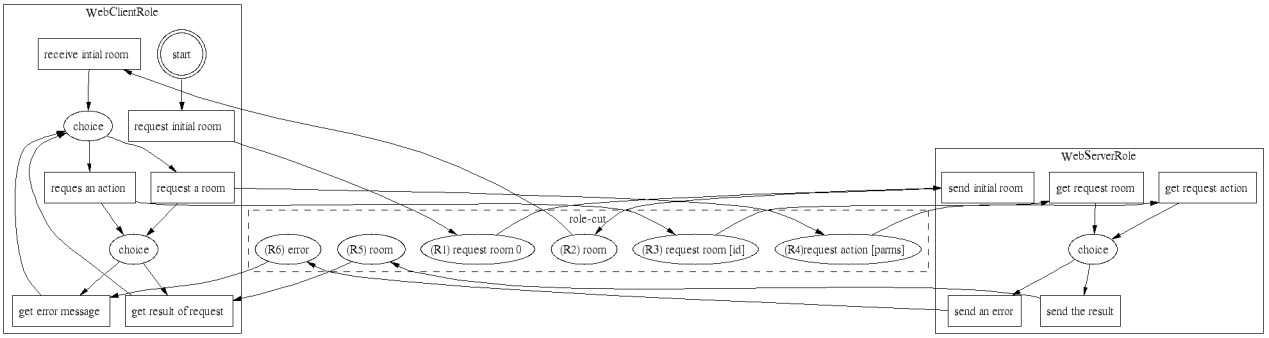


Figure 5.11: The coordination protocol of the WEBACTIVITY social task.

- Coordination Protocol:

  The WEBSOCIALACTIVITY uses the coordination protocol represented graphically in Figure 5.11. The control functions of the resource contained in the roles-cut are described as follows:

  R1: this resource node represents the request for the initial room. By convention, the initial room is identified by `'0'`. The actual syntax of this message is: `(request room 0)`.

  R2: this resource node represents the response to the initial room request. The actual resource of this node is represented by a string that follows this syntax: `(room` *identifier Base64encoding*`)`. The `room` is the keyword specifying the message type; *identifier* is a token representing the room identifier; *encoding* represents the base-64 encoding of the room. In fact, the room directory is encapsulated in a single file (using for instance the Unix-like `tar` command) and this file is then compressed following the *Gzip* compression algorithm described by RFC 1950 [Deutsch & Gailly, 1996] and RFC 1951 [Deutsch, 1996]. Finally, the compressed binary file is encoded using the Base64 encoding (RFC 2065 [Eastlake & Kaufman, 1997]) to transform the binary data to ASCII characters.

  R3: this is a request emitted by the client to get a specific room using its identifier. The actual resource associated to this node is represented by a string that follows this syntax: `(request room` *identifier*`)`. The `request` and `room` are keywords and *identifier* is a token representing the identifier of the requested room.

  R4: this resource node represents a request to execute a service. The actual resource that is associated to this resource node is represented by a string that follows this syntax: `(request action` *identifier parameter* `)`. The `request` and `action` are keywords; *identifier* is the name of the requested service; *parameters* is the encoding of the parameters. The parameters are encoded following the specifications of HTTP/1.1. Basically, this encoding is a sequence of pairs *name*=*value*. Some special characters are reserved and are replaced by an encoding described by the URL specification in RFC 1738 [Berners-Lee *et al.* , 1994].

  R5: The specification of the (R5) resource is similar to the specification of (R2).

  R6: this resource node corresponds to an error message sent by the server to the client. The actual resource associated to this resource node is a string message that follows

this syntax: `(error` *description* `)`. `error` is a keyword used to specify the type of the message and *description* is a string that describes the reasons of the failure.

Some examples of conversations generated by this coordination protocol are presented in Annex E, page ubiweb:conversation.

ServiceDiscoverySocialActivity**:**

- The implemented function: this social task implements the 'discovery of services' systemic function.
- Implied roles and social group: this activity implies the participation of the ServiceDiscoveryClientRole and ServiceDiscoveryServerRole roles. It is associated with the ServiceDiscoveryGroup.
- Description:

  Thanks to this activity the clients retrieve a description of the offered services. For each web service, a special agent playing the ServiceDiscoveryServerRole role is in charge of advertising and informing the clients about the offered services.

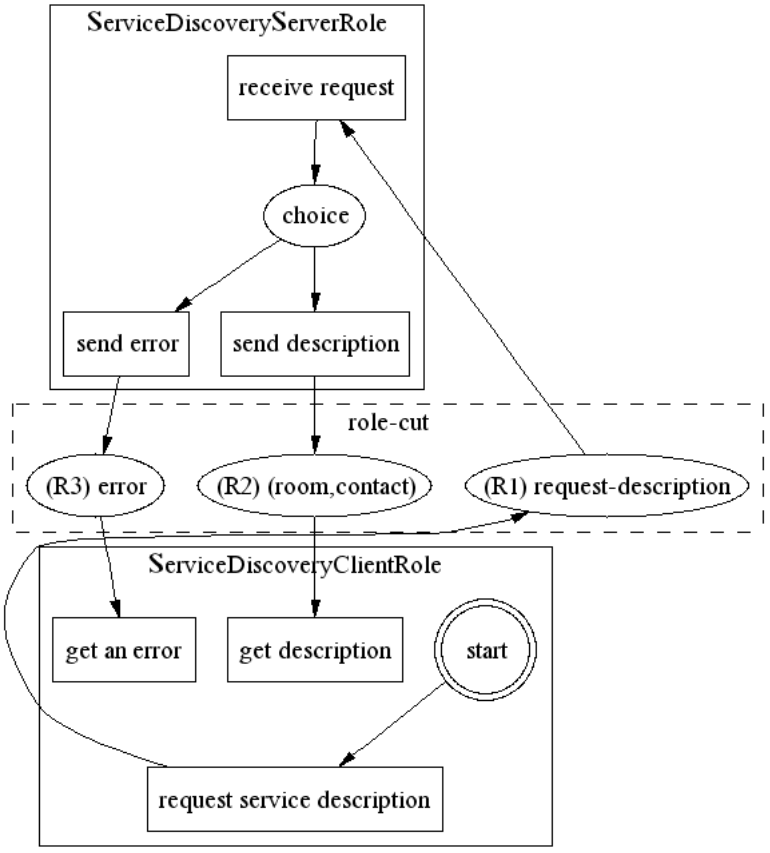


Figure 5.12: The coordination protocol of the ServiceDiscoverySocialActivity social task.

- Coordination Protocol: The coordination protocol of the ServiceDiscoverySocialActivity is presented in Figure 5.12. The ServiceDiscoveryClientRole sends

a request to SERVICEDISCOVERYSERVERROLE and waits a reply. The reply is either the description of the service or an error message. The resources that are exchanged within the roles-cut are described as follows:

R1: This resource represents the initial message sent by the SERVICEDISCOVERYCLIENTROLE to retrieve the description of the service. The syntax of the message is: `(request-description)`.

R2: This resource contains the description of the service and its access point. The syntax of the actual resource is: `(service-description` *room agent-id*`)`. The 'service-description' is the keyword to specify the type of message. The *room* is a human readable description represented as a set of HTML pages. The room contains some HTML pages and images presenting briefly to the end user the offered services. The room is encapsulated, compressed and encoded as Base64 string. The last part of the actual resource, *agent-id*, is the identity of the agent playing the WEBSERVERROLE. Consequently, if the end user is interested by the offered service, she/he can contact directly the web server using the coordination protocol presented in the WEBSOCIALACTIVITY (cf. Figure 5.11).

R3: This resource represents the error cases. If the SERVICEDISCOVERYCLIENTROLE is unable to give the description of the service, he/she sends a message of this syntax `(error` *description*`)`. The *description* is a string that explains the reason of the failure.

Web Server Architecture

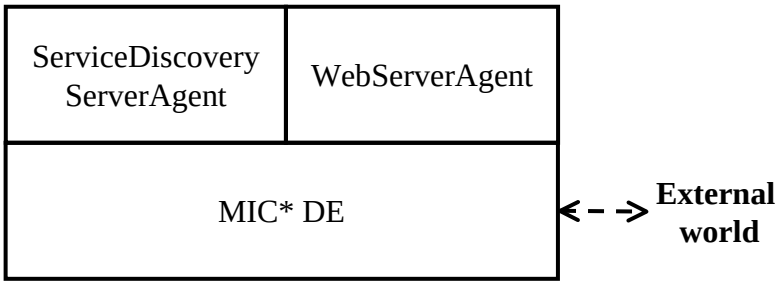


Figure 5.13: The architecture and main components of the 'web server'.

Figure 5.13 presents the components of the 'web server':

- ServiceDiscoveryServerAgent: this agent is deployed on the MIC* DE and plays the ServiceDiscoveryServerRole role within the ServiceDiscoveryGroup group.
- WebServerAgent: this agent is deployed on MIC* and plays the role of WebServerRole within the WebGroup social group.
- MIC* DE: this represents the DE that has been generated automatically from the organisational specifications of the 'web server'. This DE holds the autonomous agents and is on-the-fly composed with other MIC* DE.

5.5.2 The 'Web Client' Specifications

The 'web client' system imports the organisational interface defined by the 'web server' artificial society. Since the architecture of the 'web client' is quite simple the organanisational structre that is imported from the 'web server' is not extended with new roles and groups. Notice that in the imported organisational structure the interface roles become internal roles and simarly the internal roles become interface roles within the 'web client' system.

Organizational Structure

Roles

1. WebServerRole: this is an interface role that responds to the requests of the agents playing the role: WebClientRole.
2. WebClientRole: this is an internal role that represents the intermediary function between the end user and the WebServerRole.
3. ServiceDiscoveryServerRole: the function of this interface role is to deliver a human readable description of the offered service and to deliver an access point where to contact the actual service provider.
4. ServiceDiscoveryClientRole: the main function of this internal role is to check the presence of the ServiceDiscoveryServerRole agents and to retrieve their description and access points.

Groups

1. WebGroup: this group holds agents that play the WebClientRole and WebServerRole roles.
2. ServiceDiscoveryGroup: this group holds agents that play ServiceDiscoveryServerRole and ServiceDiscoveryClientRole roles.

Structural Constraints The artificial society of the web client defines the roles and groups with the following cardinalities:

- exactly one WebGroup is defined within the artificial society.

- exactly one ServiceDiscoveryGroup is defined within the artificial society.
- several intracives agents are able to play WebClientRole within the WebGroup;
- exactly one intracives agent is able to play the ServiceDiscoveryClientRole within the ServiceDiscoveryGroup group.

Mapping the Systemic Functions on the Organizational Structure

The mapping of the systemic functions to the organisational structure is directly imported from the web server specifications as described in section [5.5.1, page 162].

Web Client Architecture

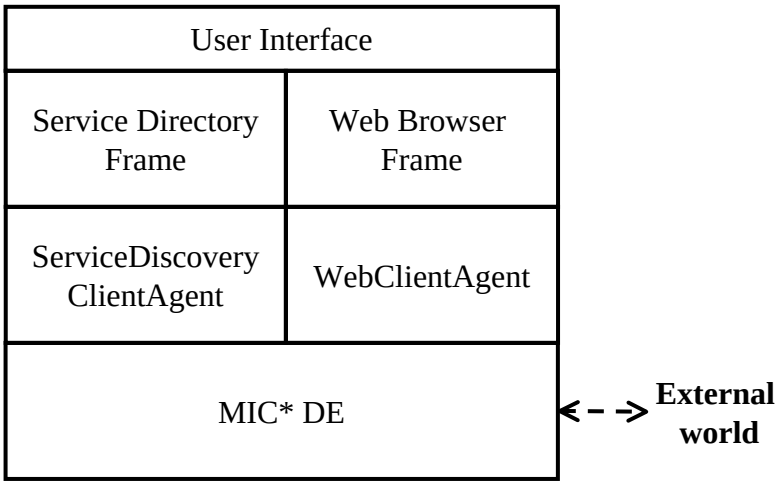


Figure 5.14: The architecture and main components of the 'web client'.

Figure 5.14 presents the components of the 'web client':

- UserInterface: this component represents a window-based user interface that contains the service discovery frame and the web browser frames.
- ServiceDirectoryFrame: this frame presents a logical view of the discovered services as a directory. When the user wants to interact with a particular service a WebClientAgent is launched along with its associated WebBrowserFrame.
- ServiceDiscoveryAgent: this agent is deployed on the MIC* DE and plays the ServiceDiscoveryClientRole role within the ServiceDiscoveryGroup group. When

the user requests a service lookup, this agent uses the service discovery coordination protocol in order to get information about all the accessible services. The list of services is then returned to the ServiceDirectoryFrame to be displayed logically as a directory.

- WebBrowserFrame: this class presents the HTML pages collected by the WebClientAgent. The user requests are forwarded to the WebClientAgent.
- WebClientAgent: this agent is deployed on MIC* and plays the role of WebClientRole within the WebGroup social group. This agent gets events, such as the submission of a form or a request for a specific room, from the WebBrowserFrame and starts the corresponding coordination process with the peer agent that plays the WebServerRole. The obtained results are returned to the WebBrowserFrame that presents them to the user.
- MIC* DE: this represents the DE that has been generated automatically from the organisational specifications of the web client application. This DE holds the autonomous agents and is on-the-fly composed with other MIC* DE.

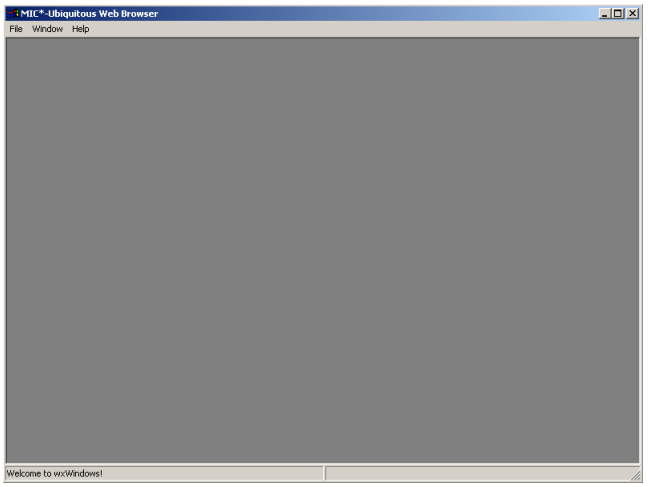


Figure 5.15: The main frame of the web client user interface.

Figure 5.15 presents the user interface that appears when the user launches the 'web client' application. The service directory frame is launched by clicking the appropriate menu.

When the service directory frame is launched (cf. Figure 5.16), the user may lookup for the services that are in the vicinity. The number of services depends on the DEs that are composed with the 'web client' DE.

For instance, Figure 5.17 presents a situation where the service discovery agent has successfully discovered a service. The left-side of the frame presents all the discovered services as a directory. By selecting a particular service, the right-side of the frame displays its presentation to the user. By double-clicking the user is connected to the service through the WebClientAgent. For instance, Figure 5.18 presents an example where the WebClientAgent has retrieved the initial room that is displayed by the WebBrowserFrame.

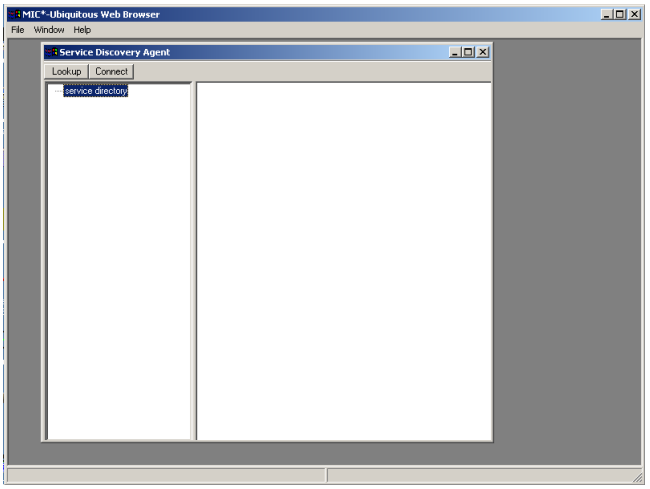


Figure 5.16: The service directory frame is used by the user to search for the available services .

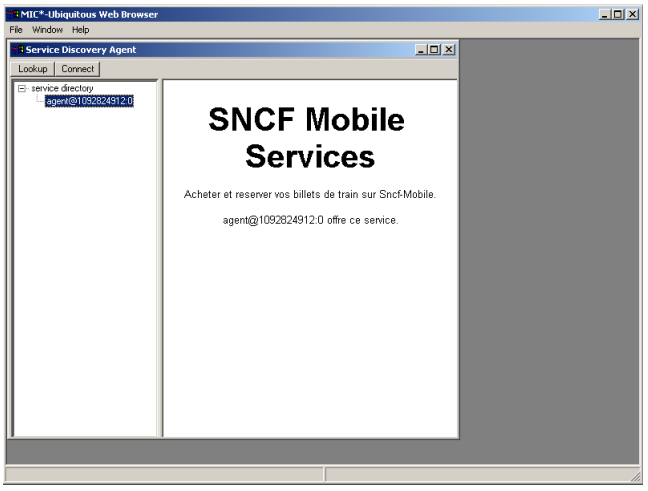


Figure 5.17: Description of a discovered service.

Figure 5.18: The initial room of the service is retrieved and displayed to the user. Now the user is able to interact directly with the service.

5.6 The EAC Simulation Platform

5.6.1 Presentation of the Simulation Platform

In order to experiment with the developed applications, a simulation platform has been developed using computer games technologies. The goal of this simulation platform is to emulate a physical world where the user, represented by its avatar, can move and interact with the deployed services, represented also by their avatars. Within the virtual world each avatar has a certain range where it can establish a local point-to-point communication link. So, the simulation layer is in charge of firing two main events: connection event, when the avatars are able to communicate; and disconnection event, when the avatars cannot communicate. No further information is given to the upper layers about the location of the avatars. This makes the simulation platform very realistic and its constraints similar to the EAC constraints.

