Universitat Autònoma de Barcelona

Facultat de Ciències
Secció d'Enginyeria Informàtica

On the Design and Construction of Agent-mediated Institutions

Memòria presentada per en
Juan Antonio Rodríguez Aguilar
per a optar al grau de
Doctor Enginyer en Informàtica

Bellaterra, Maig 2001

A mis padres y a mi hermano.

Els sotasignats, Carles Sierra i Pere Garcia, certifiquen que la present memòria ha estat realitzada sota la nostra direcció per en Juan Antonio Rodríguez Aguilar i es troba en condicions de ser defensada davant del tribunal corresponent per optar al grau de doctor en *Informàtica*.

Carles Sierra
Bellaterra, Maig 2001

Pere Garcia
Bellaterra, Maig 2001

Abstract

This thesis focuses on the specification, design and implementation of open agent organisations. We argue that open agent organisations can be effectively designed and implemented as institutionalised electronic organisations (*electronic institutions*) composed of a vast amount of heterogeneous (human and software) agents playing different roles and interacting by means of speech acts. Then we take the view that the design and development of electronic institutions must be guided by a principled methodology. For this purpose we propose a formal specification of electronic institutions that founds their design, analysis and development.

Next we present a computational model that arises from and fully captures the resulting formalisation of electronic institution. Our computational model strongly relies on and exploits the notion of mediation for founding the realisation of electronic institutions' infrastructures. Thus we propose how to fully realise an electronic institution based on *institutional agents* —the agents to which the institution delegates its services— and *interagents* —autonomous software agents devoted to mediating the interaction between each agent and the agent society in the framework of an electronic institution. Therefore electronic institutions are computationally conceived as *agent-mediated electronic institutions* since both institutional services and interactions are mediated by agents.

We illustrate the practical realisation of agent-mediated electronic institutions by describing the development of an electronic auction house inspired by the age old institution of the fish market where heterogeneous (software and human) agents may trade.

Lastly we present the evolution of the electronic auction house into a test-bed for experimenting with auction-based trading scenarios. We show how in these scenarios trading agents of arbitrary complexity participate in auctions under a collection of standardised market conditions and are evaluated according to their actual market performance. We argue that such competitive situations constitute convenient problem domains in which to study issues related with trading agent architectures in general and agent-based trading strategies in particular.

Acknowledgements

En primer lugar, mis agradecimientos más profundos son para mi familia y mis amigos, los cuales me han ofrecido en todo momento el apoyo y el empuje necesarios para completar este trabajo a pesar de mis suspensos continuos como hijo, hermano, novio y amigo. Intentaré no olvidar a ninguno de ellos.

A mis padres por dar tanto, por sacrificarse tanto, por ayudarme, aliviarme, apoyarme y sujetarme. A ellos debo todo lo que soy.

A mi hermano y a María del Mar por haber estado todo este tiempo a mi lado.

A mi tío Juan y a mis abuelos por su entusiasmo y su fe ciega en cualquier proyecto en que me implico.

A los de siempre (Vince, Javi, Flouie, Raquel, Mark, Jesús y Álvaro) por estar siempre ahí.

A la quinta del megabyte (Francisco Martín, Noyda Matos, Jesús Cerquides, Maite López, Peyman *bombmaker* Faratin y Marc Esteva) por dejarme compartir con ellos unos cafés, más de una cerveza (:-)) y unas risas y por ayudarme a soportar el último tramo de este trabajo.

Querría agradecer también a mis directores de tesis, Carles Sierra y Pere Garcia, el interés que siempre han demostrado por este trabajo y la dedicación incondicional que me han ofrecido. Especialmente, me gustaría destacar la motivación que Carles Sierra me ha transmitido durante el transcurso de esta tesis —además de múltiples valiosos consejos que me han ayudado mucho a encauzar mi anarquía—, el esfuerzo realizado por Pere Garcia y LLuís Godo por trasladarse a mi terreno y, sobre todo, la paciencia y comprensión de todos ellos. Trabajar con ellos ha sido una experiencia tremendamente enriquecedora que me ha permitido aprender el rigor que debe existir en todo trabajo científico-técnico. Sin duda alguna, sin ellos esta tesis no habría sido posible.

Querría dar las gracias a todo el personal del IIIA, tanto científico como administrativo, por la inestimable ayuda que siempre me han ofrecido.

Very special thanks to Julian Padget, Andreas Kind, Russell Bradford, Jeremy Leach and Rob, the people who hosted me during my short stage in Bath. Apart from awaking my interest for distributed computing, they let me participate in their enriching discussions and transmitted me their enthusiasm for building things. Their influence undoubtedly changed my mind about what I wanted to do for my PhD.

Destacar a Ulises Cortés, la persona que más interés y esfuerzo ha dedicado a los torneos *Fishmarket*. Sin su colaboración, el software producido durante esta tesis no habría alcanzado el nivel deseado.

Tones of thanks to Mark Klein, a colleague, a friend, and a brilliant researcher who helped me enjoy my one-year stage at MIT. Thanks also to my good amigo Robert Guttman for more than I can drop in a few lines.

Finalmente, y muy especialmente, infinitas gracias a los programadores del fantástico *Fishmarket team* (Francisco Martín, Julio García, Francisco J. Giménez, David Gutiérrez, Óscar Molina, Miguel Mateos, Santiago Macho y Juan Luís de Amaya) por haber contribuido con entusiasmo y brillantes ideas a construir diferentes partes del software que se presenta en esta tesis.

Este trabajo ha sido parcialmente financiado por los proyectos de investigación SMASH (TIC96-1038-C04-01) y CEC/HCM VIM (TMR number PL93-0186 , contract CHRX-CT93-0401) y por una beca CIRIT (FI-PG/96-8.490).

Blanes, Mayo del 2001

Juan Antonio Rodríguez Aguilar
IIIA, CSIC
jar@iiia.csic.es

"I guess one of the reasons that I've never been a very good private detective is that I spend too much time dreaming of Babylon."

(Richard Brautigan, *Dreaming of Babylon*)

"En efecto, no basta tener un buen entendimiento, sino que lo principal es aplicarlo bien. Las almas más grandes son capaces de los más grandes vicios, como también de las más grandes virtudes; y los que no caminan sino muy lentamente, si siguen siempre el camino recto, pueden adelantar mucho más que los que corren y se apartan de él."

(René Descartes, *Discurso del Método*)

Lanark looked sideways at Sludden.
"Is your life a continual feast?"
"I enjoy myself. Do you?"
"No. But I'm content."
"Why are you content with so little?"
"What else can I have?"

(Alasdair Gray, *Lanark A Life in 4 Books*)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The main goal of this thesis is to present a principled, integrated methodology for the formal specification, design, and implementation of open agent organisations. We exemplify how to put the resulting approach in practice by developing an auction-based agent-mediated computational marketplace in which heterogeneous (human and software) agents may trade. Taking one step further, we extend such computational marketplace and make it evolve in order to produce a test-bed that supports controlled experimentation to help design, build and tune trading agents in auction-based markets.

In what follows, we firstly identify the key research issues that have motivated the work presented in this thesis. Next, we describe how our research has contributed to tackle such issues and, finally, we provide the description of two case studies that will be employed throughout the rest of the thesis for illustrative purposes.

1.1 Motivation

Multi-agent systems (MAS) have emerged as a promising approach for creating agile information systems suited for addressing problems that have multiple problem-solving methods, multiple perspectives and/or multiple problem solving entities [Jennings et al., 1998]. Apart from inheriting the traditional advantages of distributed problem solving systems, MAS are characterised by their high flexibility, fundamentally based on the exploitation of an ample variety of sophisticated patterns of interaction. For instance, agents may cooperate to achieve a common goal, they may coordinate their activities in order to avoid harmful interactions or else exploit beneficial interactions, or they may negotiate an agreement. Such forms of interaction allow to consider MAS as highly flexible systems distinct from other forms of software.

Up to date most of the work produced by MAS research has focused on *closed* systems, i.e. systems in which agents and their infrastructure are designed, developed and enacted under centralised control [Klein, 2000]. Thus, MAS researchers have bargained for well-behaved agents immersed in reliable infrastructures in relatively simple domains. Such assumptions are not valid when considering *open systems* [Hewitt, 1986] whose components are unknown beforehand, can change over time and can be both human and software agents developed by different parties. In general, the design and construction of open MAS[1] pose a number of issues that must be addressed:

- *Heterogeneity.* An open MAS must be capable of accommodating heterogeneous agents, i.e. agents developed in different languages, by different parties, with different purposes (objectives) and preferences, and endowed with varying degrees of sophistication.
- *Reliability.*"Open systems must be reliable. They must be designed do that failures of individual components can be accommodated by operating components while the failed components are repaired or replaced." [Hewitt, 1986].
- *Accountability and legitimacy.* Since agents may possibly exhibit either deviating or fraudulent behaviour, their behaviours must be monitored in order detect and prevent dysfunctional behaviours which may jeopardise the overall functioning of the system. Thus only those actions that are regarded as legitimate should be permitted.
- *Societal change.* Agent societies are not static; they may evolve over time by altering, eliminating or incorporating rules. Hence the need of demanding flexible agent societies capable of accommodating future changes.

Examples of open agent systems include open electronic marketplaces and virtual supply chains, disaster recovery operations, collaborative design and international coalition military forces [Pro, 1999,Fischer and al., 1996,Rodríguez-Aguilar et al., 1997,IEEE, 1993]. Although open systems have recently started to be considered by MAS researchers as the most important application of multi-agent systems, their inherent issues have not been conveniently addressed yet. Therefore, open MAS undoubtedly appear as a class of systems worthy studying. At this point a major, capital question arises: how to approach their design and construction?

Notice that at present most agent applications lack of a principled methodology underpinning their development, and so they are produced in an ad hoc fashion. Since open MAS are highly complex, costly, critical applications it would be desirable to bargain for principled methodologies that support their specification, analysis and validation. Indeed there has been a surge of interest in agent-oriented methodologies and modelling techniques in the last few years motivated and spurred by the first generation of agent developments. Researches seem to have recently started to highlight the importance of methodologies that help mature and succeed agent research [Fisher et al., 1997,Brazier

[1]Henceforth through the rest of this chapter we employ the terms open MAS and open agent society interchangeably

et al., 1997, Sierra and Noriega, 1998, Wooldridge et al., 1999, Rodríguez-Aguilar et al., 2000, Esteva et al., 2000b]. Along these lines, we distinguish two main research areas represented by the use of formalisms and the adaption of existing software engineering and knowledge engineering techniques.

On the one hand, formalisms encompass logics and general-purpose specification languages from software engineering techniques such as Z, π-calculus, Petri Nets, CSP and others. Many advanced logics have been put forth to deal with several issues, but they all have shared the commonality of focusing on individual agents' aspects. Nonetheless, temporal logics have proven useful for dealing with multi-agent systems [Fisher and Wooldridge, 1997], though they appear to be more valuable to verify and validate their properties since there is a gap between any logic specification and an actual development. Using a general purpose-specification languages such as Z (employed in [d'Inverno et al., 1998b, d'Inverno and Luck, 1999]), π-calculus (employed in [Padget and Bradford, 1998]), or CSP represents a straightforward solution. However these languages are limited by their lack of expressiveness to cover all agent issues. For instance, Z has been reported to be inappropriate for modelling interactions, being CSP or Petri Nets more appropriate for this purpose[2]. Nevertheless Z is in turn more appropriate to model how the state of an agent changes in response to a message protocol.

On the other hand, and alternatively, many approaches extend tools and modelling techniques from software and knowledge engineering([Brazier et al., 1997]). Concerning the first case, most approaches (f.i. [Kinny et al., 1996, Burmeister, 1996]) consider existing object-oriented modelling techniques in order to either extend them, adapt them or simply directly apply them to the design of agent systems[3].

When considering the principled design of open MAS a fundamental issue is to decide whether to employ a new or an existing technique, or even combination of techniques. Our analysis of the literature indicates that generally existing techniques need to be adapted and/or combined. Alternatively, a purposely-devised technique ideally offers more expressiveness, structure and support. Unfortunately, there is no a clear answer to the issue of choosing a methodology, fundamentally because the use of principled approaches has not been common practice, and so not enough knowledge has been accumulated to elucidate this matter. Although it is widely admitted that agent applications represent a different type of application with distinguishing features that may require new formalisms and modelling techniques, no much work has been done in this direction. Furthermore, the extra features needed by the many existing techniques is still a matter of research.

Whatever the chosen technique, another fundamental aspect is to opt for either a micro (agent-centered) view or a macro (organisation-centered) view of MAS. Most existing approaches are characterized by their agent-centered per-

[2]Although Petri Nets have been recently criticized for the combinatorial size for actual protocols and the ultra-powerful simulation tools needed.

[3] [Iglesias et al., 1999] offers a survey of agent-oriented techniques.

spective (f.i. [Brazier et al., 1997, d'Inverno and Luck, 1999, Wooldridge et al., 1999]. Typically agent-centered approaches focus on closed systems; they commit to a given architecture or formal specification accounting for an agent's behaviour and attempt at explaining how an agent's inner behaviour is related to the social interactions (coordination, cooperation, negotiation, etc.) in which an agent is involved. Although early work in DAI identified the advantages of organisational structuring as one of the main issues in order to cope with the complexity inherent to designing DAI systems [Gasser et al., 1987, Pattison et al., 1987, Corkill and Lesser, 1983, Werner, 1987] —clearly influenced by related research disciplines like organisation and management theory—, DAI research, and MAS research in particular[4], have traditionally kept an individualistic character. Thus, MAS research has evolved patterned on a strong agent-centered flavour.

Criticisms against the inconveniences of employing agent-centered approaches for open agent systems have been upheld [Carley and Gasser, 1999, Esteva et al., 2000a]. In fact, notice that there is an increasing interest in incorporating organisational concepts into MAS as well as in shifting from agent-centered to organisation-centered designs [Rodríguez-Aguilar et al., 1997, Noriega, 1997a, Ferber and Gutknetch, 1998, pa So and Durfee, 1998, Sierra and Noriega, 1998, Barbuceanu et al., 1998, Dellarocas and Klein, 1999, Esteva et al., 2000b] that consider the organisation as a first-class citizen likewise the pioneering works reported in [Gasser et al., 1987, Pattison et al., 1987].

Therefore, organisation-centered approaches do not consider that an organisation design may emerge spontaneously and instead this is imposed in order to structure activities or control the actions of a system as a corporate entity. A shared organisational structure is expected to provide agents with descriptions about their roles and responsibilities in the multi-agent context and contains guidelines for their intelligent cooperation and communication. In other words, an organisational structure defines a behaviour space for agents with a whole set of norms and rules that agents are required to follow.

Evidently organisation-centered approaches are more valuable than agent-centered approaches from the point of view of open MAS designers. At this point we must formulate another fundamental question: what type of organisational structure (agents' roles, activities, relationships among activities, etc.) is more appropriate for the open MAS to be modelled?

Notice that a principled methodology will be enormously valuable for a MAS designer so as to conceive and realize structures and mechanisms of open agent systems. Complementarily there is also the need of providing support to agent architects to help them design, develop, test and tune their agents. Along this direction, MAS research has been very prolific, producing a large variety of both commercial and non-commercial agent development environments such as [Beegent, www, Agentbuilder, www, Gossip, www, Cost et al., 1998] to name a few. Such development environments provide different sets of facilities to aid agent developers to construct their agents' architectures and the sophisticated pat-

[4]In Chapter 2 we study the relationship between DAI and MAS as related but not identical research fields.

terns of interaction through which they communicate. Considering open agent systems where agents are developed and owned by different parties, this type of support is left to the agent developer's choice. Another type of support is needed when aiming at testing and tuning an agent's social behaviour, i.e. an agent's behaviour once immersed within an open system.

Let us consider electronic marketplaces as a typical example of open agent system. We observe that research in e-commerce has spurred the creation of a large number of agent-based marketplaces [5]. However no much effort has been devoted to provide support to agent developers to help them tune their trading agents' behaviour before let them play in extremely competitive trading scenarios. Apart from AuctionBot [Wurman et al., 1998] and FM [Rodríguez-Aguilar et al., 1998c] there is a paradoxical lack of test-bed for trading agents. We can generalize this observation and extend it to MAS in general, since in the last few years there has not been much activity concerning the development of agent test-beds. This is mainly motivated by the evidence that the development of DAI experimental platforms manifests as a long, tedious process characterised by the need of an enormous development effort. Nonetheless, despite the costly development inherent to test-beds, they are of tremendous value and relevance as research tools for MAS in general, and for open MAS in particular, that can help both the MAS designer and the agents' developers to cope with the high complexity inherent to open MAS.

1.2 Contributions

According to [North, 1990], human interactions are guided by *institutions*, which represent the rules of the game in a society, including any (formal or informal) form of constraint that human beings devise to shape human interaction. Thus, institutions are the framework within which human interaction takes place, definining what individuals are forbidden and permitted and under what conditions. Human organisations and individuals conform to the rules of institutions in order to receive legitimacy and support[6]. Therefore, establishing a stable structure to human interactions appears as the *raison d'être* of institutions.

We observe that indeed human societies successfully deal with similar issues to those identified above for open agent societies by deploying institutions that:

- determine what individuals are prohibited from doing and what are permitted to undertake;
- costlinessly ascertain violations and the severity of the punishment to be enacted; and
- evolve and are altered by human beings.

[5]Refer to Section 2.4 in Chapter 2 for a complete survey on agent-based electronic marketplaces

[6]For instance, the Constitution and the common law are well-known examples of institutions.

In this thesis we defend the adoption of a mimetic strategy in order to cope with the complexity of deploying open multi-agent systems. Thus we will be upholding that open multi-agent systems can be effectively designed and implemented as *institutionalized agent organizations* —henceforth referred to as *electronic institutions* or *e-institutions* for shorter. The major contribution of this thesis lies in the fact that it makes headway in the production of principled developments of electronic institutions, continuing the work presented in [Noriega, 1997b]. At this aim, we offer an integrated approach which departs from a specification formalism, proposes a wholly agent-oriented design and finally demonstrates its usefulness by implementing a proof-of-concept agent-mediated electronic marketplace. In this way we intend to illustrate how to bridge the gap between specifications and actual implementations. In what follows we decompose the major contribution highlighted above.

First of all, in Chapter 3 we introduce a novel graphical specification formalism for electronic institutions, building upon our previous work reported in [Rodríguez-Aguilar et al., 2000, Esteva et al., 2000b, Esteva et al., 2000a]. Why a formal specification language? We adhere to the following recommendation [Dav, 1993](page 215) to answer the question:

> Use a formal technique when we cannot afford to have the requirements misunderstood.

For this purpose, instead of resorting to well-known, general-purpose formal specification languages, here we have opted for proposing a new specification language founded on our formal notion of institution —so far only vaguely defined in the literature— that we introduce. Languages such as Z, CSP, Petri Nets and logics are appropriate at several stages of the design, but as argued above in order to overcome their lack of expressiveness they should be used in combination to complement one another. If so, the important issue then becomes how to define consistency constraints between each paradigm model in a single unifying framework. Alternatively we have preferred to define a single unified language that provides the expressiveness required by electronic institutions.

The proposed language allows to create graphical specifications of electronic institutions. Graphical specifications are extremely easy to understand. They are similar to the informal diagrams employed by engineers and designers while designing, constructing and analysing a system.

Furthermore, such a language focuses on the macro-level (societal) aspects of electronic institutions instead of the micro-level (internal) aspects of agents. In this way we adopt a societal view in accordance to the early works in DAI which followed a societal perspective of agent systems to ease their design and development.

Next in Chapter 4 we present a computational model that arises from and thoroughly captures the formalisation of electronic institution introduced in Chapter 3. Our computational model proposes to realise an electronic institution by means of a fully agent-oriented design based on two types of agents: *institutional agents* and *interagents*.

Firstly, institutional agents are those agents to which the institution delegates its services (f.i. registration, accountability, brokering, etc.). We detail a general, flexible computational model of institutional agent based on the notion of interpreting responsibility plans with which every institutional agent is endowed. In this way, all institutional agents are realised by means of the same computational model, only differing on their assigned responsibilities. Apart from flexibility, this model intends to provide support for scalability purposes (institutional services might be loaded into a single agent or else distributed among multiple agents).

Secondly, based on our previous work in [Martín et al., 2000b, Martín et al., 2000a] we introduce interagents as autonomous software agents, functionally acting as a particular type of facilitators devoted to mediating the interaction between each agent and the agent society. We illustrate how interagents are valuable for realising agent-based systems in general and electronic institutions in particular.