5.6.2 Link between the Simulation Platform and the MIC* Layer

The upper layer that runs over the simulation layer is composed by the MIC* DEs. MIC* DEs are represented in the simulated world by avatars: when the avatars' communication areas overlap, the corresponding MIC* DEs are composed; when the avatars communication areas do not intersect, their corresponding MIC* DEs are disconnected (cf. Figure 5.19).

5.6.3 User Views in the Simulation Platform

The user has a 'First Person Shooter' (FPS) perspective and can move within the virtual world. Figure 5.20 shows an example of the FPS perspective. The user perceives the virtual 3D world as the real world and when moving she/he can interact with the different services.

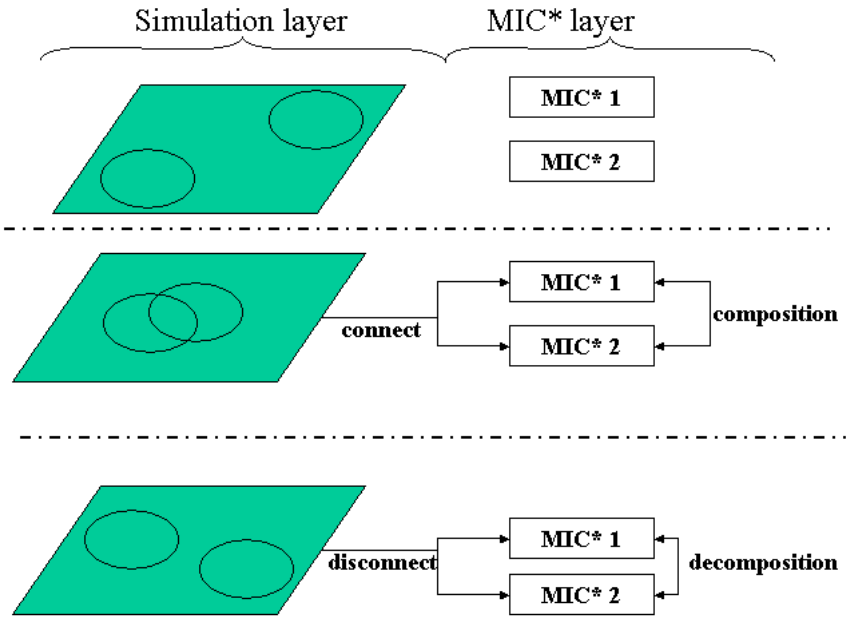


Figure 5.19: Link between the EAC simulation platform and the MIC* DEs. When the avatars communication areas intersect in the virtual world, their corresponding MIC* environments are composed. On the other hand, when the communication areas do not intersect, the corresponding MIC* environments are decomposed. The composition and decomposition events are commands defined by MIC* and sent through TPC/IP sockets.

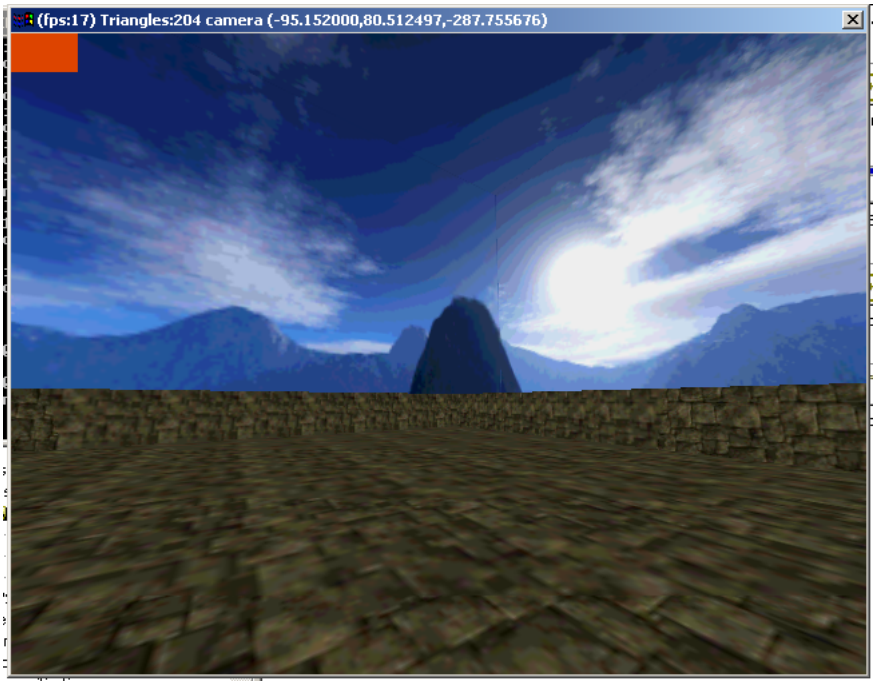


Figure 5.20: The First-Person-Shooter (FPS) perspective of the user in the simulation platform .

The rectangle located in the top left corner of the interface defines the state of communication link: when no service is in the vicinity of the user, the rectangle is filled with a red colour; when some services are located in the vicinity of the user, but situated too far to establish a communication link, this rectangle is filled with a yellow colour; finally, when the distance between the user the service is sufficient to establish a communication link, this rectangle is drawn with a green colour. Remember that the establishment of a direct point-to-point communication link is the minimal requirement needed in order to compose the MIC* DEs. When this communication link is broken the MIC* DEs are immediately decomposed.

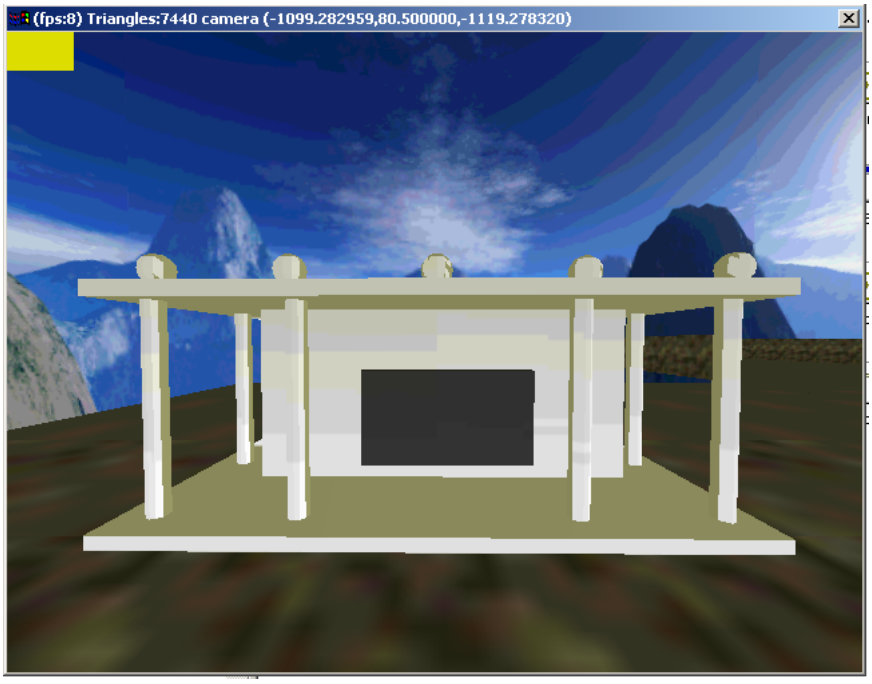


Figure 5.21: The software service is represented within the virtual world as a building that owns a communication area.

The services' avatars are represented in the virtual world as buildings. Hence, as presented in Figure 5.21, each software service is associated to a building and owns a communication area where it can establish local communication links with the users' avatars.

The description of the virtual world is given in a special description file. Figure 5.22 gives an example of such a file. Each service is described by: its name; the host and port where its corresponding deployment environment is listening to commands; 3D coordinates of the service avatar in the virtual world. The simulator takes this file as an input and builds the corresponding virtual world. The user is then able to move in this world and interact with the services.

For example, Figure 5.23 presents a situation where the user enters the service building. Since, a communication link can be established, the MIC* DEs of the user and the service are composed. Consequently, the software agents located in both DEs can interact. When the user leaves the building of the service, the MIC* DEs are decomposed immediately. Consequently, the software agents located in these DEs cannot interact. Thanks to the on-the-fly composition of MIC* DEs and to the generative communication paradigm, now we can show experimentally that the EAC constraints on the communication media do not drastically disturb the software systems.

```
# world.txt simple example of wrold 3D description
# virtual services are specified by a line:
# <name> <host> <port> X Y Z
# name: is a token that should not contain blank characters
# host: is either an ip address or the host name where the
#      service mic* deployment is running
# port: is the listening port of the MIC* deployment environment
# Y X Z: coordinate in the virtual 3D world.
#
# Comment lines are started by the '#' character

#<name>          <host>         <port>  X     Y   Z
helloWorld       127.0.0.1      2001    1100  0  -4000
shop01           127.0.0.1      2002    1300  0  -4000
shop02           127.0.0.1      2003   -1100  0  -4000
tourism-info01   127.0.0.1      1235    1100  0  -900
train-station01  localhost      1236   -1100  0  -900
#end of the simple world
```

Figure 5.22: Description file of the virtual world. The simulator takes this file as an input and positions correctly the services avatars using information on localisation. The simulator also uses information about the host and port in order to send the 'connect' or 'disconnect' commands to the service's deployment environment

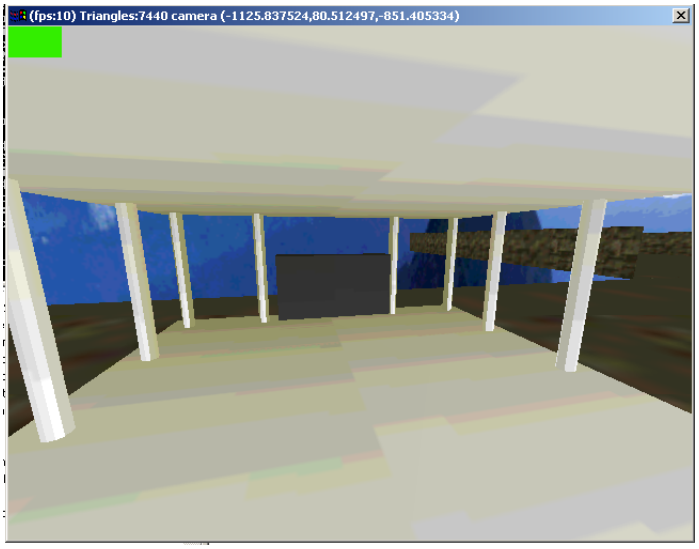


Figure 5.23: The user is in an interaction with the service.

5.7 The Virtual City Project

The goal of this experimental project is to experiment with EAC applications. The project aims to build a virtual city where several services are deployed. Several different developers develop the services using the engineering framework presented in this chapter. Each service has also to request a virtual place within the virtual city. Currently, the services that have been developed within the virtual city are the following:

- Train stations: this service represents a train station where the users can have information about the available trains and can buy an electronic train ticket.
- Tourism information points: the virtual city contains a set of services that provide description of the interesting sites. Hence, when a user is visiting the city, she/he may get information about the visited site. Even if the site is located outside any coverage of a global wireless network, the users are able to access this service only by using local communication technologies.
- News: these services are located near bus and metro stations to allow users to read electronic newspapers.
- Restaurants: this service has been provided to allow users to read the menu and to order meals directly. The users have only to be placed in the restaurant, and by using this application she/he can read the menu of the restaurant, order and pay the desired meal directly.
- Shops and small businesses: a set of shops were developed in the virtual city. When the user moves in the virtual city, she/he is able to see the catalogue of each shop. If the user is interested in a particular item, she/he may buy it directly from the shop. This is a very interesting example of the power of local communication technologies. In fact, since the cost of a local communication facility is very insignificant. The EAC services are more retained for big businesses and companies. Every small business and shop may take part in the system without a heavy cost.
- Virtual university campus: each department of the virtual university campus is represented by a service that informs the students about the offered courses, administrative information and the planning of the courses. Hence, by using their laptops the students have access quickly to information they are seeking.

The virtual city is an open project and developers are free to imagine new set of services to deploy. The developers are given a development kit with some examples to ease the development of new services.

5.8 Conclusion

In this chapter we have presented an engineering framework based on the social metaphor where the software systems are viewed as open and self-contained artificial societies that interact with other societies through an organizational interface.

The organizational structure of the artificial society is given by the set of groups and roles and the systemic functions are implemented by the social tasks. The coordination of each

social task is specified using the DBCM that were presented in chapter 4. We have introduced new organizational concepts such as the interface/internal roles and the intracives/extracives agents in order to explicitly express the openness of the artificial society within the EAC context.

The presented engineering framework offers the opportunity to non-expert persons to benefit easily from the results presented in the previous chapters in order to build application for the EAC context: while the engineering framework presents a social view of the software systems, the link between this view and the MIC* model and the coordination framework is still maintained. In fact, the social models of the software system are translated automatically into specific MIC* DEs that implement the organizational structure and guarantee the consistency of the conversations with regards to the coordination protocols.

The UbiquitousWeb is a realistic application that has been entirely developed using this engineering framework. The simulation platform has offered the opportunity to experiment with this application in a virtual world that offers the same characteristics as the physical world were the EAC application would be deployed.

To extend this experiment, we have presented the virtual city project in order to experiment how different engineers can offer open and interoperable services. Obviously, the next step would be to deploy the developed services in the physical world using local communication networking facilities.

From a practical point of view, we have experimented that the development time of EAC applications has been significantly reduced. In fact, the engineering framework that we have proposed is model driven and generates most of the source code of the application. Thus, both of the development time and bugs are reduced.

Chapter 6

Conclusion

6.1 Synthesis

In this thesis we have presented an approach for the design and implementation of open software systems within the EAC context. The EAC context exhibits properties and features that challenge our assumptions and way to think software systems. Consequently, these assumptions have to be revised in order to take into account the features and constraints of the EAC context.

In chapter 2 we have selected three major criteria of the EAC that are relevant for the engineering of open software systems. These criteria are as follows:

1. **Constraints of the communication medium**: the communication media used within the EAC are different from the one used in classical distributed software systems such as local area networks, wide area networks or centralized wireless networks. In fact, while the classical communication media are centralized and quite reliable, the communication media within the EAC are local and appear spontaneously among the systems as soon as some physical or logical conditions hold. When these conditions are unsatisfied the communication media do no longer accomplish their functions. Consequently, the communications are supposed to be intermittent. This property is no longer a *problem* but an hypothesis of work.

2. **Autonomy of the software entities and systems**: within the EAC, the software systems are considered as autonomous since we do not have enough knowledge and control means to change directly their behaviors or to guess what are their actual goals. Despite this fact, autonomous software systems should be able to interact and collaborate with other systems without challenging their consistency.

3. **Openness of the software systems**: the EAC context can offer a new set of applications and innovative services only when considering the interaction capabilities of the software systems. The openness of the software systems also implies that the systems collaborate in a coherent manner. So, this introduces the problem of interoperability. We have suggested addressing the interoperability of software systems by using coordination means.

The literature of software engineering already presents interesting approaches to deal with the constraints and hypothesises of the EAC. For instance, the MAS and BB approaches have been considered as relevant starting point for our work.

MASs are relevant for the EAC since they take care about the autonomy of software entities at the conceptual level and offer some means for the interoperability of autonomous systems. Still, the MAS approach suffers from the gap that exists among the conceptual considerations and the implementation. For instance, while the autonomy is well studied and documented at the conceptual level, few works provide objective criteria to use in order to implement autonomy. We have suggested to consider the internal integrity as an objective criterion to guarantee the autonomy of software entities. We have also mentioned the communication paradigm to illustrate the gap between the conceptual considerations and the implementation of the MASs. In fact, at the conceptual level the MAS paradigm states that the software entities are communicating entities without providing any information on the paradigm to use for the communication. At the implementation level most of current approaches use the asynchronous communication paradigm without explicating that this is just a special case of the communication among software entities and that other communication paradigms can also be used. We have considered that the asynchronous communication paradigm is not suitable to handle the constraints of the communication media within the EAC context and we have preferred to use the generative communication paradigm.

The BB approach and the TS offer some good properties for the EAC context. In fact, these approaches *de facto* uses a generative communication as a paradigm of communication between software entities. The generative communication makes the communication process persistent and thus independent from the disturbances of the communication media. The advantage of the BB approach is not only limited to the generative communication. In fact, as illustrated by the works developed around TuCSon TS and SODA, the interoperability of software systems is also addressed from a coordination point of view and 'artifacts' of coordination are defined within the tuple centres to offer the software entities coordination means. Still, we have mentioned that the TS-based approaches suffer from the lack of structure of the TS. This lack of structure causes both conceptual and practical problems. At the conceptual level, the lack of structure poses the problem of the adequacy between the autonomy of actions of the entities and the TS. In fact, since the TS is a flat set that is shared similarly among all the software entities, the shaded actions problem described in section [2.4.4, page 37] appears. Practically, the lack of structure of the TS raises also some problems in order to specify and to implement the on-the-fly composition of the software systems.

To solve the problems revealed by the state of the art analysis, we have presented the MIC* model in chapter 3. MIC* is an algebraic model of an abstract infrastructure where autonomous and interacting software entities are deployed. MIC* implements the internal integrity criterion since the software agents do not have a direct access to the software structure of other agents. In fact, the interactions are mediated by the MIC* DE.

By contrast to the TS, MIC* is structured horizontally by the agents and vertically by the interaction spaces. Consequently, each agent can access only to a specific region of the DE: its outboxes, where both the interaction means and influences are located, and its inboxes where the perceptions are located. The means of interaction, influences and perceptions are unified and reified as interaction objects.

Thanks to the algebraic modeling, the on-the-fly composition of software systems is formally specified by the composition of the DEs. In other words, we are now able to get formally the result of a composition of several MIC* DEs. At the implementation level the composition

of MIC* deployment environments is also implemented and the composed MIC* environments offer a virtual deployment environment that is consistent with the formal specifications.

The MIC* model has answered two main questions that are related to the constraints of the communication medium by offering a generative communication and the autonomy of software entities by implementing the internal integrity. The point that has not been addressed by the MIC* model concerns the coordination between the autonomous agents.

The coordination theory of Malone and Crowston [Malone & Crowston, 1994] has constituted the basis of the coordination framework presented in chapter 4. In this chapter, we have presented the dependency-based coordination model (DBCM) to specify the coordination protocols among a set of roles. The DBCM represents the set of activities, resources and consumption/production relationships. This formalism is suitable to capture the parallel aspects of a coordination process. Furthermore, we have formally provided the link between a DBCM coordination protocol and its set of conversations. This contrast with current approach found within the literature of MASs that specifies explicitly the conversations among the agents to perform joint activities. In fact, only the coordination protocols expressed as DBCMs are sufficient to specify the structure of the conversations that hold among the entities. Besides we have also provided practical means in order to check the consistency of the conversation according to the definition of a coordination protocol. For this reason we have extended the definition of the Petri Nets to define the Queue Petri Nets (QPNs). Thanks to the QPNs the consistency of the conversation and the coordination protocols are checked in incremental manner. We have also presented some algorithms that show how the structure of the QPN is generated from the DBCM coordination protocol.

Moreover, the integration of the coordination framework within MIC* has been performed without contradicting the hypotheses of the EAC context. In fact, we have seen that the centralized approach to check the consistency of the conversations was inconsistent with the hypothesis of the EAC. For this reason, we have presented a decentralized approach that checks only that each role that participates to the coordination process does not challenge locally the order constraints imposed by the DBCM coordination protocol.

In chapter 5 we have presented a practical engineering framework for the design and the implementation of software systems as open artificial societies. The open artificial society is specified by an organizational model in which we have integrated the concepts of internal/external roles and intracives/extracives in order to reify explicitly the openness of software systems. The coordination of the joint activities among the roles is specified using DBCMs. The organizational specifications of the artificial society are then automatically translated to a MIC* deployment environment that represents in this case the social deployment environment. The interoperability of different artificial societies is addressed by the coordination of the joint activities that hold among these artificial societies.

To experiment with the developed applications and to validate experimentally the soundness of the presented approaches, we have developed a simulation platform to represent the EAC context. For instance, this simulation platform has offered the opportunity to experiment with the Ubiquitous Web application that has been presented as a case study. The Virtual City project is also experimented using this simulation platform.

6.2 Summary of the Main Contributions

6.2.1 MAS

1. **Considering the infrastructure of MASs as a first-class entity**: the infrastructure that surrounds the agents was not explicitly represented as a first-class entity at the design and implementation levels within MASs. We have referred to this infrastructure as the deployment environment. We have highlighted the important role played by the deployment environment since it affects the dynamics of the whole MAS. In fact, as the deployment environment implements the interaction among the agents, one has to know explicitly the properties and constraints that the deployment environment imposes on the interactions of these agents in order to capture the whole dynamics of the MAS. The deployment environment also guarantees the internal integrity of the agents and consequently their autonomy.

2. **Filling the gap between conceptual considerations and the implementation of the autonomy**: the autonomy of agents is well studied and documented at the conceptual level within MASs. Still, at the implementation levels few works have presented objective criteria in order to either implement or guarantee the autonomy of software entities. We have related the autonomy of software entities to lack of knowledge and means to control their behaviors or predict directly their intentions. As consequence to this interpretation of the autonomy, the internal integrity becomes a *sine qua none* condition to guarantee the autonomy of software entities. Thanks to the identification of the deployment environment as a first-class entity, the internal integrity of software entities is guaranteed at the implementation level.

3. **Identification of the responsibility of the autonomous agents**: the identification of the deployment environment within MASs has also allowed the identification of the responsibilities of the autonomous agents. In fact, we cannot provide the agents the autonomy of actions without establishing a referential to which we evaluate the consistency of their actions. In other words, the more we consider the agents as autonomous entities, the more we need to pay attention to their actions. We have suggested encapsulating the rules and norms that specify the consistency of the MAS within the deployment environment. The other consequence of the autonomy is always to consider the actions of agents as attempt of actions or influences [Ferber & Muller, 1996]. The deployment environment defines its owns reaction rules to the influences of the agents. For instance, within the MIC* model the influences are reified as interaction objects and the deployment environment completely controls the agents' influences before committing them.

4. **Making the interaction process independent from the agents**: the MIC* model reifies the information carriers, influences and interaction means in a unified structure that is the interaction objects structure. The shift that is made with this approach is to consider the interaction not between the agents but between the interaction objects. So, the interaction process is defined within the deployment environment independently from the agents. This helps in the separation of concerns for the engineering of MASs. In fact, deployment environments are engineered independently from the models and structures of the agents. Furthermore, heterogeneous agents may be deployed on the same deployment environment at least if they share the same ontology of interaction objects.

5. **Separating the interaction time from the computation time**: the MIC* model clearly distinguishes between the computation time and the interaction time. During the computation time, the agents emit influences, while during the interaction time the deployment environment calculates the result of the interactions between the interaction objects. This result has been particularly interesting for MASs with synchronous agents as it is the case for agent-based simulations [Michel, 2004]. In fact, this property guarantees that the actions of the agents and perceptions are coherent during the simulation process.

6. **Structuring the interaction means of MAS** : some MASs have taken the asynchronous message passing paradigm as legacy from concurrent objects technologies without questioning this hypothesis. We have abstracted the interactions of the MAS and defined an abstract structure of interaction object. The structure of the interaction objects as a commutative group closed by a composition law has helped for the formal definition of the composition of several deployment environments.

6.2.2 BlackBoards & Tuple Spaces

1. **Structuring the interaction medium and enabling their composition**: the MIC* model structures the interaction medium contrary to classical TS approaches. The MIC* model is structured as inbox and outbox matrices that are divided horizontally by the agents and vertically by the interaction spaces. This structure makes the interaction medium consistent with the hypotheses that were made on the software entities. In fact, each software entity has only access to a reserved region of the deployment environment. Furthermore, the structure of the MIC* deployment environment has helped in implementing the on-the-fly composition of software systems. In fact, when the MIC* deployment environments are composed the synchronization occurs on shared interaction spaces only for the calculation of the interactions of the software agents. Thanks to the modeling of the interaction means as interaction objects, the perceptions of the agents are calculated remotely by moving the interaction means. The decomposition of the deployment environments is flexibly handled since it does not require any special management.

2. **Unification between the interaction means and information carriers**: the TS uses the tuples as information carriers and anti-tuples as interaction means to retrieve the tuples from the interaction medium. We have unified the information carriers and interaction means as interaction objects. Consequently, even when a software entity is not explicitly requesting to extract an information from the interaction medium, it is still interacting within the system. This property cannot be defined within TS architectures. Besides, the unification of the interaction means and information carriers also offers a high level of introspection and reflection. In fact, depending on the interaction rule that is used, the same interaction object can be considered either as an interaction mean or as an information carrier.

6.2.3 Coordination

1. **Unified view of the coordination and conversation protocols**: we have presented an approach that unifies the coordination and conversation protocols. The coordination

protocols are expressed as DBCM digraphs that represent the relationships among different roles in terms of activities that produce and consume resources. DBCMs are particularly interesting to express the parallelism of activities during a coordination process. Furthermore, the conversations that are generated by the DBCM coordination protocol are defined formally as a rewriting system and recognized in practice by using the Queue Petri Nets.

2. **Checking the consistency between the conversation and coordination protocols**: we have presented both a formal link between the coordination and conversation protocols and means to control the consistency of a conversation protocol with regards to the DBCM. Besides, the monitoring of conversations is performed in a decentralized manner that is consistent with the hypotheses stated on the EAC context.

6.2.4 Social Metaphor for the Engineering of EAC Applications

1. **Using the social metaphor for the engineering of EAC applications**: we have used the social metaphor for building software systems as open artificial societies. The organization of the open artificial societies is specified using classical organizational concepts found in the literature of MASs. We have added new ideas such as the distinction between the internal and interface roles and intracives and extracives agents in order to model the artificial society as a self-contained system and to capture explicitly the openness at the organizational level. We have presented a model driven approach. So, the designer specifies the properties of the artificial society using an IDE and the specifications are then automatically translated into a MIC* deployment environment that implements the organizational structure of the MAS and checks the consistency of the conversations with regards to the established coordination protocols.

6.3 Perspectives

6.3.1 Grid Computing

GRID technologies support the sharing and coordinated use of interconnected resources in dynamic virtual organizations. In the beginning, the focus was on computing power: the dynamic creation from geographically and organizationally distributed components of a virtual computing system able to generate *autonomously* the service consisting of dynamically allocated computing resources for heavy processes [Foster *et al.* , 2001]. In the beginning GRID technologies were designed for advanced science and engineering oriented services consisting mainly of processing power. Since then [Foster *et al.* , 2002] the ambition of GRID computing has shifted its original goal to address any kind of electronic services such as those related to interactive processes, as it is the case, for instance, of persistent transactions with distributed Information Systems, e-Commerce and e-Learning applications, Ambient intelligence and so on.

The main logical components of a GRID architecture are the following: computing elements, representing the computers and clusters that run users' processes and jobs; storage elements, that represent the storage space where temporary and persistent data are stored; user interfaces, represent the services that permit to the final user to access and use the offered resources of the GRID architecture; resource brokers, these are the central elements of a GRID architecture that handle the users' requests, allocate the desired resources and track the

execution of the users' requests and finally inform the users about the result of their requests; information services, offering a 'yellow page' service that informs about the available resources and location within the GRID.

The current central debate on GRID technologies (GLOBUS OGSA/I [Alliance, n.d.a, Alliance, n.d.b] and WSRF [Alliance, n.d.c]) concerns the most suitable architecture that hosts two contradictory properties of distributed systems: their independence on the asynchronicity of message exchange requiring a purely functional behaviour in order to avoid a heavy synchronization control or the risk of unforeseeable failures and the need for modeling, deploying and using persistent, stateful transactions (interactions, conversations). The MIC* model represents a progress for a grounded solution for these concerns.

6.3.2 Multi-Agent Based Simulations

The MIC* framework has already been used as an infrastructure for agent based simulations. The main advantage of using MIC* model in agent based simulation is the fact that this model implements the influence/reaction principle [Ferber & Muller, 1996] and separates the interaction time from the computation of the agents. It is interesting to extend the work presented by F. Michel [Michel, 2004] in order to provide simulation oriented platforms that are based on the MIC* model to experiment the implementation of large-scale and distributed agent-based simulations.

6.3.3 Fault Tolerant Software Systems

Classical MASs approaches that do not provide a model of the deployment environment at the design level and do not consider the deployment environment as a first-class entity at the implementation level cannot capture all the parameters that characterize MASs. For instance, the state of a FIPA-based MAS is not defined when a message is sent by agent and is traveling across the infrastructure to reach its destination. Consequently, it is difficult to save the state of the entire MAS in order to restart it later. MIC* defines clearly the state of the deployment environment at any time. So, by knowing the states of the agents, the state of the entire MAS is known and can be stored and used to restart the system in case of partial or total failure. We are currently experimenting with a persistent implementation of MIC* the opportunity to build fault tolerant MASs that can support either the failure of the agents or the failure of the deployment environment.

6.3.4 Extending the Coordination Framework

We have presented a simple coordination framework that captures the flow of activities and resources that exchanged among the different roles played by agents. This framework can be extended in order to include other characteristics such as time constraints. Consequently, the coordination protocol does not only express the order of resources but also expresses the which is allocated for each activity to produce its resources. Again, the deployment environment should be able to check the consistency of the coordination protocols and conversations that occur between the agents.

6.3.5 Development Environments and Tools to Build EAC Applications

We have presented a preliminary organizational model and an IDE to develop EAC applications as open artificial societies. We expect to have feedbacks from developers that have experimented these tools to ameliorate the IDE and to provide an engineering methodology to facilitate the engineering process of real world EAC applications.

6.4 General Conclusion

The EAC offers a real opportunity to develop innovative services and applications that meet users' needs and expectations in various domains. The added value of EAC services is the fact that these services are no longer statically retained, as it is the case nowadays for *Internet* grounded services, but they can be located everywhere in ubiquitous manner.

From a technical point of view, the EAC challenges different assumptions that we have on software systems. In order to develop services and applications for the EAC context we have to forget some of our assumptions that were legated from distributed software systems that have been conceived for other technical contexts such as the LANs and WANs. So, the EAC has to be considered as a new technical context where to build software systems. Understanding the properties of this context is an essential point for the success of building innovative and stable applications.

After establishing the main properties of the EAC context, it is important to develop models and theories that are suitable to answer the EAC challenges. We have followed a formal approach and produced the MIC* algebraic model to abstract the deployment environment where the autonomous software entities interact. The formalisation is not the main objective but a mean that will help us to understand and answer some of the challenges imposed by the EAC. Consequently the applicability of the produced theories and models is very important. This point distinguishes between the concerns of mathematicians and computer scientists: while the former consider the development of models and theories as a core problem, the later considers the development of models and theories as a mean to resolve other problems that are more related to informatics. So, it is important to translate the models and theories to software structures and prototypes in order to check their applicability.

Even if the computer scientists play sometimes the role of the mathematician, she/he has to clearly understand the difference between its own role and the role of the mathematician. In fact, if the mathematician has to be compared to a poet, the computer scientist is then compared to the music composer that writes melodies for the poems and transforms them into songs. It will be more or less difficult to write some melodies for certain poems but this process is essential for the computer scientist.

On the other hand, it is not always easy to directly share formal models and theories specially when these models are complex and involve some prior knowledge. So, the use of metaphor helps in sharing knowledge and the diffusion of ideas. Still, the used metaphors have not to be independent from models and theories. In fact, we have to make the correspondence between the metaphor and the theories clear and explicit. For instance, in this thesis we have presented both the social metaphor for the development of EAC applications and its translation into MIC* and DBCM coordination protocols. So, the complexity of models and theories can be hidden by the use of metaphors that simplify the views. This is done to take into account an important factor of any activity that involves humans which is the human

factor. In fact, we have always to keep in mind that we are using computers, but as scientists and engineers we are always communicating with other humans.

Appendix A

Mathematical notations

- $\mathbb{N}$: set of positive integers.
- $\mathbb{Z}$: set of integers.
- $\mathbb{B}$: the Boolean ring.
- $\emptyset$: empty set (containing no elements).
- $\{a, b, c\}$: a set containing three elements a, b, c.
- $x \in X$: x is an element of the set X.
- $x \notin X$: x is not an element of the set X.
- $A \subset X$: the set A is a contained in X.
- X^*: the set or ordered sequences that can be empty built using elements of X.
- $|A|$: if A is a set then, $|A|$ is the number of its elements.
- $X - A$: elements of X that are not contained in A.
- $A_1 \cup A_2$: union set of A_1 and A_2.
- $\{\ x \mid x$ such that ...$\}$: set of elements x such that...
- $\exists x$: ...: exists an element x such that...
- $\forall x \in X$: for all element x in X.
- $A \times B$: cartesian product of A and B: set of couples (a, b) with $a \in A$ and $b \in B$.
- $\mathcal{P}(A)$: power set of A or the set of all the subsets of A.
- X^∞: the multiset or bag of elements of X.
- $\#(x, P)$: the cardinality of the element x in the bag P.
- $(P) \Rightarrow (Q)$: P implies Q.
- $(S \rightarrow T)$: set of all functions defined from S to T.
- $\lambda(x) : Q$: lambda style definition of a function; Q represents the core of the function. For instance $\lambda(x) : (x - 1)$ is a function that takes an argument x and return $x - 1$.

- $V^{(K)}$: the set of sequences of elements of V that are indexed by the elements of K.
- $[x_i]_{i \in Y}$: sequence of elements that are indexed by the element of the set Y.
- $a.P@S$: the agent P has a as interaction object within the space S.
- $G =< N, V >$: a graph where N represents the set of nodes and V the set of vertexes.
- n^+ : the set of successors of the node n within a graph.
- n^- : the set of predecessors of the node n within a graph.
- $u \xrightarrow{\sigma} u'$: u can be reduced to u' using the relation σ.
- $u \overset{R}{\rightsquigarrow} u'$: it exists a path of reductions that simplifies u to n'.
- $|u|$ if u is sequence of marked elements, then $|u|$ represent the sequence u where the elements are no longer marked.

Appendix B

Formal definition of Petri-Net Theory

B.1 MultiSet (Bag) Theory

The formal definition of Petri Net theory is based on the multiset theory, so this section presents a brief introduction to the multiset theory and notations.

The multiset is a set for which repeated elements are considered. In fact, within the set theory the repeated elements are considered only once. For instance, the set $\{1, 1, 3\}$ is exactly the same as the set $\{1, 3\}$. However, the multiset $\{1, 3\}$ is different from the multiset $\{1, 1, 3\}$ since the *cardinality* of the element 1 differs. The cardinality of an element x in a multiset Q is represented as $\#(x, Q)$. For instance: $\#(1, \{1, 3\}) = 1$ and $\#(1, \{1, 1, 3\}) = 2$.

We represent the multiset that is built from a set X as follows X^∞. The multiset X^∞ allows the repetition of the elements of X.