Originally interagents were conceived to mediate *conversations* among agents. Because of the proved inability of agent communication languages to capture the context of agent interactions, we adhered to the promotion of the notion of conversation as the unit of communication among agents. Here by conversation we mean a pattern of message exchange that some agents agree to follow in communicating with one another. Conversations are useful for structuring interactions among agents by organising messages into relevant contexts. In this thesis, we introduce a novel way of modeling and implementing conversation protocols as a special type of *pushdown transducer* [Roche and Schabes, 1997] which lies in between the simplest finite state machine models and complex, but highly expressive, models[7] of more difficult realisation.

The management of conversation protocols, consisting fundamentally in the validation of agents' utterances [Searle, 1969], is identified as the main task of interagents. Two major benefits, from the point of view of agent developers, are achieved by employing interagents to articulate agent interactions within an agent society: on the one hand, their agents can reason about communication at higher levels of abstraction, and on the other hand developers are released from dealing with low-level interaction issues, and so they can concentrate on the design of the agents' logics.

Further on, interagents needed to have their capabilities extended in order to handle not only conversation protocols but also institutions' specifications. Interagents constitute the sole and exclusive means through which agents interact with the rest of agents within the institution. They are all owned and set up by the institution in order to validate, enforce and monitor agents' social behaviour. Notice that interagents are a key element to realise open electronic institutions by offering a common interface to electronic institutions to all exogenous agents that encapsulates the institutional rules.

We introduce the notion of *agent-mediated electronic institution* so as to

[7]For instance, models based on Dooley graphs [Parunak, 1996] or Colured Petri Nets [Cost et al., 1999].

refer to an electronic institution whose infrastructure is fully realised by means of institutional agents and interagents. In other words, institutional agents and interagents are responsible for the computational realisation of the rules, the normative environment of institutions.

Considering also exogenous agents, we thus conceive a multi-layer logical architecture of agent-mediated electronic institutions composed by a kernel (composed of institutional agents), a middle-ware layer (composed of interagents) and an external layer (inhabited by exogenous agents).

In order to exemplify how to realise actual agent-mediated electronic institutions based on the computational model introduced in Chapter 4, in Chapter 5 we present a *FM96.5* [Rodríguez-Aguilar et al., 1997], an open electronic auction house where heterogeneous (software and human) agents may trade. To the best of our knowledge, along with *AuctionBot* [Wurman et al., 1998] FM96.5 was the first auction house providing support to software agents: it encloses libraries of agent templates in several languages and offers the possibility of generating agents with customised trading strategies.

Considering auction protocols, FM96.5 includes, as a distinguishing feature from the rest of auction houses for human and software agents, a sound and fair implementation of a real-time Dutch auction protocol. Although as a matter of fact originally it only included Dutch auctions, FM96.5 had its capabilities extended in order to accommodate other classic auction types, namely English, First-price sealed-bid and Vickrey.

One capital aspect which does not seem to be conveniently considered by most agent marketplaces, apart from *Zeus* [Collins and Lee, 1998], is the notion of accountability of the market activity. FM96.5 bargains for monitoring facilities aimed at keeping track of all events taking place in the market as well as at integrating into a coherent whole the vast amount of information generated by individual agents in order to picture the behaviour of the system in such a manner that humans can better understand it and market makers can better analyze and debug it.

But undoubtedly the two more remarkable, differentiating features of FM96.5 with respect to the rest of computational marketplaces are its underlying flexible, principled design and its organisational structure, deriving from our computational model and our formal notion of institution.

Firstly, its highly flexible, purely agent-oriented design has proven to be enormously valuable in the evolution of the development in order to accommodate changes in the market's structure and activities (upgrades and changes of protocols, inclusion of new protocols, etc.). Secondly, its organisational structure has added the modularity and abstraction needed to cope with the complexity inherent to the development of large multi-agent systems.

Although in this thesis we only present FM96.5 as a proof-of-concept of our methodology based on the formal specification language of Chapter 3 and the computational model of Chapter 4, the agent-mediated electronic institution reported in [Plaza et al., 1999] is another proof of the usefulness of our approach.

If a trading agent intends to behave aptly in the context of an auction marketplace such as FM96.5, the agent's decision-making process may be quite elaborate. It could involve procedural information—when to bid, how to withdraw—, reasoning about individual needs and goals, information and reasoning about supply and demand factors—which may involve other agents' needs and goals—and assessment of its own and rivals' performance expectations—which in turn may require knowledge or reasoning about the external conditions that might affect the auction. Evidently, many approaches can be taken to deal with this decision-making process. From highly analytical game-theoretic ones, to mostly heuristic ones. From very simple reactive traders, to deliberative agents of great plasticity.

There is the intricate matter of providing agent developers (and agent users) with some support to help them face the arduous task of designing, building, and tuning their trading agents, before letting them loose in wildly competitive markets. We have attempted to make headway in this direction by developing the *FM* test-bed [Rodríguez-Aguilar et al., 1998b, Rodríguez-Aguilar et al., 1998c], described in Chapter 6, for experimenting with auction-based trading scenarios that can be used to test and tune trading agents. Conceived and implemented as an extension of FM96.5, FM allows to generate auction-based scenarios in which trading (buyer and seller) heterogeneous (human and software) agents of arbitrary complexity participate in auctions under a collection of standardised market conditions and are evaluated according to their actual market performance.

To the best of our knowledge, AuctionBot [Wurman et al., 1998] and FM are the only test-beds for trading agents in auction-based markets. Apart from the originality of the test-bed domain, there are several features that differentiate it from the rest of test-beds. Firstly, FM is open, likewise the marketplace itself and unlike most test-beds which are closed systems, and so agents owned by different parties can take part in a same experiment. Secondly, FM allows to handle eventual scalability problems by allowing experimenters to distribute their agents across a network, and allowing remote agents to participate in auction tournaments. Lastly, FM allows to artificially generate a wide range of trading scenarios: from simple toy scenarios to the real-world scenarios represented by the market scenarios generated by FM96.5.

Notice that differently to the vast majority of agent research devoted to computational marketplaces —particularly concerned with supporting trading agents' construction— we also provide the necessary support to investigate and analyze trading agents' behaviour. At the time of writing, the different versions of FM [Rodríguez-Aguilar et al., 1998a, Rodríguez-Aguilar et al., 2001] have demonstrated to be valuable in hosting several auction-based competitions [Fishmarket, www].

One of the distinguishing features of this dissertation is its interdisciplinary scope, encompassing driving ideas and techniques from several research areas as illustrated in Figure 1.1. But notice that this is commonplace to agent research, an intellectual area whose excitement mainly stems from the challenge of integrating knowledge [Briot, 1998].

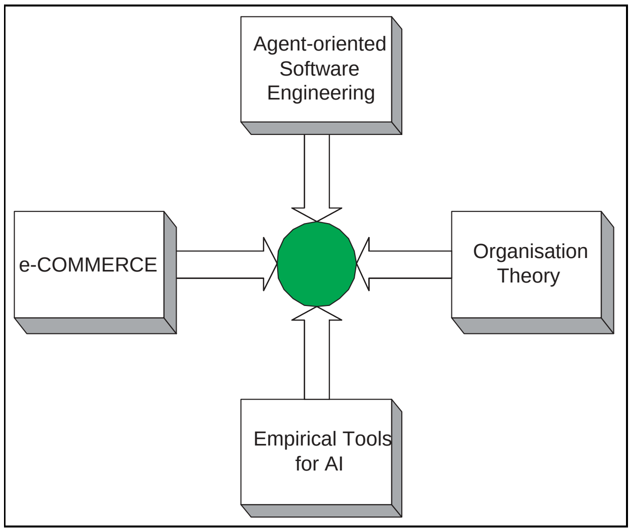


Figure 1.1: Interdisciplinary scope of this thesis.

1.3 Case Studies

In this section we describe in detail the two case studies that will be employed for motivating, upholding and illustrating the several notions that we are to introduce during the rest of the dissertation. They constitute two practical cases of institutions that in fact have served as the starting point and inspiration of two different projects, *Fishmarket* [Fishmarket, www] and *Comris* [de Velde, 1997], which pursued the practical design and construction of agent-mediated electronic institutions (denoted as *AmEI*henceforth), though for different purposes. Thus, while the *AmEIs* constructed in the framework of the *fish market* project were devoted to auction-based e-commerce, the *AmEIs* developed by *Comris* were set up for supporting a conference centre.

Above we identified as our main goal the construction of infrastructures for computational institutions. But then, the proposed infrastructures highly depend on our current conception of electronic institutions. Since we advocate for adopting a mimetic strategy when conceiving computational counterparts of human institutions, it is apparent the need for a careful analysis of institutions that help us identify all the elements needed to be considered on the computational side.

Next, a thorough description of the two cases follows.

1.3.1 The Fish Market

As a starting point for the study of institutions in the framework of the *Fishmarket* project, we chose auction houses –and the fish market in particular– as a paradigm of traditional human institutions. The description that follows intends to precisely describe how the fish market works. We firstly describe the elements surrounding the market place, and next we proceed to the identification of the activities taking place during and around the auctions and the roles adopted by the participants for each activity. Mention that the description that follows has been summarised and adapted from [Noriega, 1997b], and so we suggest the interested reader to refer to that work for a much more thorough description.

Market Place Context

So far we have referred to the fish market in a rather general sense. In order to focus our study we chose a particular, though general enough, sample of fish market: the fish market in Blanes, a fishing town in the Costa Brava, in Girona. Blanes hosts a fishing fleet whose catch is sold twice a day in the local market place, managed by the local fishermen's guild under a lease from the government.

Fishermen obtain their revenues from the daily auctioning of their catches in the fish market using the traditional *Dutch*[8] auction that we fully describe further on. However fishermen are charged with some transaction costs deriving from the auctioning: 2% commission and consumables. The total, final revenue is divided into the owner of the boat and its crew: half of it goes to the owner, and the other half is shared out among the crew (the captain gets two parts, pilots and machinists get a part and a half, and everyone else gets one part).

A distinguishing feature of the fish market in Blanes is that it is one of the first fish markets in Spain that have effectively automated part of the auctioning process. As a major benefit, buyers are enabled to freely wander around inspecting the incoming goods instead of being confined to a hall, such as in a classic Dutch auction hall. Besides, the system employs a database which registers all the accounting information deriving from the auctioning: purchases, payments, credits, buyers, sellers, etc.

The fish market is physically located in a building composed of a large market hall, an incoming wharf, a delivery wing, offices, and warehouse such as depicted in Figure 1.2. Auctions take place in the auction hall where goods are delivered by fishermen, classified by market staff members, and subsequently sold. Accounting activities of the auctioning process are in charge of the accounting staff, situated in the offices.

Participants and Activities

We distinguish three types of participants: *sellers*, *buyers* and *market intermediaries* (market employees). Next we explain away the activities in which they get involved within the market place.

(i) Sellers. A seller is a boat, or the fisherman or fishermen who own or operate some boat registered in Blanes.

[8]This protocol is called a *Dutch auction* because it is the way flowers have been traditionally traded in Holland.

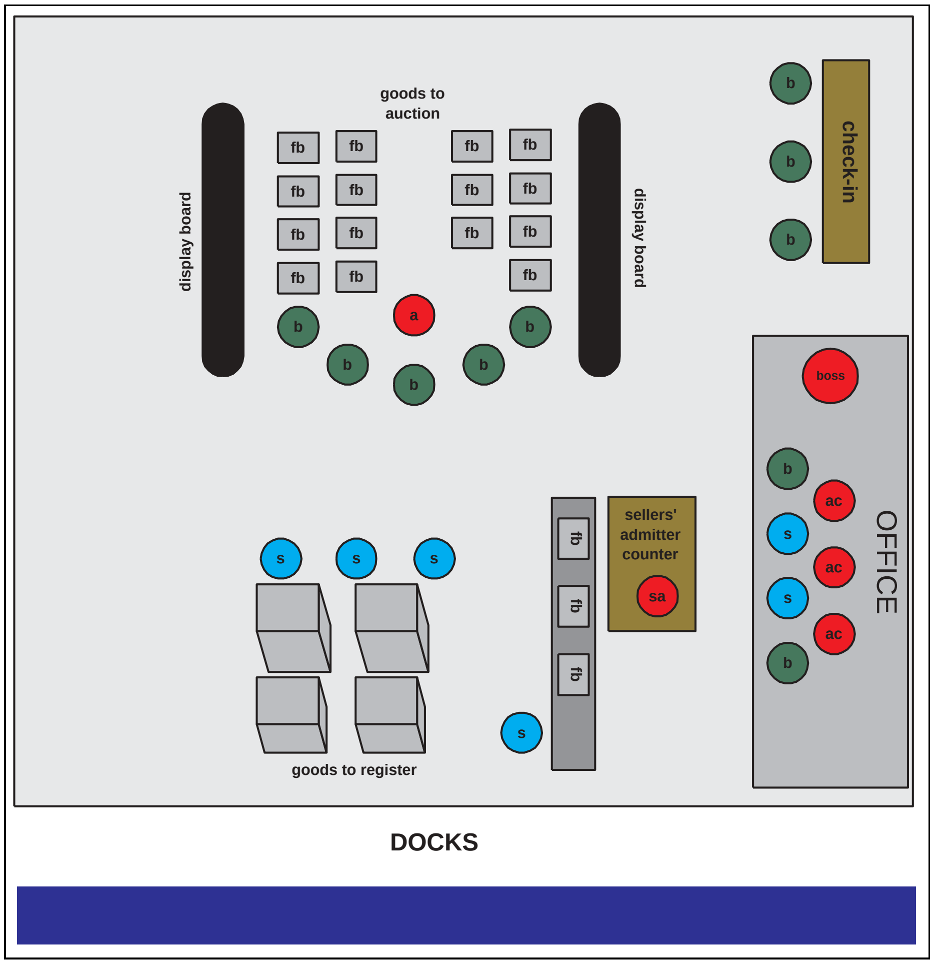


Figure 1.2: Fish Market Floor Plan. Squares labelled with *fb* stand for fish boxes, while circles represent the market's participants, both market intermediaries (a = auctioneer, sa = sellers' admitter, ac = accountant, boss) and traders (b = buyer, s = seller)

Sellers get their catches ready for sale while fishing or sailing back to port by arranging produce of the same type (same species and uniform quality) in boxes. Once reached the docks, the cargo is unloaded and the fish boxes sorted for delivery. All boxes of the same type are grouped together to compose a lot.

Within the market place, sellers' basic activity is limited to have their goods registered for sale and getting paid once these get sold by auction. As to the registration of goods, the boxes composing each lot are delivered to one of the market intermediaries, the *sellers' admitter*, who is in charge of weighting, classifying and tagging them before conveying them to the *auctioneer* for their auctioning. The institution keeps track of all transactions occurring during the registration process, and also manages the individual accounts corresponding to each seller. Typically sellers receive the revenues deriving from the sale of their goods every two weeks.

From a strategic point of view, sellers cannot do much about improving the selling prices obtained by their goods. However, sellers can eventually benefit from the dramatic price fluctuations that may result during a market session. In general, such price fluctuations will directly depend on the uncertainty of the supply[9] and the acknowledged prestige of the boat —a large bias in price is awarded to the prestige of a boat.

(ii) Buyers. We consider a buyer as an individual, acting on his own behalf or on behalf of some companies, that has legally established a credit line with the institution. Once set up the credit line, the buyer has an account open to keep track of all his transactions, becomes enabled to buy fish, and receives an electronic device —the so-called *mineing device*[10]— to be employed for bidding during the auctioning. Buyers keep their *mineing* devices locked up in special compartments (labelled as *check-in* area in Figure 1.2) in the market hall and remove them when intending to participate in the auctioning of goods.

After analysing the buyers' behaviour within the market place, we can differentiate three rather distinct types of buyers for which purchasing strategies, goals and credit requirements are largely different:

- **Wholesale buyers** acquire large amounts of cheaper fish for packing, distribution, freezing and industrialisation. Since they play on thinner margins and higher volume, they concentrate in morning auctions when there is less variety in the catches, or when a particularly large catch is brought in. They have to set up a credit line of 1.5 million pesetas for participating in the auctioning of goods.

[9]Fishing is quite an inexact science, and so the final catch of a fishing trip is quite unpredictable: in a single day different boats may obtain quite different catches despite fishing the same waters.

[10]In general, bids are expressed *calls* made by a bidder to an auctioneer in the shape of either a declaration of a buyer of his intention to purchase a good at a certain price, or the acceptance *mineing* made by the bidder of a *price quote* for a good offered for sale by the auctioneer.

- **Fishmongers** own fish shops in town or peddle fish in neighbouring towns. Buyers of this type need to keep a regular stock, but highly conditioned by the market sector that they address. In general, retailers are very sensitive to competition among them since they are not happy with the idea of others (perhaps the shop next door) having some produce that they do not have. Their credit lines range from 300,000 to 500,000 pesetas.

- **Restaurant owners.** They are concerned about obtaining high quality products to stock their regular menus. Though evidently they take advantage of abundant catches or unusual species, they seem to be rather independent from competitors. Their credit lines are above 500,000 pesetas.

Buyers are allowed to enter the market hall at any time while open, and are also allowed to freely move around, leave or come back in. A buyer's basic activity within the market restricts to inspecting incoming cargoes to be set at auction, to control other buyers' (competitors) presence and activity, and most importantly to acquire the goods that they require to satisfy their needs. To succeed in acquiring goods buyers have to push the button of their *mineing* devices at the right time, before anyone else do it, during the auctioning of the goods that they are interested in. This action is equivalent to raising a hand or saying *mine* to the lot at auction in the equivalent traditional, non-automated auction. The buyer who first pushes the button is entitled to take as many boxes as wished whenever his current credit supports the total amount of the operation.

Once sold a lot, its cost is immediately charged to the purchaser's account. Then, the lot must be taken away by the purchaser himself since the institution is responsible neither for warehousing fish, nor for delivery.

As buyers may possibly run out of credit during a market session, they are allowed to update their accounts or negotiate extensions to their credit lines at any time during a session. Typically, however, buyers settle their accounts once a week after a market session. Occasionally, in case that a buyer exceeds his credit when submitting a bid, this is declared *invalid* and the buyer's *mineing* device is automatically disabled and it is not re-enabled unless the credit line is reactivated by means of either a negotiation or payment process.

Notice that although a buyer might guess the financial power of his competitors, he does not have access to the actual figures, which are privately preserved between the market and the buyers.

(iii) Market Intermediaries. The institution is responsible for providing a *sellers' admitter* to classify, weigh and tag incoming fish boxes, *accounting staff members* who care about registering buyers and sellers and maintaining their accounts, a trained *auctioneer* to run the auction process, and support personnel to distribute fish boxes around the market hall. Complementarily, a market *boss* acts as the final authority during auctions.

Figure 1.3: The sellers' admitter registering goods.

Figure 1.4: Sellers delivering goods.

The **sellers' admitter** is in charge of the good registration process, i.e. of tagging incoming fish boxes, and storing the supply information onto the market database. The goods' registration process is undertaken for each boat separately when delivering its cargo. The sequence in which sellers are serviced by the sellers' admitter is defined by the strict order of arrival of their corresponding boats. When being serviced, the seller must decide the order in which lots and boxes must be processed by the sellers' admitter (see Figure 1.4). Then, the admitter proceeds to tag the fish (see Figure 1.3) by determining:

- the *type* of good, which in turn depends on the species in the box and its quality
- the *weight* of the box; and
- the *position* of the box and lot in the auction set.

Although the weighing and positioning of boxes are fully automated, their classification is entirely left to the sellers' admitter who has the authority to either split or join lots, or even reassigning the lot's starting unit price (in pesetas per kilogram) which is usually set 20% above the last selling price for that good type in a previous market day. No reservation price is set for any item.

The **auctioneer** (see Figure 1.5) is responsible for the management of bidding rounds. He identifies the lot to be sold, validates it —he has the authority to carry out as many changes as deemed appropriate over the lot in order to validate it— and starts the bidding till a single buyer bids for it. In such a case he adjudicates the lot to the bidder.

The lot's information is displayed on two large electronic boards. Figure 1.6 shows the structure of the electronic boards identifying the fields that compose them, and Table 1.1 details its actual content. Such information can be altered by the auctioneer using a *remote control* device. Furthermore, the auctioneer is also equipped with a microphone that permits him to vocally override the information displayed on the boards and conduct a bidding round by voice.

How does a *bidding round* work? A bidding round starts when a lot is presented for auction, and ends when it is adjudicated. First of all, the lot is presented by the auctioneer and its information is displayed on the display boards. When the auctioneer opens (starts) the round, the starting price located at the bidding clock area (Field 6) of the display board starts descending at a swift, but uniform, pace till some bids, mine commands, are received by the system from the buyers' *mineing* devices (see Figure 1.7). The fish market automated system detects and subsequently handles the bids submitted by buyers through their *mineing* devices. These emit infrared signals in different (coded) frequencies that the system can detect thanks to sensors distributed over in the auction room that read the signals.