The classical operations that are defined for the sets such as union, intersection, inclusion are naturally extended for the multiset by considering the cardinality of the elements. Here some examples:

$$\begin{aligned} \{1,2\} \cup \{1,3,4\} &= \{1,1,2,3,4\} \\ \{1,2\} \cup \{1,3,4\} &= \{1\} \\ \{1,2\} &\subset \{1,1,2,3,4\} \end{aligned}$$

B.2 Petri Net Structure

Definition B.2.1. *Petri net structure is a four tuple $C = \{P, T, I, O\}$, $P = \{p_1, p_2, ..., p_n\}$ is a finite set of places $n \geq 0$ and $T = \{t_1, t_2, ..., t_m\}$ is a finite set of transitions $m \geq 0$. $P \cup T = \emptyset$. $I : T \to P^\infty$ is defined as the input function, a mapping from transitions to bags of places and $O : T \to P^\infty$ is defined as the output function, a mapping from transitions to bags of places.*

- A place p_i is an input place of a transition t_i if $p_i \in I(t_j)$; p_i is an output place if $p_i \in O(t_j)$. The inputs and outputs of a transition are bags of places.

- Multiplicity of a place p for transition t is the number of occurrences of place p in input or output transitions bag that is:
  - Multiplicity of input place p_i for transition t_j $\#(p_i, I(t_j))$
  - Multiplicity of output place p_i for transition t_j $\#(p_i, O(t_j))$
- Transition t_j is an input of a place p_i if p_i is an output of p_j. Transition t_j is an output of a place p_i if p_i is an input of t_j.

Definition B.2.2. *The definition of the output and input functions is extended for places as follows:* $O : P \to T^\infty$ *and* $I : P \to T^\infty$ *such that:*

- $\#(t_j, I(p_i)) = \#(p_i, O(t_j))$
- $\#(t_j, O(p_i)) = \#(p_i, I(t_j))$

B.3 Petri Net Markings

A marking μ is an assignment of tokens to the places.

Definition B.3.1. *A marking* μ *of a Petri net* $C = (P, T, I, O$ *is a function defined from the set of places to the non negative integers* $\mathbb{N}$.

$$\mu : P \to \mathbb{N}$$

A marked Petri net $M = (C, \mu)$ *is a Petri net structure* $C = (P, T, I, O)$ *and marking function* μ.

B.4 Execution Rules for Petri Nets

The execution of Petri net is totally controlled by tokens distributed over places. Petri net executes by *firing* transitions. A transition fires by consuming tokens from its input places and producing new tokens that are distributed to its output places. A transition is said to be fireable if it is enabled. A transition is enabled if each of its input places has at least as many tokens in it as arcs from the place to the transition.

Definition B.4.1. *Transition* $t_j \in T$ *in a marked Petri net C = (P, T, I, O) with marking* μ *is enabled if for all* $p_i \in P$*:*

$$\mu(p_i) \geq \#(p_i, I(t_j))$$

Definition B.4.2. *Let* t_j *be a fireable transition in a marked Petri net with marking* μ*. The firing of this transition results in a new marking* μ' *defined as follows:*

$$\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j))$$

When there are no fireable transitions the petri net is said to be *dead*.

Appendix C

MIC* Message Headers

C.1 General Definition of the Message Structure

```
#ifndef __Message__
#define __Message__
/***************************************************************************
 * MIC* Library/Common Package
 * Class: _Message
 * Author: A. GOUAICH
 * Version: £version£
 * Description: Base class for messages
 *
 ***************************************************************************/
#include "global.h"
#include <map>
#include <string>
#include <iostream>
using namespace std;
//IDs of Messages
const unsigned char PONG_MSG                    =0xFF;
const unsigned char REMOTE_INTERACTION_REQUEST_MSG =0x0F;
const unsigned char REMOTE_INTERACTION_REPLY_MSG =0x0E;
const unsigned char MOVEMENT_REQUEST_MSG        =0x0D;
const unsigned char LOGOUT_MSG                  =0x0C;
const unsigned char LOGIN_MSG                   =0x0B;
const unsigned char LOGIN_RESULT_MSG            =0x0A;
const unsigned char INBOX_REQUEST_MSG           =0x09;
const unsigned char INBOX_REPLY_MSG             =0x08;
const unsigned char DISCONNECT_REQUEST_MSG      =0x07;
const unsigned char DISCONNECT_FROM_MSG         =0x06;
const unsigned char CONNECTION_REQUEST_MSG      =0x05;
const unsigned char CONNECTION_REPLY_MSG        =0x04;
const unsigned char CONNECT_TO_MSG              =0x03;
const unsigned char COMPUTATION_REQUEST_MSG     =0x02;
const unsigned char AK_MSG                      =0x01;
const unsigned char PING_MSG                    =0x00;
//abstract Message Class
class _Message
{
//pretty printing
friend ostream& operator<<(ostream& output, const _Message& msg);
public:
//the type of the message
unsigned char message_type;
//constructor
_Message();
_Message(unsigned char type);
//destructor
~_Message();
```

```
virtual pair<char*, int> encode() = 0;
};
//function used to the decode the messages from binary encoding
_Message* decode(const pair<char*, int>&);
//internal type
typedef struct
{
unsigned char flag;
unsigned int length ;
char* content;
} message;
//function to serialize the messages.
pair<char*,int> serialize(message& msg);
//function to deserialize the messages.
message deserialize(const pair<char*,int>& res);
#endif
```

C.2 Agent to MIC* Message Headers

C.2.1 Login Request Message

```
#ifndef __LOGIN_H__
#define __LOGIN_H__
/*************************************************************************
 * MIC* Library/Common Package
 * Class: LoginResult
 * Author: A. GOUAICH
 * Version: £version£
 * Description: Message sent by the agent to MIC*
 * to request to log into the deployment environment
 *************************************************************************/

#include "Message.h"
#include <iostream>
using namespace std;

/**
 * Login Message
 **/

class Login: public _Message
{
//pretty printing
friend ostream& operator<<(ostream& output, const Login& login);
public:
//agent identity
string user_identity;
//passwd
string passwd;
//constr.
Login();
Login(string username, string passwd);
Login(pair<char*,int> coding);
virtual ~Login();
//code the msg
pair<char*, int> encode() ;
};

#endif
```

C.2.2 Login Reply Message

```
#ifndef __LOGIN__
#define __LOGIN__

/************************************************************************
 * MIC* Library/Common Package
 * Class: LoginResult
 * Author: A. GOUAICH
 * Version: £version£
 * Description: Message sent by MIC* the agent to inform
 * about the result of the login request.
 ************************************************************************/

#include "Message.h"
#include <iostream>
using namespace std;

/**
 * LoginResult Message
 **/
class LoginResult: public _Message{
friend ostream& operator<<(ostream& output, const LoginResult& loginres);
public:
//login result
bool login_result;
//agent identity
string agent_entry;
//constr.
LoginResult();
LoginResult(bool result, string username);
LoginResult(pair<char*,int> coding);
virtual ~LoginResult();

//Ask to code the message
pair<char*, int> encode() ;
};

#endif
```

C.2.3 Logout Request Message

```
#ifndef __LOGOUT_H__
#define __LOGOUT_H__

/************************************************************************
 * MIC* Library/Common Package
 * Class: LogoutRequest
 * Author: A. GOUAICH
 * Version: £version£
 * Description: Message sent an Agent to MIC* to request
 * to logout.
 ************************************************************************/

#include "Message.h"
#include <iostream>
using namespace std;

/**
 * Logout Message
 * parameters: agent_entry
 *
 **/

class LogoutRequest: public _Message
{
//pretty printing
friend ostream& operator<<(ostream& output, const Logout& login);
```

```
public:
//Agent entry: agent identifier
string agent_entry;
//Default constructor
Logout();
//constructor with parameter
Logout(const string& id);
//Constructor from binary coding
Logout(const pair<char*,int>& coding);
//default desctructor
~Logout();
//ask to encode the message
pair<char*, int> encode() ;
};

#endif
```

C.2.4 Inbox Request Message

```
#ifndef __INBOXREQUEST_H__
#define __INBOXREQUEST_H__
/************************************************************************
 * MIC* Library/Common Package
 * Class: InboxResult
 * Author: A. GOUAICH
 * Version: £version£
 * Description: Message sent by the agent to MIC*
 * to ask its inbox
 ************************************************************************/
#include "Message.h"
#include <iostream>
using namespace std;

/**
 * InboxRequest Message
 **/

class InboxRequest: public _Message
{
//pretty printing
friend ostream& operator<<(ostream& output, const InboxRequest& InboxRequest);
public:
//agent identity
string aentry;
//constr.
InboxRequest();
InboxRequest(string aentry);
InboxRequest(pair<char*,int> coding);
~InboxRequest();
//ask to code the msg
pair<char*, int> encode() ;
};
#endif
```

C.2.5 Inbox Reply Message

```
#ifndef __INBOX_REPLY_H__
#define __INBOX_REPLY_H__
/************************************************************************
 * MIC* Library/Common Package
 * Class: InboxReply
 * Author: A. GOUAICH
 * Version: £version£
 * Description: Message sent by the agent to MIC*
```

```
 * to request to log into the deployment environment
 ************************************************************************/
#include "Message.h"
#include "InteractionObjectMultiSet.h"
#include <iostream>
#include <map>
#include "Codec.h"
using namespace std;
/**
 * InboxReply Message
 **/

class InboxReply: public _Message
{
//pretty printing
friend ostream& operator<<(ostream& output, const InboxReply& comp);
public:
//agent identity
string aentry;
//the inbox mapping interaction space->{Interaction Object}
map<string,InteractionObjectMultiSet*> inbox;
InboxReply();
InboxReply(string aentry, map<string,InteractionObjectMultiSet*> inbox);
InboxReply(pair<char*,int> coding);
virtual ~InboxReply();
//codes the msg.
pair<char*, int> encode() ;
};

#endif
```

C.2.6 Movement Request Message

```
#ifndef __MOVEMENT_REQUEST_H__
#define __MOVEMENT_REQUEST_H__
/************************************************************************
 * MIC* Library/Common Package
 * Class: MovementRequest
 * Author: A. GOUAICH
 * Version: £version£
 * Description: Message sent by an agent to MIC* to move
 * its interaction objects between interaction space
 ************************************************************************/
#include "Message.h"
#include <iostream>
using namespace std;

/**
 * MovementRequest Message
 **/

class MovementRequest: public _Message
{
//pretty printing
friend ostream& operator<<(ostream& output, const MovementRequest& login);
public:
//agent entry
string aentry;
//source
string source;
//destination
string destination;
//constr.
MovementRequest();
MovementRequest(const string& aentry,const string& source);
MovementRequest(const string& aentry,const string& source,const string& destination);
MovementRequest(pair<char*,int> coding);
```

```
35 //destructor.
36 ~MovementRequest();
37 //codes the msg.
38 pair<char*, int> encode() ;
39 };
40 #endif
```

C.2.7 Movement Reply Message

C.2.8 Computation Request Message

```
1  #ifndef __COMPUTATION_H__
2  #define __COMPUTATION_H__
3
4  /*************************************************************************
5   * MIC* Library/Common Package
6   * Class: ComputationRequest
7   * Author: A. GOUAICH
8   * Description: Message sent by the Agent to change its outboxes
9   * within the MIC* deployment environment.
10  *************************************************************************/
11 /*
12  *HEADERS
13  */
14 #include "Message.h"
15 #include "InteractionObjectMultiSet.h"
16 #include "Codec.h"
17 #include <iostream>
18 #include <map>
19 using namespace std;
20 /*
21  * END OF HEADERS
22  */
23
24 /**
25  * ComputationRequest Message
26  **/
27
28 class ComputationRequest: public _Message
29 {
30 //output method
31 friend ostream& operator<<(ostream& output, const Computation& comp);
32
33 public:
34 //entry: the agent identifier in the deployment environment
35 string entry;
36 //outbox: mapping between the interaction spaces and
37 //interction objects multiset
38 map<string,InteractionObjectMultiSet*> outbox;
39 //Default constructor
40 ComputationRequest();
41 //Constructor with parameters
42 ComputationRequest(string entry, map<string,InteractionObjectMultiSet*> outbox);
43 //Constructor from binary encoding
44 ComputationRequest(pair<char*,int> coding);
45 //Destructor
46 ~Computation();
47 //Encodes the message to binary data in order
48 //to be sent accross the network.
49 pair<char*, int> encode() ;
50 };
51
52 #endif
```

C.2.9 Computation Reply Message

```
#ifndef __COMPUTATION_REPLY_H__
#define __COMPUTATION_REPLY_H__

/**********************************************************************
 * MIC* Library/Common Package
 * Class: ComputationRequest
 * Author: A. GOUAICH
 * Description: Message sent by the Agent to change its outboxes
 * within the MIC* deployment environment.
 **********************************************************************/
/*
 *HEADERS
 */
#include "Message.h"
#include "InteractionObjectMultiSet.h"
#include "Codec.h"
#include <iostream>
#include <map>
using namespace std;
/*
 * END OF HEADERS
 */

/**
 * ComputationReply Message
 **/

class ComputationReply: public _Message
{
//output method
friend ostream& operator<<(ostream& output, const Computation& comp);

public:
//entry: the agent identifier in the deployment environment
string entry;
//outbox: mapping between the interaction spaces and
//the result of the computation
map<string,bool> result;
//Default constructor
ComputationReply();
//Constructor with parameters
ComputationReply(string entry, map<string,bool> outbox);
//Constructor from binary encoding
ComputationReply(pair<char*,int> coding);
//Destructor
~Computation();
//Encodes the message to binary data in order
//to be sent accross the network.
pair<char*, int> encode() ;
};

#endif
```

C.3 MIC* to MIC* Message Headers

C.3.1 Connection Request Message

```
#ifndef __ConnectionRequest_H__
#define __ConnectionRequest_H__

/**********************************************************************
 * MIC* Library/Common Package
 * Class: ComputationRequest
```

```
 * Author: A. GOUAICH
 * Description: Message sent by a MIC* instance to another MIC*
 * to request a composition.
 ************************************************************************/

#include "Message.h"
#include <iostream>
#include <set>
using namespace std;

/**
 * ConnectionRequest Message
 *
 * Parameters: login, password, hostname, port, spaces
 *
 **/

class ConnectionRequest: public _Message
{
//output method
friend ostream& operator<<(ostream& output, const ConnectionRequest& DisconnetFrom);
public:
//login for secured composition
string login;
//passwd for secured compositions
string passwd;
//the hostname
string hostname;
//int port
int port;
//set of string identifying public interaction spaces
set<string> spaces;
//Default constructor
ConnectionRequest();
//full parameters constr.
ConnectionRequest(string login, string passwd, string hostname, int port, const set<string
    >& spaces);
//constructor from the binary data
ConnectionRequest(pair<char*,int> coding);
//Destructor
~ConnectionRequest();

//Encodes the message to binary data in order
//to be sent accross the network.
pair<char*, int> encode() ;
};
#endif
```

C.3.2 Connection Reply Message

```
#ifndef __ConnectionReply_H__
#define __ConnectionReply_H__

/************************************************************************
 * MIC* Library/Common Package
 * Class: ComputationRequest
 * Author: A. GOUAICH
 * Description: Message sent by a MIC* instance to another MIC*
 * to inform if the connection is accepted or not.
 ************************************************************************/
/*
 *HEADERS
 */
#include "Message.h"
#include <iostream>
#include <set>
using namespace std;
```

```
18
19 /**
20  * ConnectionReply Message
21  * Parameters: login, passwd, result.
22  *
23  **/
24
25 class ConnectionReply: public _Message
26 {
27 friend ostream& operator<<(ostream& output, const ConnectionReply& DisconnetFrom);
28
29 public:
30 //login: used for secured compositions
31 string login;
32 //passwd: used for secured compositions
33 string passwd;
34 //the result of the connection
35 bool result;
36
37 //set of string identifying public interaction spaces
38 //that will be synchronized on interactions.
39 set<string> spaces;
40
41 //Default Constructor
42 ConnectionReply();
43
44 //full parameters constructor
45 ConnectionReply(string login, string passwd, bool result,const set<string>& spaces);
46
47 //Constructor from the binary encoding
48 ConnectionReply(pair<char*,int> coding);
49
50 //Destructor
51 ~ConnectionReply();
52
53 //Encodes the message to binary data in order
54 //to be sent accross the network.
55 pair<char*, int> encode() ;
56 };
57 #endif
```

C.3.3 Diconnection Request Message

```
1  #ifndef __DisconnectRequest_H__
2  #define __DisconnectRequest_H__
3  /*************************************************************************
4   * MIC* Library/Common Package
5   * Class: DisconnetFrom
6   * Author: A. GOUAICH
7   * Version: £version£
8   * Description: Message sent by an instance of MIC* to another MIC*
9   * to disconnect from it
10  *************************************************************************/
11 #include "Message.h"
12 #include <iostream>
13 #include <set>
14 using namespace std;
15 /**
16  * DisconnectRequest Message
17  *
18  **/
19 class DisconnectRequest: public _Message
20 {
21 //pretty printing
22 friend ostream& operator<<(ostream& output, const DisconnectRequest& DisconnetFrom);
23 public:
24 //command login
```

```
25 string login;
26 //passwd
27 string passwd;
28 //local hostname
29 string hostname;
30 //int the remote port
31 int port;
32 DisconnectRequest();
33 DisconnectRequest(string login, string passwd, string hostname, int port);
34 DisconnectRequest(pair<char*,int> coding);
35 ~DisconnectRequest();
36 //codes the message
37 pair<char*, int> encode() ;
38 };
39 #endif
```