A mine command fixes the clock at that instant. But then two major situations may arise:

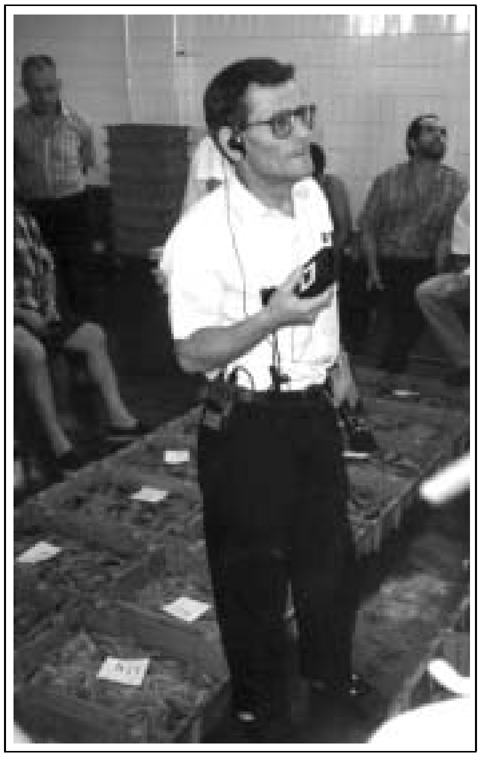


Figure 1.5: The Auctioneer.

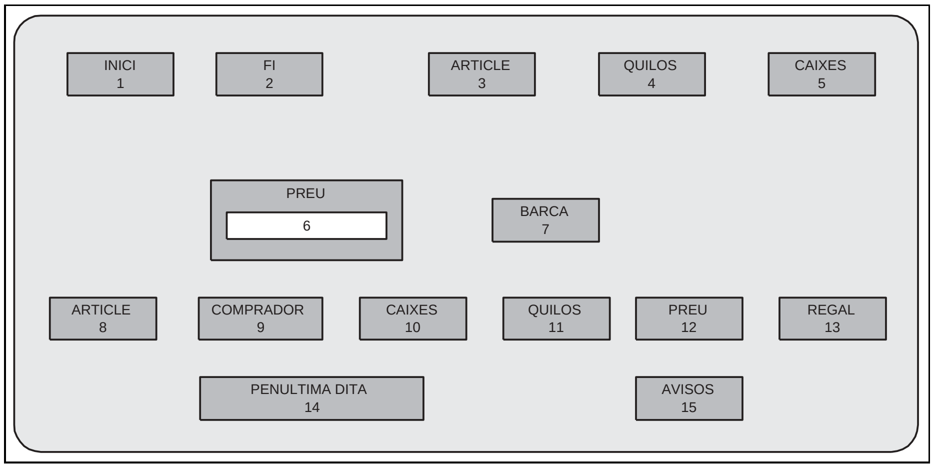


Figure 1.6: A display board's structure.

Field	Label	Content
1	INICI	The number of the first box in the lot to be auctioned
2	FI	The number of the last box in the lot to be auctioned
3	ARTICLE	Type of good
4	QUILOS	Weight
5	CAIXES	Number of available goods
6	PREU	Bidding clock. Displays the current unit-price quote
7	BARC	Name of the boat where the lot comes from
8	ARTICLE	Type of good
9	COMPRADOR	Buyer's name
10	CAIXES	Buyer's quantity option (boxes taken)
11	QUILOS	Total weight of these boxes
12	PREU	Unit price paid
13	REGAL	Not in use
14	PENULTIMA DITA	The last available final bid for the same type of good
15	AVISOS	Collision or invalid sales

Table 1.1: Auction room display boards: field content.

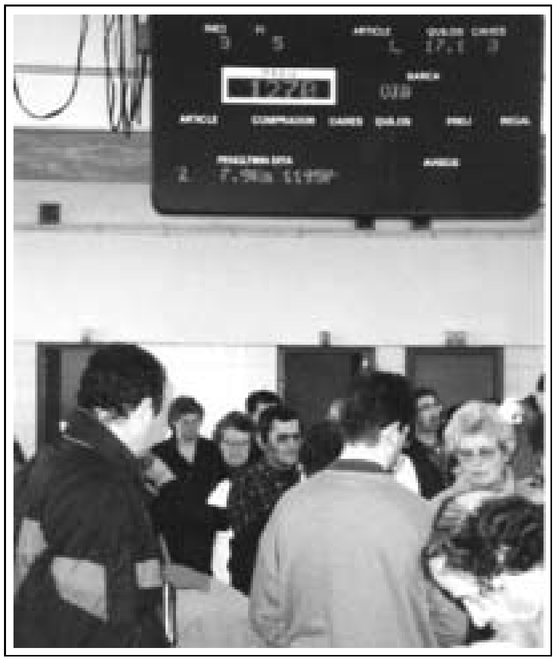


Figure 1.7: Bidding round.

- *Single valid bid.* The good is adjudicated to the bidder, whose identity is revealed at Field 9 on the display boards[11]. Then the bidder chooses a subset of boxes of the lot at the selling price. That depends on how long the buyer kept the button of his *mineing* device pressed. If the pulse was long, the signal is considered as the willingness to take the whole lot. Otherwise, the buyer is considered to be opting for a quantity option. In this second case, the remaining boxes are re-auctioned setting a starting price 1.2 times the sale price.

- *Collision, invalid sale or cancellation.* The round is repeated over setting a starting price price 1.2 times the price showed at the bidding clock area (the price at which the round was stopped).

  - *Collision.* Two or more buyers pushed their mineing devices at exactly the same time.
  - *Invalid sale.* Some buyer pushed the button before starting the bidding clock.
  - *Cancellation.* The auctioneer over-rid the round, either because a lot is to be split, several lots are joined into one, or a buyer asked for an exceptional treatment (involuntary or erroneous bid, malfunction of the *mineing* device, or return of an adjudicated lot).

  Whatever the case, the occurring event is conveniently displayed at Field 15 on the display boards.

During the bidding round the auctioneer is permanently in touch with the market boss and the accounting staff. Thus the boss can communicate special directions (close the market, change the order of auctioning, raise the starting price of a good) or validate the auctioneer's decisions.

The auctioneer can somewhat control the rhythm of auctioning. Time step (between quotes) is either automatically set by the system or fixed by the auctioneer when running voice auctions. Latency periods between lots are controlled by the auctioneer subject to eventual delays provoked by the waiting for a boat's load to be admitted and displayed. In general, the auctioneer tries to adequate the auction rhythm to the buyers' state of interest and attention.

Historical information on catches and daily market session prices, as well as weekly averages, is available form the market in printed form.

Finally, the **boss** supervises market sessions. In general, he is responsible for ordering the auctioneer when to open the session to auction the first lot and for deciding with the auctioneer the closing of the session. Nonetheless, and occasionally —f.i. because of a storm, the lack of catch, or the suspicion about collusion among buyers—, he is authorised to either delay or suspend the market session. Furthermore, he is also authorised to override the admitters' and the auctioneer's decisions if necessary.

[11]Notice that the bidding round is *private*, and so the identity of a buyer is only revealed when he is the highest-bidder.

Analysis

We analyse the description above so as to start identifying and advancing the core notions of institutions that shall serve throughout the dissertation as the foundations to achieve the goals presented in Section 1.2.

Our first observation has to do with the *role* of the agents participating in the market. We fundamentally distinguish two classes of roles: *institutional*, adopted by market intermediaries (employees), and *non-institutional*, adopted by traders (buyers and sellers). Market employees are responsible for guaranteeing the sound operation of the several activities conducted in the market. In other words, the functioning of the institution is delegated to the market employees, and so we can think of them as the *agents*[12] of (working for) the institution. Analogously we could also think of traders as agents. Thus, in general each institutional agent becomes either partially or totally in charge of some activity or activities taking place within the market place.

Each activity is governed by the rules of a protocol established by the institution itself. Such rules explicitly state the legal, allowed behaviour for each role participating in the activity, and so they do affect both institutional and non-institutional agents (market employees and traders). Moreover, apart from specifying the legal behaviour for each role, the rules also make explicit how to deal with deviating behaviours. For instance, the *mineing* device of a buyer is disabled after submitting bids that cannot be supported by his current credit. In addition to establishing which roles can participate in the activity, the institution restricts the number of roles involved in each activity and the conflicts possibly existing among the roles. For instance, the institution requires a single auctioneer for the auctioning of goods, whereas multiple buyers might participate up to a limit of one hundred whenever none of them is the auctioneer himself.

We also observe that there exist relationships among activities. While some activities like the admission of buyers and sellers are completely independent, others are tightly related. For instance, consider the case of a buyer: he cannot make any payment if he has not bought anything by auction, and in turn he cannot bid (and subsequently buy) any good unless he has previously and successfully set up his own credit line. From this follows that we can see the fish market as a place where multiple related and independent activities take place concurrently.

And finally, some words about the *commitments* adopted by the institution and the traders. We observe that some actions in the context of a given activity may imply the adoption of a future commitment (f.i. when accepting and tagging goods the institution is subscribing the commitment to put them at auction, or after winning a bidding round the bidder is committed to subsequently pay for the acquired good). It seems then that some rules, norms, of the institution cannot be locally associated to a particular activity because they may affect

[12]Here we employ the term *agent* to mean a person who acts on behalf on another when doing a particular task. Further on we will be employing the same term to mean a particular type of programs.

several activities.

It is our view that any formal specification or computational model of an institution must successfully capture the elements and structures identified throughout the analysis above.

1.3.2 A Conference Centre Environment

Most large conferences are organised around a large variety of events. In order to take part in them, participants must previously complete a registration process. Once completed the registration, a participant can take part in some of the following events in different manners:

Main conference talks. These are the central activity held during a conference. Before the conference, researchers submit their papers to the programme committee requesting for giving a presentation during the conference. Only papers that are accepted for presentation are scheduled time for a speaker's talk at the conference venue. All registered participants are allowed to attend conference talks.

Usually there are several session tracks being held in parallel, and so participants must eventually opt for one out of the simultaneously ongoing talks.

Invited talks. Talks given by experts in some field of interest. All registered participants are allowed to attend these talks.

Tutorials. They are offered by experts in some field of interest. Differently to invited talks, a tutorial does not consist of a short speech, but of a 1-day, mid-day lesson. Participants must additionally register for those tutorials that they aim at attending. Notice that several tutorials may be occurring simultaneously.

Exhibitions (demos). Researchers and industries are usually invited to show the results of their investigation during a special exhibition session. Likewise tutorials and main conference talks, exhibitions are usually run in parallel.

Workshops. These are more specialised mini-conferences held in parallel to the main conference. Their scope is not as broader as the main conference, and so they are oriented to small groups of researchers working on the same or highly similar topics.

In general, tutorials, workshops and main conference talks are celebrated on different dates.

Apart from the official events above, one of the most important participants' activities consists in meeting other people sharing research interests. Notice that this purpose can turn out to be rather difficult in a large conference environment and furthermore there is the matter of scheduling suitable meeting times for the parties involved in the meeting.

Analysis

Next, we analyse the description above with the purpose of conceptualising a conference centre environment by identifying its core components, similarly to the process followed when analysing the fish market.

First, we distinguish several roles for the participants in a conference event that again we divide into two classes: *institutional*, adopted by conference organisers, and *non-institutional*, adopted by conference participants. In the first class we find the *chair* of the conference, the *programme committee* members, along with the *administrative* personnel. Within the second class we find *speakers*, *invited speakers*, *tutorial speaker*, *workshop participant*, *tutorial participant*, *demonstrator*, and *participant*. Notice that some participants may eventually play several roles. For instance, an invited speaker might be also a tutorial speaker or simply a demonstrator. Notice also that from the point of view of a conference attendant perhaps one of the most interesting aspects of attending a conference is to meet other people with similar research interests.

Here we also identify several activities, each governed by the rules of a protocol established by the conference organisation. For instance, concerning the main conference talks, the speaker is presented by the chair of the session to all attendants prior to his speech. After the talk, attendants are invited to ask questions when authorised by the chair. Observe the several roles taking part in this simple scene and their interaction.

Furthermore, there also exist relationships among activities. Some activities are completely independent while others may be causally related. For instance, all participants are forced to registering before taking part in any activity at the conference centre.

Concerning the commitments adopted by the institution and the participants, analogously to fish market we observe that some actions in the context of a given activity may imply the adoption of a future commitment. For instance, speakers are obliged to give their talks at the scheduled time and exhibitors to run their demos when required. And so, the need for norms that regulate the participants' commitments is also required for the conference centre.

1.4 Structure

This thesis is organised in seven chapters, including this introduction chapter, and one Appendix:

Chapter 2 reviews the main research relevant to our thesis and discusses their contributions and limitations. We present the current research on methodologies for designing agent-based systems; research in agent interaction, focusing on those approaches concerned with modelling agent conversations; the MAS approaches to computational organisation theory and applications; and finally, we describe the features of the test-beds developed for agent research.

Chapter 3 proposes that open agent organisations can be effectively designed and implemented as institutionalised electronic organisations (*electronic institutions*) composed of a vast amount of heterogeneous (human and software) agents playing different roles and interacting by means of speech acts. We take the view that the design and development of electronic institutions must be guided by a principled methodology. Along this direction, we advocate for the presence of an underlying formal method that underpins the use of structured design techniques and formal analysis, facilitating development, composition and reuse. For this purpose we propose a formal specification of electronic institutions that founds their design, analysis and development.

Chapter 4 presents a computational model that arises from and fully captures the formalisation of electronic institution provided in Chapter 3. First we argue on the capital importance of mediation for founding the realisation of infrastructures for agent-based systems in general and for electronic institutions in particular. Subsequently we introduce a particular type of facilitators, the so-called *interagents*, as autonomous software agents devoted to mediating the interaction between each agent and the agent society in agent-based systems. Then we discuss how to extend this basic interagent model so that interagents can be also employed in the framework of an electronic institution. Furthermore, we also detail a computational model of *institutional agents*, i.e. the agents to which the institution delegates its services. Lastly, we present how to fully realise an electronic institution based on interagents and institutional agents.

Chapter 5 thoroughly describes the development of a practical agent-mediated electronic institution. Concretely we present the computational counterpart of the fish market institution presented as a case study in Chapter 1. Thus, we detail the design and implementation of an agent-mediated electronic auction house inspired by the age old institution of the fish market, where heterogeneous (software and human) agents may trade. Such an exercise serves us to illustrate how to put in practice the concepts introduced in former chapters.

Chapter 6 presents a framework for experimenting with auction-based trading scenarios. In these scenarios, trading (buyer and seller) heterogeneous (human and software) agents of arbitrary complexity participate in auctions under a collection of standardised market conditions and are evaluated according to their actual market performance. The proposed framework, conceived and implemented as an extension of the agent-mediated electronic auction house presented in Chapter 5, constitutes a test-bed for trading agents in auction tournament environments. We argue that auction-based tournaments constitute convenient problem domains in which to study issues related with trading agent architectures in general and agent-based trading strategies in particular. In order to illustrate the practical usage of the resulting framework, we comment on the experiences acquired from

Chapter 2

Background and Related Research

In this chapter we summarise and analyse the research appearing in the literature relevant to this thesis. We present this research along four different lines, corresponding to the issues that we face in this work. Thus, Section 2.2 discusses the role of organisational design in DAI systems. Next, in Section 2.3 we review the state-of-the-art of the research in inter-agent communication, focusing on those approaches concerned with modelling agent conversations. In Section 2.4 we present and discuss the features of the most salient computational markets. The chapter finishes describing in Section 2.5 the features of the test-beds developed for agent research.

But first of all, we believe that is it worth dedicating part of this chapter to define our terminology. This is particularly important to obtain a concrete definition of the fundamental terms of distributed artificial intelligence research so that we can conveniently situate our research in the right place. We regard this effort as necessary due to the overuse and abuse of such terms made by current, frantic agent research.

2.1 Distributed Artificial Intelligence. Basic Concepts

From its inception, the field of *distributed computing* became mainly concerned with the synchronisation and parallelisation of different processors sharing the data associated with a problem. Later on, AI researchers started adding intelligence to the field by proposing to share not only data but also control among processors. This research originated a new field, *distributed artificial intelligence* (DAI), focusing on problem solving, communication, and coordination differently to the lower level issues of parallelisation and synchronisation [Lesser, 1995]. As noted in [Weiss and Send, 1996], DAI systems do not necessarily involve agents, and they can simply consist of several interacting entities with certain degrees of autonomy and intelligence which are logically and often spatially distributed. However, when DAI systems are conceptualised in terms of agents, we refer to them as *agent-based systems.*

DAI was decomposed into Parallel AI and Distributed Expert Systems, which in turn was decomposed into Distributed Knowledge Sources and Distributed Problem Solving (DPS) [Decker, 1987]. But nowadays DAI systems tend to be classified as either DPS systems or multi-agent systems (MAS).

The term multi-agent system —as well as agent— is currently in vogue, but its overuse and abuse tends to generally apply it to any system composed of multiple interacting agents. As it has become progressively used, it seems to be encompassing a larger variety of issues that difficult its distinction from DAI.

In this work we consider MAS as a sub-field of DAI research. And then, what does distinguish DPS from MAS?

On the one hand, given a problem to solve, DPS concentrates on how to get multiple agents to work together to solve it in a coherent, robust, and efficient manner under varying (dynamic and uncontrollable) environmental conditions. DPS is mainly concerned with *information management*, consisting of *task decomposition* (how to divide a complex task into manageable subtasks to be adopted by different agents), and *solution synthesis* by combining the results of the different subtasks.

In DPS systems there are strong assumptions about the compatibility of different agents and their *benevolence* (DPS assumes that agents agree, share tasks, communicate truthfully, and so on). The benevolence assumption became unrealistic when modelling the social systems upon which much of DPS research was built. Thus, experiences in the social sciences demonstrated that achieving benevolence in a collection of individuals is far from simple.

At that point a new metaphor was needed that did not assume agents' benevolence. MAS research departs from borrowing from the social since literature the assumption that agents are rational, and so they act motivated by maximising their own benefits. MAS research focuses on getting agents with individual preferences interact in particular environments so that each agent acts in a way that leads to desired global properties. Thus, MAS research intends to answer how, given a particular environment, can certain collective properties be realised if the properties of agents can vary uncontrollably. For instance, the work in [Rosenschein and Genesereth, 1985] studies how self-interested agents can yet converge on agreement about deals so that they can mutually benefit.

Differently to DPS, MAS deals with *behaviour management.* A distinguishing feature of MAS is that typically the individual behaviours of agents are less complex than the interactions among agents. In fact the interaction issues among agents form the basis for MAS research. Therefore MAS exploit sophisticated patterns of interaction. Examples of common types of interactions include [Jennings et al., 1998]: cooperation (working together towards a common aim); coordination (organising problem solving activity so that harmful interactions are avoided or beneficial interactions are exploited); and negotiation (coming to an agreement acceptable to all involved parties). The flexibility and high-level nature of these interactions distinguishes multi-agent systems from other forms of software.

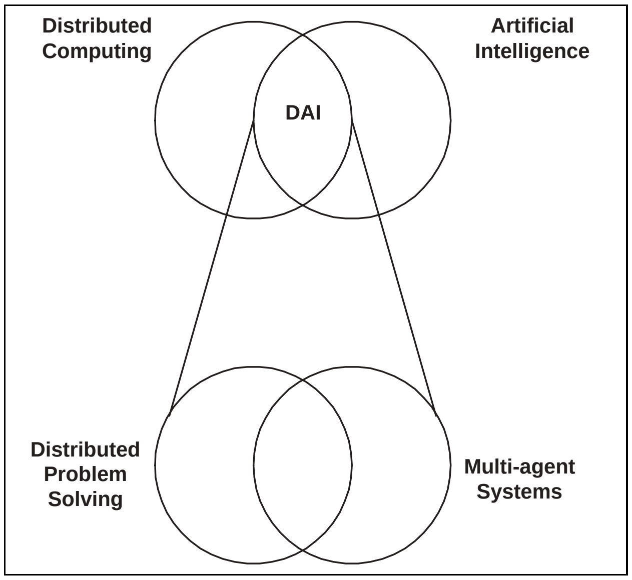


Figure 2.1: A taxonomy of fields containing MAS [Stone and Veloso, 1996].

Figure 2.1 depicts the relationships between distributed computing, artificial intelligence, DAI, DPS and MAS.

At this point, a major question arises: in what circumstances are MAS useful? Some domains do require a MAS approach. Particularly, this is true when there are different people or organisations with different (possibly conflicting) goals and proprietary information whose interactions need to be handled.