Appendix D

DTD and XML Definitions

D.1 Dependency Based Coordination Model

D.1.1 DTD Definition

```
<!-- Coordination Model Formalism Definition -->
<!--
Definition of the XML formalism to express coordination models based
on producer/consumer relationships among activies.
CoordinationGraph: the overall structure
ResourceNode: represents the abstract resource representation in the
coordination graph.
ActivityNode: represents an activity in the coordination model.
ControlFunction: a boolean function that recognise the actual resources.
ProductionLink: Production relation among an activty and a resource.
ControlFunction: Consumption relation among an activty and a resource.
Author: GOUAICH Abdelakder, version 1.1, date 15/07/2004
-->

<!ELEMENT CoordinationGraph (
                              ResourceNode+,
                              ActivityNode+,
                              ProductionLink+,
                              ConsumptionLink+,
                              ControlFunction+
                              ) >

<!ATTLIST CoordinationGraph name ID #REQUIRED>

<!ELEMENT ResourceNode EMPTY >
<!-- unique id of the resource-->
<!ATTLIST ResourceNode id ID #REQUIRED >
<!-- label used when drawing the digraph -->
<!ATTLIST ResourceNode label CDATA #REQUIRED>
<!-- tells if this resource node is an axiom or not-->
<!ATTLIST ResourceNode axiom (true|false) #REQUIRED>
<!-- tells if this resource node is mapped to a control function-->
<!ATTLIST ResourceNode epsilon (true|false) #REQUIRED>
<!ATTLIST ResourceNode group CDATA #REQUIRED >
```

```
<!-- sets The control function of this resource.
If this resource is not an epsilon this attribute must be set-->
<!ATTLIST ResourceNode control_function IDREF #IMPLIED >
<!ELEMENT ActivityNode EMPTY >
<!-- unique id of the activity-->
<!ATTLIST ActivityNode id ID #REQUIRED >
<!-- label used when drawing the digraph -->
<!ATTLIST ActivityNode label CDATA #REQUIRED>
<!ATTLIST ActivityNode group CDATA #REQUIRED>

<!ELEMENT ControlFunction (#PCDATA) >
<!-- the code of the function in include in a CDATA section -->
<!-- name and id of the control function -->
<!ATTLIST ControlFunction name ID #REQUIRED>
<!ELEMENT ProductionLink EMPTY >
<!-- idref of the resource-->
<!ATTLIST ProductionLink resource IDREF #REQUIRED>
<!-- idref of the activity-->
<!ATTLIST ProductionLink activity IDREF #REQUIRED>
<!ELEMENT ConsumptionLink EMPTY >
<!-- idref of the resource-->
<!ATTLIST ConsumptionLink resource IDREF #REQUIRED>
<!-- idref of the activity-->
<!ATTLIST ConsumptionLink activity IDREF #REQUIRED>
```

D.1.2 Examples of Coordination patterns expressed using the XML representation:

D.2 XML Formalism For the Specification of Open Artificial Societies

```
<!-- AEC Application Specification -->
<!--
     XML Formalism used for the specification of AEC applications
       as open artificial societies.
       Author: GOUAICH Abdelakder, version 1.1, date 15/09/2004
-->

<!--
    The root element is the Project. The project is composed by two
    subconcepts the ProjectInformation node and the OpenArtificialSociety node.
-->

<!ELEMENT Project (ProjectInformation,OpenArtificialSociety) >
<!-- name of the project -->
<!ATTLIST Project name       CDATA  #REQUIRED >
<!-- namespace of the project -->
<!ATTLIST Project namespace  CDATA  #REQUIRED >

<!-- Text description of the project-->
<!ELEMENT ProjectInformation (Description) >
```

```
<!--
OpenArtificialSociety:
SystemicFunctionSet: set of functions that the system should implement.
SocialStructure: the structure of the artificial society.
SocialMapping: mapping between the social strtructure and the functions.
-->
<!ELEMENT OpenArtificialSociety (SystemicFunctionSet,SocialStructure,SocialMapping) >
<!ATTLIST OpenArtificialSociety name       CDATA  #REQUIRED >
<!ATTLIST OpenArtificialSociety namespace  CDATA  #REQUIRED >

<!-- SYSTEMIC FUNCTION -->
<!ELEMENT SystemicFunctionSet (SystemicFunction*) >
<!ELEMENT SystemicFunction (Description) >
<!-- Identification and name of the socialtask -->
<!ATTLIST SystemicFunction name     CDATA  #REQUIRED >
<!ATTLIST SystemicFunction id  ID          #REQUIRED >

<!-- SOCIAL STRUCTURE -->
<!ELEMENT SocialStructure (RoleSet,GroupSet,GroupRoleAssociationSet, Constraint) >

<!-- The role organisational concept-->
<!ELEMENT RoleSet (Role*) >
<!ELEMENT Role (Description) >
<!ATTLIST Role id     ID                   #REQUIRED >
<!ATTLIST Role name CDATA                  #REQUIRED >
<!ATTLIST Role type (internal|interface)   #REQUIRED >

<!-- The group concept: a group is a cllection of roles performing a social task-->
<!ELEMENT GroupSet (Group*) >
<!ELEMENT Group (Description) >
<!ATTLIST Group id           ID            #REQUIRED >
<!ATTLIST Group name  CDATA  #REQUIRED >

<!-- Association between the group and the role -->
<!ELEMENT GroupRoleAssociationSet (GroupRoleAssociation*) >
<!ELEMENT GroupRoleAssociation (Description) >
<!ATTLIST GroupRoleAssociation group_id    IDREF #REQUIRED >
<!ATTLIST GroupRoleAssociation role_id  IDREF #REQUIRED >

<!-- Constraint -->
<!ELEMENT Constraint (RoleConstraintSet,GroupConstraintSet, GroupRoleConstraintSet) >

<!-- Role Constraint -->
<!ELEMENT RoleConstraintSet (RoleConstraint*) >
<!ELEMENT RoleConstraint (Description) >
<!ATTLIST RoleConstraint role_id    IDREF #REQUIRED >
<!ATTLIST RoleConstraint min_occ    NMTOKEN #REQUIRED >
<!ATTLIST RoleConstraint max_occ    NMTOKEN #REQUIRED >

<!-- Group Constraint -->
<!ELEMENT GroupConstraintSet (GroupConstraint*) >
<!ELEMENT GroupConstraint (Description) >
<!ATTLIST GroupConstraint group_id  IDREF #REQUIRED >
<!ATTLIST GroupConstraint min_occ   NMTOKEN #REQUIRED >
```

```
<!ATTLIST GroupConstraint max_occ   NMTOKEN #REQUIRED >

<!-- GroupRole Constraint -->
<!ELEMENT GroupRoleConstraintSet (GroupRoleConstraint*) >
<!ELEMENT GroupRoleConstraint (Description) >
<!ATTLIST GroupRoleConstraint group_id  IDREF #REQUIRED>
<!ATTLIST GroupRoleConstraint role_id IDREF #REQUIRED>
<!ATTLIST GroupRoleConstraint min_occ NMTOKEN #REQUIRED>
<!ATTLIST GroupRoleConstraint max_occ NMTOKEN #REQUIRED>

<!-- Mapping between the social structure and the functions -->
<!ELEMENT SocialMapping (SocialTaskSet) >

<!-- SocialTask Collection concept -->
<!ELEMENT SocialTaskSet (SocialTask*) >
<!-- Social task concept -->

<!ELEMENT SocialTask (GroupAssignment, CoordinationProtocolSet) >
<!-- Identification and name of the socialtask -->
<!ATTLIST SocialTask name CDATA #REQUIRED >
<!ATTLIST SocialTask id ID          #REQUIRED >
<!ATTLIST SocialTask implemented_functions IDREFS #REQUIRED >
<!ELEMENT GroupAssignment (RoleIDSet)>
<!ATTLIST GroupAssignment group_id  IDREF #REQUIRED >

<!ELEMENT RoleIDSet (RoleID*) >
<!ELEMENT RoleID EMPTY>
<!ATTLIST RoleID role_id     IDREF #REQUIRED >

<!-- Coordination protocol in order to resolve the social task-->
<!-- assigned to a roles that are in a social group     -->
<!ELEMENT CoordinationProtocolSet (CoordinationProtocol*)>
<!ELEMENT CoordinationProtocol (Description,CoordinationGraph) >
<!ATTLIST CoordinationProtocol id   ID #REQUIRED >
<!ATTLIST CoordinationProtocol name CDATA #REQUIRED >
<!ATTLIST CoordinationProtocol roles-id IDREFS #REQUIRED >
<!-- How to describe a coordination protocol-->

<!ELEMENT CoordinationGraph
       (ActivitySet,ResourceSet,CoordinationGraphStructure,ControlFunctionSet)
>
<!-- Acticity -->
<!ELEMENT ActivitySet (Activity*) >
<!ELEMENT Activity (Description) >
<!ATTLIST Activity name      CDATA  #REQUIRED >
<!ATTLIST Activity id        ID            #REQUIRED >
<!ATTLIST Activity role-id   IDREF  #REQUIRED >

<!-- Resource -->
<!ELEMENT ResourceSet (Resource*) >
<!ELEMENT Resource (Description) >
<!ATTLIST Resource name CDATA                    #REQUIRED >
<!ATTLIST Resource id  ID                              #REQUIRED >
<!ATTLIST Resource axiom (true|false)      #IMPLIED >
<!ATTLIST Resource control_function IDREF  #IMPLIED >
```

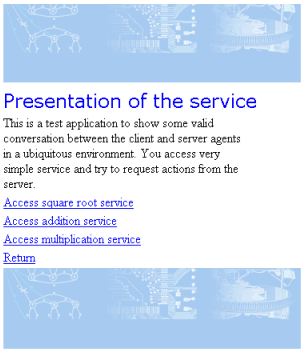
```
<!-- CoordinationGraphStructure -->
<!ELEMENT CoordinationGraphStructure (ProductionLinkSet,ConsumptionLinkSet ) >

<!ELEMENT ProductionLinkSet (ProductionLink*) >
<!ELEMENT ConsumptionLinkSet (ConsumptionLink*) >

<!-- ProductionLink -->
<!ELEMENT ProductionLink EMPTY>
<!ATTLIST ProductionLink task-id IDREF #REQUIRED >
<!ATTLIST ProductionLink resource-id IDREF #REQUIRED >

<!-- ConsumptionLink -->
<!ELEMENT ConsumptionLink EMPTY>
<!ATTLIST ConsumptionLink task-id          IDREF #REQUIRED >
<!ATTLIST ConsumptionLink resource-id  IDREF #REQUIRED >

<!-- Description Concept -->
<!ELEMENT Description (#PCDATA)>

<!ELEMENT ControlFunctionSet (ControlFunction) >
<!ELEMENT ControlFunction (#PCDATA) >
<!ATTLIST ControlFunction id        ID #REQUIRED >
```

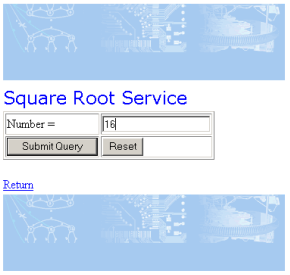
Appendix E

Testing the Conversations of the Ubiquitous Web Application

This chapter presents a test application has been developed for the ubiquitous application example. This application offers some arithmetic services such as the calculation of a sum, multiplication and square root of numbers. The table presented below shows the messages exchanged during a conversation (the first column), what the user actually sees displayed (the second column), and the result of the protocol controller (the third column):

<table>
<tr><th>conversation</th><th>output of the client agent</th><th>result of the protocol controller</th></tr>
<tr><td>
<table>
<tr><th colspan="2">CoordinationMessage</th></tr>
<tr><td>conv-id</td><td>1000000F</td></tr>
<tr><td>proto-id</td><td>WebSocialActivity</td></tr>
<tr><td>sender-role</td><td>WebClientRole</td></tr>
<tr><td>receiver-role</td><td>WebServerRole</td></tr>
<tr><td>content</td><td>(request room 0)</td></tr>
</table>
</td><td>no output</td><td>correct</td></tr>
<tr><td>
<table>
<tr><th colspan="2">CoordinationMessage</th></tr>
<tr><td>conv-id</td><td>1000000F</td></tr>
<tr><td>proto-id</td><td>WebSocialActivity</td></tr>
<tr><td>sender-role</td><td>WebServerRole</td></tr>
<tr><td>receiver-role</td><td>WebClientRole</td></tr>
<tr><td>content</td><td>(room 0 [base64 encoding of the room])</td></tr>
</table>
</td><td> 
</td><td>correct</td></tr>
</table>

<table>
<tr>
<td>
<table>
<tr><th colspan="2">CoordinationMessage</th></tr>
<tr><td>conv-id</td><td>1000000F</td></tr>
<tr><td>proto-id</td><td>WebSocialActivity</td></tr>
<tr><td>sender-role</td><td>WebClientRole</td></tr>
<tr><td>receiver-role</td><td>WebServerRole</td></tr>
<tr><td>content</td><td>(request room services)</td></tr>
</table>
</td>
<td>not changed</td>
<td>correct</td>
</tr>
<tr>
<td>
<table>
<tr><th colspan="2">CoordinationMessage</th></tr>
<tr><td>conv-id</td><td>1000000F</td></tr>
<tr><td>proto-id</td><td>WebSocialActivity</td></tr>
<tr><td>sender-role</td><td>WebServerRole</td></tr>
<tr><td>receiver-role</td><td>WebClientRole</td></tr>
<tr><td>content</td><td>(room services [base64 encoding of the room])</td></tr>
</table>
</td>
<td> 
</td>
<td>correct</td>
</tr>
<tr>
<td>
<table>
<tr><th colspan="2">CoordinationMessage</th></tr>
<tr><td>conv-id</td><td>1000000F</td></tr>
<tr><td>proto-id</td><td>WebSocialActivity</td></tr>
<tr><td>sender-role</td><td>WebClientRole</td></tr>
<tr><td>receiver-role</td><td>WebServerRole</td></tr>
<tr><td>content</td><td>(request room square-root-service)</td></tr>
</table>
</td>
<td>not changed</td>
<td>correct</td>
</tr>
</table>

CoordinationMessage conv-id: 1000000F proto-id: WEBSOCIALACTIVITY sender-role: WEBSERVERROLE receiver-role: WEBCLIENTROLE content: (room square-root-service [base64 encoding of the room])		correct
CoordinationMessage conv-id: 1000000F proto-id: WEBSOCIALACTIVITY sender-role: WEBCLIENTROLE receiver-role: WEBSERVERROLE content: (request action square-root number=16)	not changed	correct
CoordinationMessage conv-id: 1000000F proto-id: WEBSOCIALACTIVITY sender-role: WEBSERVERROLE receiver-role: WEBCLIENTROLE content: (room square-root-result [base64 encoding of the room])		correct
(request room 0)	not changed	correct

CoordinationMessage	
conv-id	`1000000F`
proto-id	WebSocialActivity
sender-role	WebClientRole
receiver-role	WebServerRole
content	`(room 0 [base64 encoding of the room])`

correct

Appendix F

Introduction to Networking Technologies

Network technologies are communication media (cf. §2.2.1, page 22) which enable the communication among software systems. The properties of the communication medium affect considerably both the qualitative and quantitative properties of distributed software systems. For instance L. Cardelli in [Cardelli, 1999] identifies the *mentalistic images* that software designers have on the technological context of software systems. Three main mentalistic images are presented by Cardelli: *(i)* the Local Area Network (LAN) image, *(ii)* the Wide Area Network (WAN) image; *(iii)* and the Mobile computing image. Caredelli shows that there is a correlation between the mentalistic image that is adopted and the properties of the software system. Consequently, before studying software architectures and design paradigms, one has to understand the lower-level considerations that result from the properties of communication media.

This section gives a simple introduction to these technologies for readers who are not familiar with this field. For further readings, the reader is forwarded to the Tannenbaum's book [S.Tanenbaum, 1989].

F.1 The OSI Model

The *International Standardisation Organisation* (ISO) has proposed a generic network architecture for the *Open Systems Interconnection* (OSI) [Day & Zimmerman, 1983]. The OSI model is organized as seven hierarchical layers. Each layer offers *services* to upper-layers and uses the services of sub-layers. The boundary between two layers is called an *interface* and the set of conventions allowing same-level layers to communicate is called a *protocol*.

The OSI layers are presented in Figure F.1 and are described as follows:

1. **Physical layer**: The role of this layer is to transmit on the physical media signals that encode information as data units (*bits*). The questions that are answered in this layer are related to properties of the physical media such as the electrical signal's voltage; the temporal duration of a bit in microseconds; the pins' format and order.

2. **Data link layer**: The role of this layer is to hide the physical properties of the transmission media. In other words, this layer simulates a perfect connection to the network

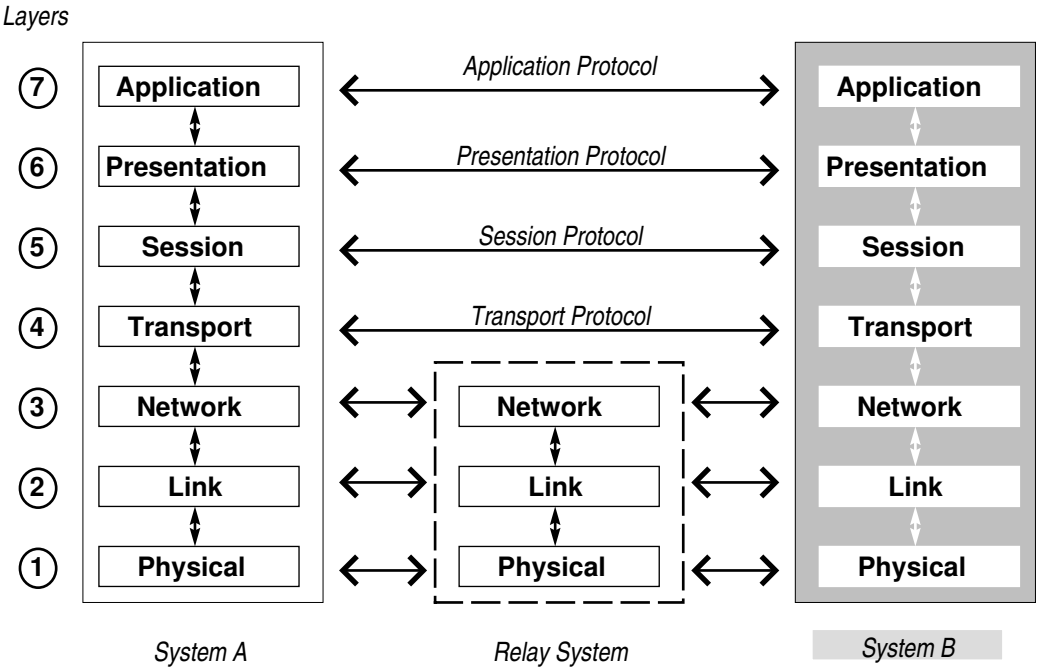


Figure F.1: The ISO7498 network architecture.

layer. Unlike the physical layer, this layer interprets and recognizes the transmitted data units namely the *(frames)*. For instance, when an error occurs on a frame it is immediately retransmitted without replication. The synchronization is another functionality of this layer. Indeed, in an open system, a transmitter might emit more quickly than a slow receiver. This causes the overloading of the receiver. To avoid this problem, some flow management mechanisms are implemented by this layer.

3. **Network layer**: This layer delivers *packets* from the emitter to an identified receiver. Again, some flow management mechanisms are used in order to control the traffic on the network.

4. **Transport layer**: The main function of this layer is to split data of the application into sub-units and to ensure that they arrive in the correct order.

5. **Session layer**: This layer allows different users on distinct machines to establish sessions. Besides, some recovery points are introduced to avoid the complete retransmission of data in case of failure.

6. **Presentation layer**: This layer allows the conversion of internal representation of data into independent abstract representations. This layer can be also concerned with other aspects such as compression and encryption.

7. **Applications layer**: The software entities using the lower layers can be placed in this layer.

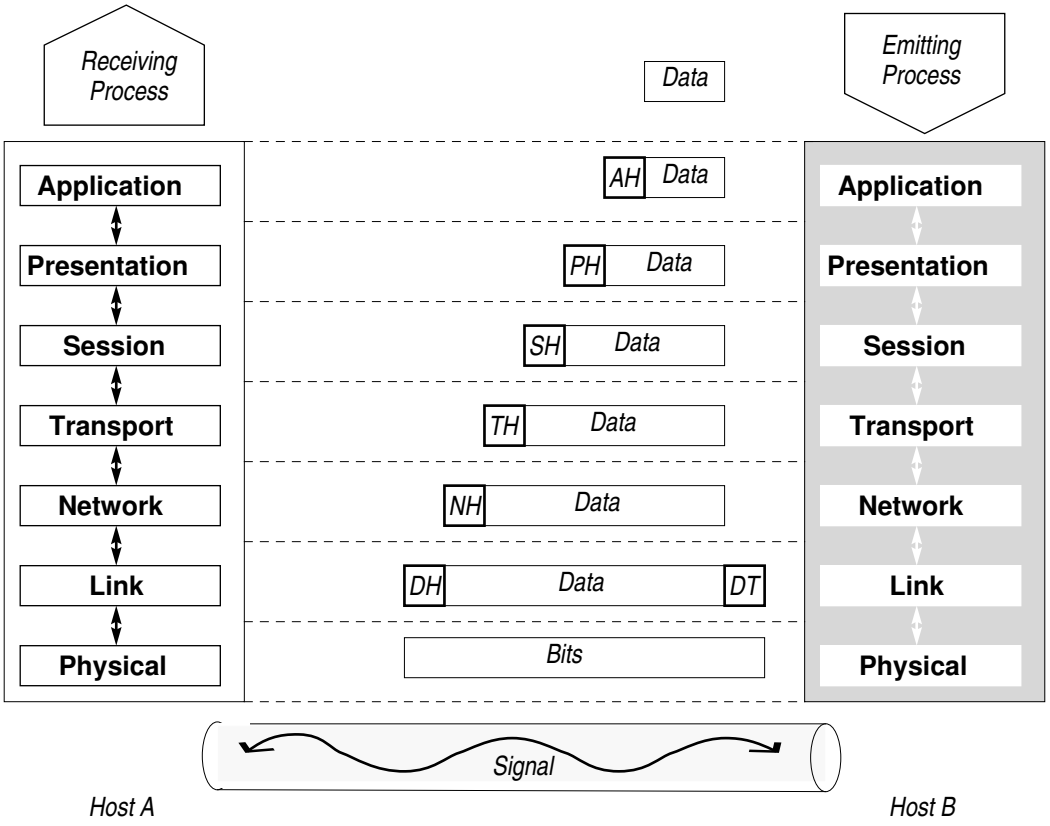


Figure F.2: Example of a data transmission[S.Tanenbaum, 1989].

F.2 How Does it Work?

In order to have a more precise idea about the process of the OSI model, this paragraph considers a simple data transmission that is illustrated in Figure F.2. The transmitter's data are delivered to the application layer, which adds an application header (AH). Then, both data and application header are delivered to the presentation layer, which might transform them (compression or encryption) and adds a presentation header (PH). This mechanism is repeated recursively until the physical layer by adding the headers corresponding to each layer: session (HS), transport (TH), network (NH) and data link (DH and DT). Finally, the physical layer considers the whole data as a rough continuation of bits that are transmitted on the physical medium as a signal.

The fundamental concept in the OSI model is the fact that same level protocols interact horizontally using vertical flows. For instance, two transport protocols interact by using their respective network layers, without being concerned with how the later layers will achieve their services. The example cited in [S.Tanenbaum, 1989] considers a diplomat talking to his counterparts in an international assembly. For that, the diplomat talks to his interpreter, who communicates the message to other interpreters. The message is then translated into each of the diplomat's language. At this time, the diplomats' level communication will be established.

F.3 Networking Technologies

F.3.1 TCP/IP

The American Defence Department (DOD) has developed the TCP/IP protocol stack in order to interconnect various network technologies resulting from different manufacturers. This has brought later the term of *Internet* as the INTERconnection of NETworks. The DOD put forth recommendations on robustness and automatic error recovery. To answer these recommendations, the TCP/IP architecture was decentralized and made flexible. This innovating architecture will be a key for the Internet worldwide success. The IP protocol fills the layer three in the OSI model [Day & Zimmerman, 1983]; it is responsible for packets routing through the network's nodes until a destination, identified by a single number[1] known also as the IP address. The TCP protocol is situated at level four in the OSI model. It is responsible for error detection and recovery.

F.3.2 Mobile IP and IPv6

Mobile IP and *IPV6* are IETF [IETF, 1986] standards. Mobile IP develops architecture and protocol for device transparent mobility over sub-networks and different transmission media. IPV6 is regarded as an extension of the IP protocol [Rakotonirainy, 1999]. IPV6 extends the address field and introduces of the quality of service *(QoS)* for packet routing.

F.3.3 Bluetooth

Bluetooth is a short-range communication technology that was initially conceived to replace wire connections between various devices. Today, it is considered as the model

[1] This number called also the IP address is encoded on 32 bits for Ipv4 and 64bits for Ipv6.

of short-range communication between various devices in a delimited geographical area [Haartsen *et al.* , 1998]. The covering zone of a Bluetooth device varies between 10 meters, with a 0 dbm as output power, and 100 meters, with 20 dbm as output power. Two types of connections are defined by Bluetooth: a point-to-point connection, emulating a serial link; and a point to multipoint connection, on a *piconet* network which can include up to 7 active devices and 200 stationary devices [Kammann *et al.* , 2001]. Bluetooth specifications endeavor to reduce the devices' cost with the guarantee a certain robustness [Haartsen *et al.* , 1998] in order to gain visibility in a market where the IrDA technology is already established.

F.3.4 Infrared Data Association (IrDA)

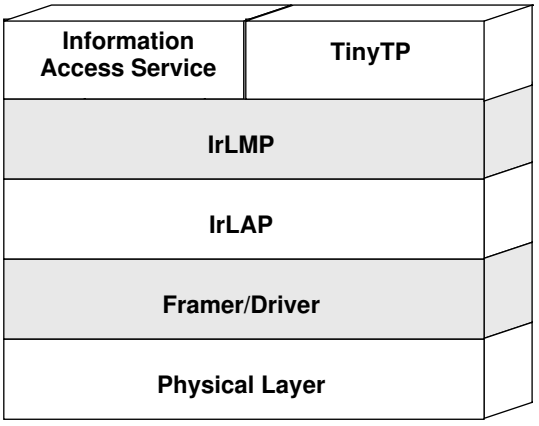


Figure F.3: The IrDA protocol stack [Knutson & Folsom, 1999].

The *Infrared Data Association* (IrDA) [IRDA, 1994] was formed on June 1993. It suggests a short-range communication standard using infrared signals. As shown in Figure F.3, the IrDA protocol stack is composed of the following layers: the physical layer, composed of an infrared transceiver propagating infrared signals in a cone ranging from 15 to 30 degrees; the framer, which transforms the data into a hardware readable format; the driver, enabling access to hardware services; the IrLAP (Link Access Protocol) is responsible for the discovery of IrDA entities present in vicinity and negotiation of best transmission rate expressed in *bauds*; the IrLMP (Link Protocol Manager) protocol deals with the multiplexing of several communications on a single IrLAP connection. The TinyTP protocol is responsible for the flow management, data segmentation and gathering; and the IAS (Information Access Service) protocol, that is the minimal service required for all the IrDA devices, represents a registry table where applications identify their services by a unique Link Service Access Point (LSAP).

Besides these elementary protocols, IrDA proposes other protocols to facilitate the development of applications in a mobile environment. This package was proposed in 1997 and is composed of four protocols presented in Figure F.4:

- IrOBEX: This protocol can be seen as an adaptation of the HTTP protocol for the IrDA environment. It is used for atomic exchange of objects between computers.
- IrCOMM: This protocol facilitates the integration of legacy systems that used COM ports.

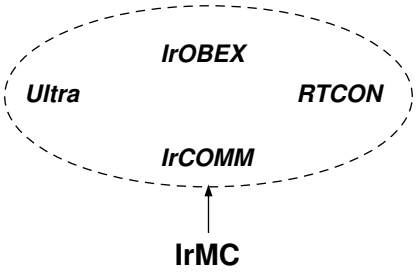


Figure F.4: Protocols of the IrMC.

- RTCONN (Real Time Transfer Protocol Control): This protocol is dedicated to the transmission of data flow with temporal constraints.
- Ultra: This protocol is particularly adapted to small and very restricted devices. It proposes a minimal set of functionalities for data exchange without maintaining the connection between the devices.

F.3.5 IEEE 802.11

The 802.11 [IEEE, 1963] standard defines radio frequencies (RF) for physical and media access layers of local area networks (LAN). Recently, *IEEE 802.11* was considered also as a short-range communication solution with the definition of the personal area networks (PAN) working group.

F.3.6 HomeRF

The home radio frequency working group (HRFWG)[HomeRF, 2003] has developed the specifications intended to introduce networks technologies into personal houses environments. The whole house's devices would be managed as a local area network using the radio frequencies as a transmission media.

F.3.7 HiperLAN

The High Performance LAN (HiperLAN) standard family is developed by the working group RES 10 (Radio operator Equipment and Systems) of the IETF [IETF, 1986]. It proposes a standardization of the layers 1 and 2 in the OSI model [Day & Zimmerman, 1983] to ensure compatibility between various heterogeneous equipments. The upper layers are not concerned with the specifications and are left free for constructors specific needs [Korhonen, 1999][Rajaniemi, 1999][Johnsson, 1999].

F.3.8 DECT

Digital Enhanced Cordless Telecommunication (DECT) is a standard developed by the technical committee *Radio Equipment and Telecommunication*(RES-03) of the ETSI [ETSI, 1992]. The offered services are described as follows:

1. great capacity of access to the cellular network;
2. mobility of the equipment in the network;
3. robustness in hostile environments.

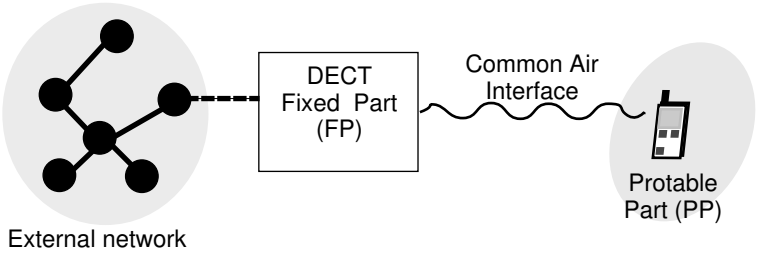


Figure F.5: DECT common radio interface.

As illustrated in Figure F.5, the DECT architecture is composed of a fixed part (FP) and a portable part (PP); the basic standard takes into account only the specification of the air interface between these two components.

F.3.9 Global System for Mobile Communication (GSM)

Global System for Mobile Communication (GSM) operates in 900 Hz, 1800 Hz and 1900 Hz frequencies. It is in vogue in Europe and the Asia-Pacific area. The GSM is used by more than 215 million people (October 1999 [Durlacher, 2003]) which represents more than 50% of the users of mobile terminals.

F.3.10 HSCSD

High Speed Switched Circuit Dated (HSCSD) is a GSM based protocol. Theoretically, it can transmit 4 times more than the GSM by using 4 data communication channels simultaneously.

F.3.11 GPRS

General Packet Radio operator Service (GPRS) is a switched packet protocol. It allows a transmission of 115 kb/s. The major advantage of the GPRS, except its significant transmission rate, is the always-on connection mode. This mode connects a mobile terminal to the network instantaneously, without establishing an explicit and expensive[2].

F.3.12 EDGE

Enhanced Data Rate for Global Evolution (EDGE) is an evolution of the GPRS, which can reach up to 384 kb/s. It is also regarded as a intermediate technology between the GPRS and the UMTS.

[2]GSM connection establishment time is about 40 seconds

	Centralized communication medium	Decentralized communication medium	Hosts Mobility
TCP/IP	yes	no	no
IPv6	yes	no	no
Mobile IP	yes	no	yes
Bluetooth	no	yes	yes
IrDA	no	yes	yes
IEEE 802.11	yes	yes	yes
HomeRF	no	yes	yes
DECT	no	yes	yes
GSM	yes	no	yes
GPRS	yes	no	yes
EDGE	yes	no	yes
UMTS	yes	no	yes

Table F.1: Properties of network technologies

F.3.13 UMTS

Universal Mobile Telephony System (UMTS) is the third generation of mobile telephony system. Often, the UMTS was associated the theoretical flow of 2 Mb/s. However, reaching this flow in practice requires an important investment on the infrastructures. The real flow would be only of 384 kbit/s at least up to 2005 [Durlacher, 2003].

F.3.14 Analysis of the Network Technologies

Table F.1 presents an overview of the presented technologies according to some features described as follows:

- centralized communication medium: within centralized communication media the devices are connected into a conceptually centralized network architecture. Generally, centralized network architectures are expensive and imply some security and control access policies.
- decentralized communication medium: within this model of communication media each device is equipped with communication facilities allowing peer to peer connections. Generally, these architectures are low-cost and easy to manage.
- Hosts mobility: This criterion studies if the computer is still able to use the communication medium while moving. Physical mobility of hosts is an important feature since mobility of computers implies necessarily mobility of the executed software systems and hence enabling their ubiquity.

The values for each criterion range from: yes, no and n/a when the criterion is irrelevant for a particular network technology.

From Table F.1 one can see that nowadays communication technologies are already offering the opportunity to enable EAC applications. Three main classes of network technologies can be identified:

- Category 1: This category represents network technologies that do not handle the mobility of devices and defines a centralized communication schema. This category is not suitable for the EAC since the criteria of availability of services everywhere is hardly achievable. TCP/IP and IPv6 fit in this category.
- Category 2: This category represents network technologies that handle the mobility of devices and with a centralized communication schema. This category is suitable for the EAC since the devices are still connected to the network while moving. However, the high-cost of these architectures may limit their use. For instance, the Wireless Application Protocol (WAP) was an unsuccessful commercial experiment in Europe.
- Category 3: This category represents network technologies that handle the mobility of devices and with a decentralized communication schema. This category is suitable for the EAC since the devices are still connected to the network while moving. However, they imply a physical proximity among the computers. So, the set of applications developed with these constraints may be limited.

For the development of EAC applications a hybrid communication medium that merges the technologies of category 2 and 3 is needed. Hence, when computers are physically distant, the centralized communication architecture is exploited and when the physical context of computer allows establish a point-to-point communication link, the use of low-cost technologies of category 3 is privileged.

References

[Agha, 1986]Agha, G. 1986. *ACTORS: A Model of Concurrent Computation in Distributed Systems.* The MIT Press, Cambridge, MA.

[Agha & Hewitt, 1985]Agha, Gul, & Hewitt, Carl. 1985. *Concurrent Programming Using Actors: Exploiting Large-Scale Parallelism.* Document Number: AI Memo No 865. MIT.

[Alliance, n.d.a]Alliance, The Globus. *The Open Grid Services Architecture (OGSA).* http://www.globus.org/ogsa.

[Alliance, n.d.b]Alliance, The Globus. *The Open Grid Services Infrastructure (OGSI).* http://www-unix.globus.org/toolkit.

[Alliance, n.d.c]Alliance, The Globus. *The WS-Resource Framework.* http://www-fp.globus.org/wsrf.

[Alvez & Yovine, 1991]Alvez, Rogelio, & Yovine, Sergio. 1991 (August). Distributed Implementation of a Linda Kernel. *In: XVII conference Latinoamericana de Informatica PANEL'91.*

[Barbuceanu & Fox, 1995]Barbuceanu, Mihai, & Fox, Mark S. 1995. COOL: A Language for Describing Coordination in Multi-Agent Systems. *Pages 17–24 of:* Lesser, Victor (ed), *First International Conference on Multi-Agent Systems.* San Francisco, California: AAAI Press/The MIT Press.

[Barbuceanu & Lo, 2000]Barbuceanu, Mihai, & Lo, Wai-Kau. 2000. Conversation Oriented Programming for Agent Interaction. *In:* [Dignum & Greaves, 2000].