A fundamental dimension to consider in multi-agent systems is *openness*. *Open systems* [Hewitt, 1986] are those in which the structure of the system itself is capable of dynamically changing. An open system's components are unknown in advance, it can change over time, and it can be composed of heterogeneous agents implemented by different people with different software tools and techniques. The best-known example of a highly open software environment is the Internet. Internet can be regarded as a large, distributed information resource, with nodes of the network designed and implemented by different organisations and individuals. And, for instance, marketplaces over the Internet constitute a typical example of *open multi-agent systems*. Though open systems represent arguably the most important application area of multi-agent systems [Wooldridge et al., 1999] —particularly if they are intended to be running over the Internet— they have not been conveniently addressed by MAS research as discussed further on in the following section. In this thesis we have intended to contribute to the understanding of these type of systems, and so we have chosen open multi-agent systems as our subject of study.

To close this section, notice that so far not much has been said about what we mean by agent. Defining what an agent is has become the subject of an endless, heated debate held by agent researchers (see [Franklin and Grasser, 1996] for a sample of contributions in this direction). Here we adhere to the definition presented in [Jennings et al., 1998] that considers an agent as a computational system "situated in some environment, that is capable of *flexible autonomous* action in order to meet its design objectives". *Situatedness* means that the agent affects and gets affected by its environment in some way. *Autonomy* is considered in the sense that the system is capable of acting without direct human intervention, controlling its own action and internal states. *Flexibility* concerns the *responsiveness* of the agent (capability of perceiving and responding in a timely fashion), its *pro-activeness* (capability of taking the initiative and exhibiting goal-directed behaviour); and its *social* dimension (capability of interacting in different ways with other artificial and human agents).

2.2 Organisational and Social Concepts in Multiagent Systems

Human organisations define the roles and responsibilities for organisational participants, who are expected to bring those into action depending on the task and environmental demands. This notion of organisation was early introduced to computational systems within the field of DAI [Gasser et al., 1987, Pattison et al., 1987]. Thus, early work in DAI identified organisational design as one of the main issues in order to cope with the complexity of designing DAI systems. However, DAI research has been accused [Decker, 1995] of having traditionally kept an individualistic character, focusing on the principled construction of individual agents. It has evolved following an agent-centered view though tightly related research disciplines such as organisation and management theory have been out there and evolved considering organisations as first class citizens from the early 60's.

Regarding MAS in particular, these have been mostly considered as mere aggregation of agents. Nonetheless, recently there is an increasing interest in incorporating organisational concepts into MAS as well as in focusing on organisation-centered designs as an alternative to the classical agent-centered designs motivated by the gradual influence of organisation theory. Organisation-centered approaches do not consider that an organisation design may emerge spontaneously and instead this is imposed in order to structure activities or control the actions of a system as a corporate entity. A shared organisational structure provides agents with descriptions about their roles and responsibilities in the multi-agent context and contains guidelines for their intelligent cooperation and communication. In other words, an organisational structure defines a behaviour space for agents with a whole set of norms and rules that agents are required to follow.

In what follows we describe different approaches appearing in the literature dealing with organisational aspects of multi-agent systems.

Most prominent, early DAI works introducing organisational aspects are reported in [Corkill and Lesser, 1983, Gasser et al., 1987, Werner, 1987]. Corkill et al. [Corkill and Lesser, 1983] developed computational representations for organisations in terms of interest areas for agents. An agents' interest area stand for the types of data-processing tasks that it is willing an able to take on, and to what extent. Given a variety of possible actions to take, an agent evaluates how well such actions fit within its areas of interest. Each agent knows the interest areas of the others.

The MACE system [Gasser et al., 1987][1] provides a language for defining multi-agent organisations and for evaluating organisational structures. It includes the notion of recursive composition of agents so that groups and groups of groups can be treated as agents. Each agent has a set of acquaintances defining its boundaries of communication and interaction, i.e. its social structures. Both social knowledge and social structure are founded on the modelling of other agents. Thus MACE includes facilities to model other agents' features (such as roles, goals, skills, etc.) in order to structure individual interactions, and so to establish social structure defined as patterns of interaction.

In both [Corkill and Lesser, 1983] and [Durfee et al., 1987] a variety of organisational structures are evaluated.

Werner [Werner, 1987] describes a unified theory of communication, intention, and social structures to develop a theory of social cooperation for multi-agent systems. Social structures are built from roles among which permissions and responsibilities are defined. They also contain the language making possible the agents' interaction as well as an environment on which agents can perform actions. Societal structures are intended to represent efficient, sophisticated, and permanent societal cooperation. Therefore, Werner argues that the formation of social structures avoids the costly process of building up cooperative intentions among agents whenever cooperation is required. Notice however that Werner's approach has a strong agent-centered flavour since the individual mental attitudes of agents are strongly considered.

Such early work has recently found continuation due to the growing influence of organisational theory and computational organisational theory in multi-agent systems.

So et al. [pa So and Durfee, 1998] argue that the model of organisation is tightly related to the model of the task the organisation is used for. Thus they consider that an organisation model must include: the set of tasks and subtasks to be done; the set of agents participating in the organisation; an assignment of the tasks and subtasks of the participating agents; a work flow structure dictating how the tasks and subtasks are to be distributed among agents and how the results and partial results are to be synthesised.

Likewise [Gasser et al., 1987, Corkill and Lesser, 1983], [Barbuceanu et al.,

[1] Here we briefly summarise its features since a more thorough description is provided in Section 2.5.

1998] and [Moss and Edmonds, 1997] propose agent development languages that allow to model organisations.

Moss et al. [Moss and Edmonds, 1997] developed SDML (Strictly Declarative Modeling Language), a multi-agent object-oriented language for modelling organisations. SDML is theory-neutral with respect to the capabilities of agents and, furthermore, includes a library for alternate agent architectures. In SDML the structure of a multi-agent system is represented as a container hierarchy. Containers and their associated agents are also linked by an inheritance hierarchy. But one of its most distinguishing features, from our view, is that within organisations there are predefined linkages among agents and predefined roles in which knowledge is embedded with the purpose of constraining behaviour.

Barbuceanu et al. [Barbuceanu et al., 1998] present an agent development language that covers the spectrum from defining organisations, roles, agents, obligations, goals, and conversations to inferring and executing coordinated agent behaviours in multi-agent applications. Though allowing for the definition of organisations, this approach exhibits a definite agent-centered view. Thus, it focuses on the realisation of agents capable of representing the organisation's normative structure in the form of obligations and inter-dictions and also efficiently reasoning about them to find non-violating courses of action. Then negotiations among agents assume that an agent can let others know about its list of obligations and inter-dictions.

In [Ferber and Gutknetch, 1998,Wooldridge et al., 1999] we find two different methodologies of agent-oriented analysis and design founded on the view of a system as a computational organisation consisting of various interacting roles.

Wooldridge et al. [Wooldridge et al., 1999] advocate for an *organisational design* of agent-based systems that regards them as artificial societies or organisations. They propose to design an organisation by defining its roles, along with their relationships, and their patterns of interaction. Then the functionality of each role is expressed as *liveness* and *safety* responsibilities. On the one hand, an agent's liveness properties specify the potential execution trajectories through the activities and interactions associated to the role. On the other hand, safety responsibilities are defined as the invariants to be held by a role that guarantee undesirable conditions to come about.

Two major criticisms apply to this approach. On the one hand, though exploiting the notion of organisation, Wooldridge et al. still keep a very much agent-centered view. On the other hand, the notion of organisational structure is not explicitly but implicitly defined.

A more organisation-centered approach is taken in [Ferber and Gutknetch, 1998]. Organisations are described using the core concepts of groups, agents, and roles. A role is regarded as an abstract representation of an agents' function, service or identification within a group. Groups are agent aggregations in which agents can interact according to the rules of pre-defined role-to-role protocols. An organisational structure is defined as a collection of groups and connections among groups standing for the paths that agents may follow to move from group to group. An important aspect of this approach is the introduction of a new model, organisation-based model, of reflection with the purpose of realising the organsiation's infrastructure. Finally, the MADKIT development environment is provided in order to support the design of multi-agent organisations.

Although from the works analysed so far it might look the contrary, it is important to notice that the division of labour is not the only concern in organisational design [Carley and Gasser, 1999]:

> The key issue under organisational design is not what is the appropriate division of labor, nor is it how should agents be coordinated. Rather, there is growing understanding that there is a complex interaction among task, agent cognition or capabilities, and the other structural and procedural elements of organisational design. As such, the issue is finding what combinations of types of agents, structures (patterns of interactions among agents), and ways of organizing the task are most likely to meet the organization's goal.

From an organisational engineering perspective, locating an optimal organisational design for a specific, multi-dimensional situation is key.

Notice that organisational approaches tend to consider multi-agent systems as corporate entities with specific goals in which agents interact according to the rules of a normative structure, which describes roles and interactions among roles separately from the agents occupying the roles. Furthermore, multi-agent systems are conceived in isolation from their environment.

Social approaches tend to have a wider scope and so they may consider open agent societies. These approaches are not only concerned about the normative structure of agent societies, but also about the social relations holding among agents, such as, for instance, joint intention [Cohen and Levesque, 1991], (social) commitment [Castelfranchi, 1995], dependence or power [Castelfranchi, 1996], etc. In general, they are mainly focused on understanding how the individual mind links to the collectivity. Therefore an agent-centered view is adopted in order to understand and analyse under what conditions certain social relations hold. The results obtained by this strand of research encompass ontologies and formal models that describe social concepts, as well as their various states and operations.

At this point we observe that while organisational approaches tend to disregard openness issues though taking a global perspective of the system, social approaches tend to keep an agent-centered view though eventually considering agents as members of a possibly open system.

Openness in multi-agent systems brings up a large variety of issues, already identified in Section 1.1, that deserve special treatment. When considering open agent systems there is not only the matter of proposing more or less sophisticated mechanisms for coordinating the activities of agents, or of understanding under what conditions, for instance, an agent decides to adopt a joint goal, but also the fundamental issue of providing an infrastructure containing mechanisms that ensure the proper workings of the system as a whole. And this appears as a delicate issue since the system is bound to be populated by multiple, heterogeneous agents which may eventually exhibit faulty or dysfunctional behaviours.

Although open systems have recently started to be considered by MAS researchers as the most important application of multi-agent systems, their inherent issues have not been conveniently addressed yet. In this thesis we focus on the formalisation, design of construction of open multi-agent systems following an organisational approach, along the lines of the early works in DAI.