[Bellifemine *et al.* , 1999]Bellifemine, F., Poggi, A., & Rimassi, G. 1999. JADE: A FIPA-Compliant Agent Framework. *Pages 97–108 of: Practical Applications of Intelligent Agents and Multi-Agents.*

[Bellifemine *et al.* , 2003]Bellifemine, F., Caire, G., Poggi, A., & Rimassa, G. 2003. *JADE A White Paper.* Document Number: V03N03Art01. Telecom Italia Lab, exp.telecomitalialab.com.

[Berger *et al.* , 2003]Berger, M., Rusitschka, S., Schlichte, M., Toropov, D., & Watzke, M. 2003. *Porting Agents to Small Mobile Devices The Development of the Lightweight Extensible Agent Platform.* Document Number: V03N03Art04. Telecom Italia Lab exp.telecomitalialab.com.

[Berger, 2002]Berger, Michael. 2002. *Agents in Ad Hoc Environments A Whitepaper.* Document Number: f-in-00068. FIPA, www.fipa.org.

[Berners-Lee *et al.* , 1994]Berners-Lee, T., Masinter, L., & McCahill, M. 1994. *RFC 1738 - Uniform Resource Locators (URL).* Document Number: RFC 1738. CERN and Xerox Corporation and University of Minnesota.

[Birrell & Nelsen, 1984]Birrell, D., & Nelsen, B.J. 1984. Implementing Remote Procedure Call. *ACM Transactions on Computer Systems*, **2(1)**.

[Bond & Gasser, 1988]Bond, A.H, & Gasser, L. 1988. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers.

[Booch, 1994]Booch, G. 1994. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley Professional. Second edition.

[Breiter & Sadek, 1996]Breiter, P., & Sadek, M.D. 1996. A Rational Agent as a Kernel of a Cooperative Dialogue System: Implementing a Logical Theory of Interaction. *Pages 261–276 of: ECAI-96 Workshop Agent Theories, Architectures, and Languages*.

[Cabri *et al.* , 2001]Cabri, G., Leonardi, L., & Zambonelli, F. 2001. *Engineering Mobile-Agent Applications via Context-dependent Coordination*.

[Caire *et al.* , 2003]Caire, G., Lhuillier, N., & Rimassa, G. 2003. *A Communication Protocol for Agents on Handheld Devices*. Document Number: V02N03ART295. Telecom Italia Lab, exp.telecomitalialab.com.

[Caire *et al.* , 2001]Caire, Giovanni, Coulier, Wim, Garijo, Francisco J., Gomez, Jorge, Pavon, Juan, Leal, Francisco, Chainho, Paulo, Kearney, Paul E., Stark, Jamie, Evans, Richard, & Massonet, Philippe. 2001. Agent Oriented Analysis Using Message/UML. *Pages 119–135 of: Agent Oriented Software Engineering Workshop (AOSE)*.

[Cardelli, 1999]Cardelli, Luca. 1999. Abstractions for Mobile Computation. *Secure Internet Programming*, 51–94.

[Castelfranchi, 1995]Castelfranchi, C. 1995. Guarantees for Autonomy in Cognitive Agent Architecture. *Intelligent Agents: Theories, Architectures, and Languages*, **890**, 56–70.

[Cerri, 1999]Cerri, S. A. 1999. Shifting the focus from control to communication: the STReams OBjects Environments model of communicating agents. *Collaboration between Human and Artificial Societies, Coordination and Agent-Based Distributed Computing*, **1624**(3), 71–101.

[Cerri *et al.* , 2004]Cerri, Stefano A., Eisenstadt, Marc, & Jonquet, Clement. 2004. Dynamic Learning Agents and Enhanced Presence on the Grid. *In: Electronic Workshops in Computing (eWiC), 3rd International LeGE-WG Workshop: GRID Infrastructure to Support Future Technology Enhanced Learning*.

[Charlton & Mamdani, 1999]Charlton, Patricia, & Mamdani, E. H. 1999. A Developer's Perspective on Multi-agent System Design. *Pages 41–51 of: MAAMAW '99: Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. London, UK: Springer-Verlag.

[Chen & Finin, 2002]Chen, Harry, & Finin, Tim. 2002. Beyond Distributed AI, Agent Teamwork in Ubiquitous Computing. *Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices, AAMAS-2002*.

[Cost *et al.* , 2000]Cost, R. Scott, Chen, Ye, Finin, Timothy W., Labrou, Yannis, & Peng, Yun. 2000. Using Colored Petri Nets for Conversation Modeling. *In:* [Dignum & Greaves, 2000].

[Day & Zimmerman, 1983]Day, D., & Zimmerman, H. 1983 (December). The OSI reference model. *Pages 1334–1340 of: IEEE*, vol. 71.

[DBMI, 2000]DBMI. 2000. *Business Process Management Initiative*. www.bpmi.org.

[Deloach, 2001]Deloach, S. 2001. *Analysis and Design using MaSE and agentTool*.

[Demazeau, 1995]Demazeau, Yves. 1995. From interaction to collective behaviour in agent-based systems. *In: 1st European Conference on Cogntive Science*.

[Deutsch, 1996]Deutsch, P. 1996. *RFC 1951: DEFLATE Compressed Data Format Specification version 1.3*. Document Number: RFC 1951. Network Working Group, Aladdin Enterprises.

[Deutsch & Gailly, 1996]Deutsch, P., & Gailly, J-L. 1996. *RFC 1950 - ZLIB Compressed Data Format Specification version 3.3*. Document Number: RFC 1950. Aladdin Enterprises and Info-ZIP.

[Dignum & Greaves, 2000]Dignum, Frank, & Greaves, Mark (eds). 2000. *Issues in Agent Communication*. Lecture Notes in Computer Science, vol. 1916. Lecture Notes in Computer Science: Springer.

[d'Inverno & Luck, 1996]d'Inverno, M., & Luck, M. 1996. A Formal View of Social Dependence Networks. *Pages 115–129 of:* Zhang, & Lukose (eds), *Distributed Artificial Intelligence Architecture and Modelling: Proceedings of the First Australian Workshop on Distributed Artificial Intelligence*. Springer-Verlag: Heidelberg, Germany.

[Douglas *et al.* , 1995]Douglas, Andrew, Rowston, Anotony, & Wood, Alan. 1995. *ISETL-LINDA: Parallel Programming with Bags*. Document Number: YCS 257. Department of Computer Science, The University of York, UK.

[Durlacher, 2003]Durlacher. 2003. *Mobile Commerce Report*. Document Number: no number. Durlacher Research, www.durlacher.com.

[E4MAS, 2004]E4MAS. 2004. *Environment for Multi-Agent Systems (E4MAS)*.

[Eastlake & Kaufman, 1997]Eastlake, D., & Kaufman, C. 1997. *RFC 2065 - Domain Name System Security Extensions*. Document Number: RFC 2065. CyberCash and Iris.

[Erman *et al.* , 1980]Erman, Lee D., Hayes-Roth, Frederick, Lesser, Victor R., & Reddy, Raj. 1980. *The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty*.

[ETSI, 1992]ETSI. 1992. *European Telecommunications Standards Institute, www.etsi.org*.

[Fallah-Seghrouchni *et al.* , 1999]Fallah-Seghrouchni, Amal El, Haddad, Serge, & Mazouzi, Hamza. 1999. Protocol Engineering for Multi-agent Interaction. *Pages 89–101 of:* Garijo, Francisco J., & Boman, Magnus (eds), *MAAMAW*. Lecture Notes in Computer Science, vol. 1647. Springer.

[Ferber & Gutknecht, 1998]Ferber, J., & Gutknecht, O. 1998. *A meta-model for the analysis and design of organizations in multi-agent systems*.

[Ferber, 1999]Ferber, Jacques. 1999. *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence*. England: Addison Wesley Longman.

[Ferber & Muller, 1996]Ferber, Jacques, & Muller, Jean-Pierre. 1996. Influences and Reactions : a Model of Situated Multiagent Systems. *Pages 72–79 of: Proceedings of Second Internationnal Conference on Multi-Agent Systems (ICMAS96),*.

[Ferber, 1995]Ferber, Jaques. 1995. *Les Systemes Multi-Agents.* InterEditions.

[Ferber *et al.* , 2003]Ferber, Jaques, Gutknecht, Olivier, & Michel, Fabien. 2003. From Agents to Organizations: An Organizational View of Multi-Agent Systems. *Agent Oriented Software Engineering Workshop (AOSE'03).*

[Fielding *et al.* , 1999]Fielding, R., Irvine, UC, Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. 1999. *Hypertext Transfer Protocol – HTTP/1.1.* Document Number: RFC 2616. The Internet Society, Network Working Group.

[Finin *et al.* , 1994]Finin, Tim, Fritzson, Richard, McKay, Don, & McEntire, Robin. 1994. KQML as an Agent Communication Language. *In: Third International Conference on Information and Knowledge Management (CIKM'94).* ACM Press.

[FIPA, 1996]FIPA. 1996. *Foundation of Intelligent and Physical Agents.* http://www.fipa.org.

[FIPA, 2001]FIPA. 2001. *FIPA ACL Message Structure Specification.* Document Number: XC00061E. FIPA.

[FIPA, 2002a]FIPA. 2002a. *FIPA ACL Message Representation in Bit-Efficient Encoding Specification.* Document Number: SC00069G. FIPA, www.fipa.org.

[FIPA, 2002b]FIPA. 2002b. *FIPA Message Buffering Service Specification.* Document Number: XC00092B. FIPA, www.fipa.org.

[FIPA, 2002c]FIPA. 2002c. *FIPA SL Content Language Specification.* Document Number: SC00008I. FIPA.

[FIPA, 2004]FIPA. 2004. *FIPA Agent Management Specification.* Document Number: SC00023K. FIPA, www.fipa.org.

[Foster *et al.* , 2001]Foster, I., Kesselman, C., & Tuecke, S. 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, **15(3)**.

[Foster *et al.* , 2002]Foster, I., Kesselman, C., Nick, J., & Tuecke, S. 2002. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG.*

[Fournet, 1998]Fournet, Cedric. 1998. *Le Join-Calcul: Un Calcul Pour la Programmation Repartie et Mobile.* Ph.D. thesis, Ecole Polytechnique.

[Fox, 1981]Fox, M. 1981. An Organisational View of distributed Systems. *IEEE transacations on systems, man and cybernetics*, **11**(1).

[Fuxman *et al.* , 2001]Fuxman, A., Giorgini, P., Kolp, M., & Mylopoulos, J. 2001. Information Systems as Social Structures. *In: Proceedings of the Second International Conference on Formal Ontologies for Information Systems (FOIS2001).*

[Gallimore *et al.* , 1998]Gallimore, R.J., Jennings, N.R., Lamba, H.S., Mason, C.L., & Orenstein, B.J. 1998. 3D Scientific Data Interpretation using Cooperating Agents. *Pages 47–65 of: 3rd Int. Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-98).*

[Gasser *et al.* , 1987]Gasser, L., Braganza, C., & Herman, N. 1987. MACE: A Flexible Testbed for Distributed AI Research. *Pages 119–152 of:* Huhns, M. N. (ed), *Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann.

[Gelernter *et al.* , 1985]Gelernter, D., Carriero, N., Chandran, S., & Chang, S. 1985 (August). Parallel Programming in Linda. *Pages 255–263 of: Proceedings of the International Conference on Parallel Programming*.

[Gelernter, 1985]Gelernter, David. 1985. Generative Communication in Linda. *ACM Transaction od Programming Languages and Systems*, **7**(1), 80–112.

[Genesereth & Fikes, 1994]Genesereth, Michael R., & Fikes, Richard E. 1994. *Knowledge Interchange Format, Version 3.0, Reference Manual*. Stanford Logic Group.

[Genesereth & Ketchpel, 1994]Genesereth, M.R., & Ketchpel, S.P. 1994. Software Agents. *Communications of the ACM*, **37(7)**, 48–53.

[Giorgini *et al.* , 2003]Giorgini, Paolo, Kolp, Manuel, & Mylopoulos, John. 2003. Organizational Patterns for Early Requirements Analysis. *In: Conference On Advanced Information Systems Engineering (CAiSE*03).*

[Giunchiglia *et al.* , 2001a]Giunchiglia, F., Mylopoulos, J., & Perini, A. 2001a. *The Tropos Software Development Methodology: Processes*.

[Giunchiglia *et al.* , 2001b]Giunchiglia, Fausto, Perini, Anna, & Sannicolo', Fabrizio. 2001b. *Knowledge Level Software Engineering*. Document Number: DIT-02-007. Informatica e Telecomunicazioni, University of Trento.

[Glaser, 1996]Glaser, Norbert. 1996. *Contribution to Knowledge Modelling in a Multi-Agent Framework (the Co-MoMAS Approach)*. Ph.D. thesis, LUniverstit e Henri Poincar e, Nancy I.

[Gouaïch, 2003]Gouaïch, Abdelkader. 2003. Requirements for Achieving Software Agents Autonomy and Defining their Responsibility. *In: Autonomy Workshop at AAMAS 2003*.

[Gudgin *et al.* , 2003]Gudgin, Martin, Hadley, Marc, Mendelsohn, Noah, Moreau, Jean-Jacques, & Nielsen, Frystyk. 2003. *SOAP Version 1.2 Part 1: Messaging Framework*. Document Number: REC-soap12-part1-20030624. W3C.

[Guessoum & Briot, 1999]Guessoum, Zahia, & Briot, Jean-Pierre. 1999. From Active Objects to Autonomous Agents. *IEEE Concurrency*, **7**(3), 68–76.

[Gutknecht & Ferber, 2000a]Gutknecht, Olivier, & Ferber, Jacques. 2000a. *MadKit A generic multi-agent platform*. Autonomous Agents 2000 -Barcelona. http://www.madkit.org.

[Gutknecht & Ferber, 2000b]Gutknecht, Olivier, & Ferber, Jacques. 2000b. *MadKit A generic multi-agent platform*. Autonomous Agents 2000 -Barcelona. http://www.madkit.org.

[Haartsen *et al.* , 1998]Haartsen, J., Naghshineh, M., Inouye, J., Joeressen, O., & Allen, W. 1998. Bluetooth: Vision, Goals, and Architecture. *Mobile Computing and Communications Review*, **2**(4), 38–45.

[Harel, 1987]Harel, David. 1987. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, **8**(3), 231–271.

[Hawick *et al.* , 2002]Hawick, K.A., James, H.A., & Pritchard, L.H. 2002. *Tuple-Space Based Middelware for Distributed Computing*. Document Number: DHPC-128. Computer Sicence Division, School of Informatics, University of Wales, UK.

[Hewitt, 1976a]Hewitt, C.E. 1976a (Decemder). *Viewing Control Structures as Patterns of Message passing*. Document Number: AI MEMO 410. MIT.

[Hewitt, 1976b]Hewitt, C.E. 1976b (Decemder). *Viewing Control Structures as Patterns of Message Passing*. Document Number: AI MEMO 410. MIT.

[Hoare, 1985]Hoare, C. A. R. 1985. *Communicating Sequential Processes*. Prentice-Hall International.

[HomeRF, 2003]HomeRF. 2003. *HomeRF Working Group, www.palowireless.com/homerf*.

[IEEE, 1963]IEEE. 1963. *IEEE, www.ieee.org*.

[IETF, 1986]IETF. 1986. *The Internet Engineering Task Force (IETF), www.ietf.org*.

[Iglesias *et al.* , 1999]Iglesias, Carlos, Garrijo, Mercedes, & Gonzalez, Jose. 1999. A Survey of Agent-Oriented Methodologies. *Pages 317–330 of:* Muller, Jorg, Singh, Munindar P., & Rao, Anand S. (eds), *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, vol. 1555. Springer-Verlag: Heidelberg, Germany.

[Iglesias *et al.* , 1996]Iglesias, Carlos A., Garijo, Mercedes, e C. Gonz alez, Jos, & Velasco, Juan R. 1996. A methodological proposal for multiagent systems development extending CommonKADS. *Pages 17–25 of: 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, vol. 1.

[IRDA, 1994]IRDA. 1994. *Infrared Data Association (IrDA), www.irda.org*.

[ITU, 1993]ITU. 1993. *Recommendation Z.120: Message Sequence Chart (MSC)*. Document Number: Z.120. International Telecommunication Union.

[Jennings, 1993]Jennings, N. R. 1993. Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. *The Knowledge Engineering Review*, **8**(3), 223–250.

[Jennings & Wooldridge, 1997]Jennings, Nicholas R., & Wooldridge, Michael. 1997. Agent-based software engineering. *Pages 26–37 of: Software Engineering*, vol. 144. IEEE.

[Johnsson, 1999]Johnsson, M. 1999. *HIPERLAN/2 The Broadband Radio Transmission Technology Operating in the 5 GHz Frequency Band*.

[Kammann *et al.* , 2001]Kammann, Jens, Strang, Thomas, & Wendlandt, Kai. 2001. Mobile services over short range communication. *In: Workshop Commercial Radio Sensors and Communication Techniques*.

[Karp, 1993]Karp, Alan. 1993. *Some Experiences with Network LINDA*. Document Number: G320. IBM Scientific Center, Palo Alto, CA 94304.

[Kendall, 1999]Kendall, E. A. 1999. Role Modelling for Agent System Analysis, Design, and Implementation. *In: 1st International Symposium on Agent Systems and Applications*. IEEE CS Press.

[Kettani *et al.* , 1998]Kettani, Nasser, Mignet, Dominique, Pare, Pascal, & Rosenthal-Sabroux, Camille. 1998. *De Merise a UML*. Eyrolles.

[Kinny *et al.* , 1996]Kinny, David, Georgeff, Michael, & Rao, Anand. 1996. A Methodology and Modelling Technique for Systems of (BDI) Agents. *In:* van Hoe, Rudy (ed), *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*.

[Knutson & Folsom, 1999]Knutson, Charles D., & Folsom, Brad. 1999. IrMC: Infrared Solutions for Mobile Communications. *In: Proceedings of the 1999 Embedded Systems Conference Summer*, vol. 71.

[Kolp & Mylopoulos, 2001]Kolp, Manuel, & Mylopoulos, John. 2001. Software Architectures as Organizational Structures. *In: ASERC Workshop on "The Role of Software Architectures in the Construction, Evolution, and Reuse of Software Systems"*.

[Kolp *et al.* , 2002]Kolp, Manuel, Giorgini, Paolo, & Mylopoulos, J. 2002. Information Systems Development through Social Structures. *In: 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*.

[König, 2003]König, Ralf. 2003. State-Based Modeling Method for Multiagent Conversation Protocols and Decision Activities. *Pages 151–166 of:* Kowalczyk, Ryszard, Müller, Jörg P., Tianfield, Huaglory, & Unland, Rainer (eds), *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*. Lecture Notes in Computer Science, vol. 2592. Springer.

[Korhonen, 1999]Korhonen, Janne. 1999. *HIPERLAN/2*.

[Kornfeld, 1979]Kornfeld, William A. 1979. *Using Parallel Processing for Problem Solving*. Document Number: AIM-561. MIT.

[Laukkanen *et al.* , 2002a]Laukkanen, Mikko, Tarkoma, Sasu, & Leinonen, Jani. 2002a. FIPA-OS Agent Platform for Small-Footprint Devices. *Pages 447–460 of: Published in the book Intelligent Agents VIII*.

[Laukkanen *et al.* , 2002b]Laukkanen, Mikko, Helin, Heikki, & Laamanen, Heimo. 2002b. Supporting nomadic agent-based applications in the FIPA agent architecture. *Pages 1348–1355 of: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. ACM Press.

[Law & McCann, 2000]Law, G., & McCann, J. 2000. A New Protection Model for Component-Based Operating Systems. *In: IEEE Conference on Computing and Communications*.

[Lenat, 1975]Lenat, Douglas B. 1975. *BEINGS: Knowledge as interacting experts*. International Joint Conference on Artificial Intelligence (IJCAI).

[Lesser & Corkill, 1983]Lesser, V., & Corkill, D.D. 1983. The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. *A I Magazine*, **4**(3), 15–33.

[Levesque *et al.* , 1990]Levesque, H. J., Cohen, P. R., & Nunes, J. H. T. 1990. On Acting Together. *Pages 94–99 of: The Eighth National Conference on Artificial Intelligence (AAAI-90)*.

[Leymann, 2001]Leymann, Frank. 2001. *Web Services Flow Language*. Document Number: WSFL (1.0). IBM.

[Lind, 2000]Lind, Jurgen. 2000. The Massive Development Method for Multiagent Systems. *Pages 339–354 of:* Bradshaw, Jeffrey, & Arnold, Geoff (eds), *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000).* Manchester, UK: The Practical Application Company Ltd.

[Luck & D'Inverno, 2001]Luck, M., & D'Inverno, M. 2001. Autonomy: A Nice Idea in Theory. *Intelligent Agents VII: Proceedings of the Seventh International Workshop on Agent Theories, Architectures and Languages.*

[Luck & d'Inverno, 1995]Luck, Michael, & d'Inverno, Mark. 1995. A formal framework for agency and autonomy. *Pages 254–260 of:* Lesser, Victor, & Gasser, Les (eds), *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95).* San Francisco, CA, USA: AAAI Press.

[Malone & Crowston, 1994]Malone, Thomas W., & Crowston, Kevin. 1994. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, **26**(1), 87–119.

[Mamei *et al.* , 2003]Mamei, Marco, Zambonelli, Franco, & Leonardi, Letizia. 2003. Programming Ubiquitous and Mobile Computing Applications with TOTA Middleware. *IEEE Distributed Systems Online*, 1–5.

[Martial, 1992]Martial, F. Von. 1992. *Coordinating Plans of Autonomous Agents.* Lecture Notes in AI, vol. 610. Berlin: Springer.

[McCann, 2002]McCann, Julie. 2002. Ubiquitous Systems - a New Challenge for Operating Systems Design? *In: 6th CaberNet Radicals Workshop.*

[McIlraith *et al.* , 2001]McIlraith, S. A., Son, T. Cao, & Zeng, H. 2001. Semantic Web Services. *IEEE Intelligent Systems*, **16(2)**, 46–53.

[Menezes *et al.* , 2001]Menezes, Ronaldo, Tolksdorf, Robert, & Wood, Alan M. 2001. Scalability in Linda-like Coordination Systems. *Chap. 12, pages 299–319 of:* Omicini, Andrea, Zambonelli, Franco, Klusch, Matthias, & Tolksdorf, Robert (eds), *Coordination of Internet Agents: Models, Technologies, and Applications.* Springer-Verlag.

[Michel, 2004]Michel, Fabien. 2004 (December). *Formalisme, méthodologie et outils pour la modélisation et la simulation de systèmes multi-agents.* Ph.D. thesis, Université Montpellier II.

[Milner, 1989]Milner, Robin. 1989. *Communication and Concurrency.* Prentice-Hall.

[Nwana *et al.* , 1999]Nwana, Hyacinth, Ndumu, Divine, Lee, Lyndon, & Collis, Jaron. 1999. ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. *Applied Artifical Intelligence Journal*, **13(1)**, 129–186.

[Odell *et al.* , 2000]Odell, J., Parunak, H., & Bauer, B. 2000. Extending UML for Agents. *In: In Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence.*

[OMG, 2002]OMG. 2002. *CORBA 3.0.* Document Number: formal/2-12-02. The Object Management Group.

[OMG, 2003]OMG. 2003. *OMG Specification of Unified Modeling Language.* Document Number: formal/03-03-01. Object Management Group (OMG).

[Omicini & Denti, 2001a]Omicini, A., & Denti, E. 2001a. From Tuple Spaces to Tuple Centres. *Science of Computer Programming*, **41(3)**, 277–294.

[Omicini, 2001]Omicini, Andrea. 2001. SODA: societies and infrastructures in the analysis and design of agent-based systems. *Pages 185–193 of: First international workshop, AOSE 2000 on Agent-oriented software engineering*. Springer-Verlag New York, Inc.

[Omicini & Denti, 2001b]Omicini, Andrea, & Denti, Enrico. 2001b. Formal ReSpecT. *Electronic Notes in Theoretical Computer Science.*, **48**, 179–196.

[Omicini & Zambonelli, 1998]Omicini, Andrea, & Zambonelli, Franco. 1998. TuCSoN: a Coordination model for Mobile Information Agents. *Pages 177–187 of:* Schwartz, David G., Divitini, Monica, & Brasethvik, Terje (eds), *1st International Workshop on Innovative Internet Information Systems (IIIS'98)*. Pisa, Italy: IDI – NTNU, Trondheim (Norway).

[Papadopoulos, 2000]Papadopoulos, G.A. 2000. Models and Technologies for the Coordination of Internet Agents: A Survey. *Chap. 2, pages 25–56 of:* Omicini, Andrea, Zambonelli, Franco, Klusch, Matthias, & Tolksdorf, Robert (eds), *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer-Verlag.

[Parunak & Odell, 2001]Parunak, H. Van Dyke, & Odell, James. 2001. Representing Social Structures in UML. *Agent Oriented Software Engineering Workshop AOSE'01*, 1–16.

[Parunak, 1997]Parunak, H.V.D. 1997. "Go to the Ant": Engineering Principles from Natural Multi-Agent Systems. *Annals of Operations Research*, **75**, 69–101.

[Perini *et al.* , 2001]Perini, A., Bresciani, P., Giunchiglia, F., Giorgini, P., & Mylopoulos, J. 2001. *A knowledge level software engineering methodology for agent oriented programming.*

[Rajaniemi, 1999]Rajaniemi, A. 1999. *HIPERLAN Overview.*

[Rakotonirainy, 1999]Rakotonirainy, Andry. 1999. Trends and Future of Mobile Computing. *Pages 136–140 of: DEXA Workshop.*

[Rao & Georgeff, 1995]Rao, A. S., & Georgeff, M. P. 1995. (BDI)-agents: from theory to practice. *In: Proceedings of the First Intl. Conference on Multiagent Systems.*

[Ricci *et al.* , 2004]Ricci, Alessandro, Viroli, Mirko, & Omicini, Andrea. 2004. Agent Coordination Context: From Theory to Practice. *Pages 618–623 of:* Trappl, Robert (ed), *Cybernetics and Systems 2004*, vol. 2. Vienna, Austria: Austrian Society for Cybernetic Studies. 17th European Meeting on Cybernetics and Systems Research (EMCSR 2004), Vienna, Austria, 13–16 Apr. 2004. Proceedings.

[Robin *et al.* , 1992]Robin, Milner, Joachim, Parrow, & David, Walker. 1992. A calculus for mobile processes, parts 1 and 2. *Information and Computation*, **100**(1).

[Russell & Norvig, 1995]Russell, Stuart, & Norvig, Peter. 1995. *Artificial Intelligence: A Modern Approach.* Prentice Hall.

[Saif & Greaves, 2001]Saif, Umar, & Greaves, David J. 2001. Communication Primitives for Ubiquitous Systems or RPC Considered Harmful. *In: ICDCS International Workshop on Smart Appliances and Wearable Computing.*

[Schreiber *et al.* , 1994]Schreiber, A., Wielinga, B., Akkermans, J., & de Velde, W. Van. 1994. *CommonKADS: A comprehensive methodology for KBS development.* Document Number: M1.2a KADS-II/M1/RR/UvA/70/1.1. University of Amsterdam and Netherlands Energy Research Foundation ECN and Free University of Brussels.

[Sichman *et al.* , 1994]Sichman, Jaime Simão, Conte, Rosaria, Castelfranchi, Cristiano, & Demazeau, Yves. 1994. A Social Reasoning Mechanism Based On Dependence Networks. *Pages 188–192 of:* Cohn, A. G. (ed), *Proceedings of the Eleventh European Conference on Artificial Intelligence.* Chichester: John Wiley & Sons.

[Simon, 1957]Simon, H. A. 1957. *Models of Man.* New York: Wiley.

[Singh, 1998]Singh, Munindar. 1998. Agent Communication Languages: Rethinking the Principles. *IEEE Computer*, 40–47.

[Smith, 1980]Smith, Reid. 1980. The Contract Net Protocol:High-Level Communication and Control in a Distributed Problem Solver. *IEEE Trans. on computers*, **29**(12), 1104–1113.

[SOCE, 2000]SOCE. 2000. *Workshop on Software and Organisation Co-evolution.*

[Spivey, 1987]Spivey, J.M. 1987. *The Z notation A reference manual.* Prentice Hall.

[S.Tanenbaum, 1989]S.Tanenbaum, Andrew. 1989. *Computer Networks.* Prentice-Hall, INC.

[Steels, 1995]Steels, Luc. 1995. When are robots intelligent autonomous agents? *Robotics and Autonomous Systems*, **15**, 3–9.

[Sun, 2000]Sun. 2000. *JavaSpaces^{TM} Service Specification.* Document Number: version 1.1. Sun Microsystems.

[Sun-Microsystems, 1994]Sun-Microsystems. 1994. *Java Remote Method Invocation - Distributed Computing for Java.* Document Number: . Sun Microsystems.

[Thatte, 2001]Thatte, Satish. 2001. *XLANG: Web Services for Business Process Design.* Document Number: XLANG. Microsoft Corporation.

[Turing, 1936]Turing, Alan. 1936. On Computable Numbers, With an Application to the Entscheidungsproblem. *In: Proceedings of the London Mathematical Society.* 2, no. 42.

[Vinoski, 1997]Vinoski, Steve. 1997. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine*, **14**(2).

[W3C, 2002a]W3C. 2002a. *Web Service Choreography Interface (WSCI) 1.0.* www.w3.org/TR/wsci.

[W3C, 2002b]W3C. 2002b. *Web Services Activity.* www.w3.org/2002/ws.

[W3C, 2004a]W3C. 2004a. *Extensible Markup Language (XML) 1.0 (Third Edition).* Document Number: REC-xml-20040204. W3C.

[W3C, 2004b]W3C. 2004b. *OWL Web Ontology Language.* Document Number: REC-owl-ref-20040210. W3C.

[W3C, 2004c]W3C. 2004c. *RDF/XML Syntax Specification.* Document Number: REC-rdf-syntax-grammar-20040210. W3C.

[WAP-Forum, 2002]WAP-Forum. 2002. *Wireless Application Protocol (WAP 2.0): Technical White Paper*. Document Number: WAP-0.2. Open Mobile Alliance.

[Weerawarana & Curbera, 2002]Weerawarana, Sanjiva, & Curbera, Francisco Paco. 2002. *Business Process with BPEL4WS: Understanding BPEL4WS, Part 1*. Document Number: ws-bpelcol1. IBM.

[Weiser, 1991]Weiser, M. 1991. The Computer for the 21st Century. *Scientific American*, **265(3)**, 94–104.

[Weiser & Brown, 1997]Weiser, M., & Brown, J. S. 1997. The Coming Age of Calm Technology. *Beyond Calculation: The Next Fifty Years of Computing*.

[Weiss *et al.* , 2003]Weiss, Gerhard, Rovatsos, Michael, & Nickles, Matthias. 2003. Capturing agent autonomy in roles and XML. *Pages 105–112 of: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. ACM Press.

[Wood & DeLoach, 2000]Wood, Mark F., & DeLoach, Scott. 2000. An Overview of the Multiagent Systems Engineering Methodology. *Pages 207–222 of: AOSE*.

[Wooldridge & Jennings, 1994]Wooldridge, M., & Jennings, N. R. 1994. Towards a Theory of Cooperative Problem Solving. *Pages 15–26 of: Proc. Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-94)*.

[Wooldridge & Jennings, 1995]Wooldridge, Michael, & Jennings, Nicholas R. 1995. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, **10**(2), 115–152.

[Wooldridge *et al.* , 2000]Wooldridge, Michael, Jennings, Nicholas R., & Kinny, David. 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, **3**(3), 285–312.

[Yu, 1995]Yu, E. 1995. *Modelling Strategic Relationships for Process Reengineering*. Ph.D. thesis, University of Toronto.

[Zambonelli & Parunak, 2002]Zambonelli, Franco, & Parunak, H. Van Dyke. 2002. Signs of a Revolution in Computer Science and Software Engineering. *In: Agent Oriented Software Engineering Workshop at AAMAS 2002*.

[Zambonelli *et al.* , 2003]Zambonelli, Franco, Jennings, Nicholas R., & Wooldridge, Michael. 2003. Developing multiagent systems: The Gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, **12**(3), 317–370.

List of tables

List of figures

Résumé de la Thèse: Cette thèse présente des concepts, modèles et outils pour la construction de systèmes informatiques dans le cadre des services électroniques disponibles partout et n'importe quand (Everywhere Anytime Computing, AEC). Dans cette thèse nous considérons qu'un cadre de conception et de développement dans le contexte de l'AEC doit répondre aux points suivants: gestion de l'intermittance des communications, la composition des systèmes, le respect de l'autonomie des entités, et l'interopérabilité. Pour répondre aux trois premiers points le modèle algébrique d'infrastructure nommé Mouvement, Interaction, Computation (MIC*) est proposé. L'autonomie des agents est garantie grâce à l'intégrité structurelle. Le modèle d'interaction est persistant ce qui permet de s'affranchir des intermittences du médium de communication. La composition des systèmes est réalisée par la composition des environnements de déploiement. L'interopérabilité est abordée par la coordination. Les protocoles de coordination sont représentés comme des graphes de dépendance. Nous avons présenté le lien formel ainsi que le moyen concret pour valider des séquences de ressources par rapport à un protocole de coordination donné. Finalement, nous proposons un cadre de conception des systèmes informatiques dans le contexte de l'AEC où chaque système est spécifié comme une société artificielle, peuplée d'agents autonomes, ouverte et interopérable avec d'autres sociétés en coordonnant certaines activités. Nous avons pu implémenter et tester nos approches grâce à une plate forme de simulation où un utilisateur peut naviguer à travers un monde virtuel et interagir avec différents services.

Abstract: This thesis presents concepts, models and tools for the design and development of software systems for the Everywhere Anytime Computing context. In fact, the technological context of software systems is continuously evolving and nowadays the availability of small communicating devices offers the opportunity to make the Everywhere, Anytime Computing (EAC) a reality that naturally integrates our societies and economies. Up until now there has not been an EAC solution that supports the many levels of design in a single framework. This thesis produces such a framework that deals with: (i) the constraints of the communication medium in terms of intermittence of the communications; (ii) the management of the on-the-fly composition and decomposition of the software systems; (iii) the autonomy of the software entities and systems; (iv) and the management of the interoperability of the open systems. To answer the first three points, we have introduced an algebraic model of a deployment environment holding the interacting and autonomous agents. This model is known as Movement, Interaction, Calculus* (MIC*). The autonomy of the software agents is guaranteed since MIC* preserves their structural integrity. The interaction scheme within MIC* is persistent and timely uncoupled. Consequently, the intermittent nature of the communication media does not affect drastically the functioning of the systems. Finally, thanks to the algebraic modeling, the composition of MIC* deployment environments is formally specified and concretely implemented. We have addressed the interoperability from a coordination point of view. We propose a formalism to express coordination protocols as a graphs expressing the dependencies on resources between the roles. Using this formalism, a formal link has been made, using a rewriting system. Furthermore, using Queue Petri Nets we have offered a practical mean to check the consistency between a coordination protocol and the conversations among the agents. Finally, an engineering framework is proposed for the design and implementation of applications within the AEC context as open artificial societies of autonomous agents. In order to experiment with the suggested approaches, an AEC simulation platform has been developed. The user can move within a virtual world where she/he can interact dynamically with the services that are deployed.