Already in our early work [Rodríguez-Aguilar et al., 1997] and [Noriega, 1997b], it is noted that human societies have successfully coped with similar issues to those inherent to the design of open multi-agent systems by creating *institutions* that set and enforce laws. In both works it is advocated the adoption of a mimetic strategy for the realisation of open multi-agent systems by deploying *electronic institutions*. In this thesis, we continue, refine and extend and [Noriega, 1997b], and uphold that open multi-agent systems can be effectively designed and implemented as *institutionalized agent organizations* (or *electronic institutions* for shorter). Computationally speaking we defend that not only agents compose an open multi-agent system but also the computational mechanisms realising the rules of the society. By taking this perspective of the society designer, we manage to draw a clear distinction between the rules and the players, considered by [North, 1990] as key in the design of institutions.

To the best of our knowledge only Dellarocas [Dellarocas and Klein, 1999] follows a similar approach to realise open agent societies. In [Dellarocas and Klein, 1999] an agent society is reported to count on several social institutions, namely the *socialisation service* —in charge of establishing social contracts indicating membership of agents to the society and their rights and capabilities—, the *notary service* —in charge of verifying that agents' interactions are legal—, and an agent-based *exception handling service* —realised by means of a special type of agents, the so-called *sentinel* agents. Notice that special care is taken as to the handling of exceptions (more thoroughly discussed in [Klein and Dellarocas, 1999]) in order to guarantee the robustness of the society.

Differently, in our approach the major concern, and contribution and novelty at the same time, is the principled development of electronic institutions. At this aim, we offer an integrated approach which departs from a specification formalism, proposes a wholly agent-oriented design and finally demonstrates its usefulness by implementing a proof-of-concept agent-mediated electronic marketplace. In this way we pretend to illustrate how to jump from specifications to actual implementations.

Firstly, similarly to [Gasser et al., 1987, Moss and Edmonds, 1997, Barbuceanu et al., 1998] , we introduce a language to define an electronic institution. But instead of a programming language, we introduce a novel graphical specification formalism for electronic institutions, building upon our previous work [Rodríguez-Aguilar et al., 2000, Esteva et al., 2000b, Esteva et al., 2000a]. Why a formal specification language? Because we consider that a formal technique must me used when we cannot afford to have the requirements misunderstood. And this is the case since electronic institutions are highly complex, critical applications.

Secondly we present a computational model that arises from and thoroughly captures the formalisation of electronic institution introduced. Our computational model proposes to realise an electronic institution by means of a fully agent-oriented design based on two types of agents: *institutional agents* and *interagents*. Differently to [Wooldridge et al., 1999] we adopt a totally organisation-centered approach.

Institutional agents are those agents to which the institution delegates its services (f.i. registration, accountability, brokering, etc.). We detail a general, flexible computational model of institutional agent based on the notion of interpreting responsibility plans with which every institutional agent is endowed. In this way, all institutional agents are realised by means of the same computational model, only differing on their assigned responsibilities. Apart from flexibility, this model intends to provide support for scalability purposes (institutional services might be loaded into a single agent or else distributed among multiple agents).

Interagents are a special type of facilitators mediating all interactions among agents; responsible for enforcing the institutional rules[2].

Finally, an electronic institution exhibits an explicit organisational structure building upon the notions of *role*, *scene*, *performative structure* and *normative rules*. Agents, like in most organisational approaches, adopt different roles within an institution. A scene, similarly to the notions of patterns of interaction [Gasser et al., 1987], social structures [Werner, 1987], conversation [Barbuceanu et al., 1998] and groups [Ferber and Gutknetch, 1998] is intended to model a particular activity involving multiple interacting agents. A performative structure allows to compose scenes, establishing different relationships, to model the whole institution activity as a network of scenes. For the sake of expressiveness, instead of defining performative structures as graph-like scene networks (such as in [Ferber and Gutknetch, 1998], we provide several types of connectors for scenes so that a larger variety of relationships among scenes can be expressed (f.i. synchronization, parallelization, sequencing, etc.). Likewise [Barbuceanu et al., 1998] and most social approaches, we also consider the notion of commitment as fundamental in the context of an electronic institution. Thus, agent actions in the context of an institution have consequences, usually in the shape of compromises which impose obligations or restrictions on actions of agents in the scenes wherein they are acting or will be acting in the future. We introduce the so-called normative rules to make explicit the obligations or prohibitions fired by agents' actions.

2.3 Agent Communication and Coordination

Independently of whether the development of an agent-based system is guided by a principled methodology or not, there is the fundamental issue of deciding how to make agents interact.

So far, much effort in agent research concerning agent interaction has focussed on the semantic and pragmatic foundations of different agent communication languages (ACLs) based on speech act theory [Austin, 1962,Searle, 1969]. Speech

[2]A more thorough discussion about the features of interagents is provided in the next section

act theory is founded on Austin's observation that utterances must not be considered as simple propositions, but as a speaker's attempts that either succeed or fail [Austin, 1962]. Research in agent interaction departs from this assumption to understand and explain how agents communicate with one another.

ACLs were originally proposed as the means to allow the exchange of knowledge among software agents in order to make easier their inter-operation [Genesereth and Ketchpel, 1994]. Generally, an ACL is composed of three main elements: an open-ended vocabulary appropriate to a common application area [Gruber, 1993b], an inner language (f.i. KIF–Knowledge Interchange Format [Ginsberg, 1991]) to encode the information content communicated among agents, and an outer language (f.i. KQML–Knowledge Query and Manipulation Language [Finin et al., 1995]) to express the intentions of agents.

However, current ACLs, such as KQML [Finin et al., 1995], FIPA ACL [FIPA, 1998], or FLBC [Kimborough and Moore, 1997], are valuable for the traditional message-passing processing but they do fail to capture the notion of message context, i.e. the context of agent interactions. This is fundamentally because most work on speech acts, from which ACLs take inspiration, has focused on individual utterances. In fact, the major contributions of speech act theory are related to the understanding of the relationship between an agent's internal state and the utterances that it exchanges with other agents.

The proved inability of current ACLs to capture the context of agent interactions has motivated the recent promotion of the notion of conversation as the unit of communication between agents[3]. A conversation is a pattern of message exchange that some agents agree to follow in communicating with one another. Conversations are useful for structuring interactions among agents by organising messages into relevant contexts. The use of conversations allows to attach context to the sending and reception of messages, easing interpretation. Hence that current agent research in ACLs is broadening from the specification of individual utterances to include the characterisation of goal-directed conversations for which agents will use ACLs. In this new context, the representation, specification, validation and implementation of conversations arise as new issues. Nevertheless limited work has been done on implementing and expressing conversations for software agents.

New works in agent communication research attempt at modelling and reasoning about the relationships within and among conversations, or groups of utterances. Even there appears to be a growing consensus that the notion of conversation is the principal factor to consider in defining an ACL's formal semantics [Pitt and Mamdani, 1999,Smith et al., 1998].

A conversation-based approach must be valued according to the conversational model proposed. In this section we offer a review of current conversation models, analysing their contributions and drawbacks.

Traditionally most proposals concerning the modelling of conversations have

[3]Although this remark is also made in the FIPA specification [FIPA, 1998], it is only proposed to include a `:protocol` slot in the syntax of performatives, and a simple protocol specification language.

fundamentally exploited Finite State Machines (FSMs). Winograd and Flores pioneered this research proposing the use of state transition diagrams [Winograd and Flores, 1988]. Currently the *COOL* language [Barbuceanu and Fox, 1995] has perhaps the most complete finite state model to describe agent conversations. Furthermore, it has been practically employed for modelling the supply chain of manufacturing enterprises. In COOL each arch in a transition diagram represents either a message transmission, a message receipt or both, so that different agents must use different automata to engage in the same conversation. In the COOL language, conversations are computational objects containing conversation rules, error recovery rules, continuation rules (applied to restart suspended conversations), and conversation nesting. COOL use an `:intent` slot within the performative to help the recipient choose the conversation structure needed for understanding the message.

The *KAoS* [Bra, 1997],*Jackal* [Cost et al., 1998] and *Agentis* [d'Inverno et al., 1998a] systems also employ FSMs to model conversations.

The implementation of conversations in KAOS is realised by means of *off-line* design: conversation laws are hard-wired in advance into agents. Evidently this rigid, early approach does not contribute to produce flexible systems. KAoS introduces the notion of conversation *suite*, defined as a grouping of conversations that support a set of related services commonly available to agents, as a primitive form of conversation sharing. Smith et al. [Smith et al., 1998] have recently proposed semantics for the conversation policies developed for KAoS based on the theory of joint intentions.

The Jackal system originally started employing a loose Extended Finite State Machine to model conversations built as a Deterministic Finite State Automaton extended with a local read/write store. But currently the work in Jackal has shifted to the modelling of conversations by means of concurrency models [Cost et al., 1999] as detailed further on.

Agentis [d'Inverno et al., 1998a] is a framework for constructing interactive multi-agent applications based on a model of agent interaction whose characterising elements are services and tasks. A fundamental element of the system is the set of protocols underlying the provision of services and tasks for agents. Agentis contains a single model to cover all different types of inter-agent interactions based on four types of basic protocols. Interestingly, a formal specification in Z of these protocols is provided as part of a more complete and formal specification of the entire system. Furthermore, the execution of protocols is described in terms of the operations that transform it from one state to another, and hence it is showed how the actual communication among the agents of a system occurs. The use of Z as the protocol specification language allows to take advantage of the wealth of tool support for verifying the design. However, in order to analyse the properties of an Agentis system, the authors admit the need for a concurrent model (such as CSP).

The *AgenTalk* [Kuwabara, 1995] and *Infosleuth* [Nodine and Unruh, 1999] systems add some innovations to the use of FSMs by introducing methods for composing and reusing them.

On the one hand, the *AgentTalk* system [Kuwabara, 1995] employs inheritance as a mechanism for extending a finite state automaton to support new conversations. A conversation is inherited by adding new states and/or transitions, and by overriding existing states and transition. Then conversations in AgentTalk are specified as a set of states, a set of transition rules, a set of local script functions invoked when certain events come about, and lastly the conversation from which the specified conversation inherits.

On the other hand, the *InfoSleuth* system [Nodine and Unruh, 1999] introduces extensions and concatenations of conversations. Extensions are defined in order to enrich the transitions that can occur in the middle of a conversation. Extensions, differently to the inheritance methods proposed by [Kuwabara, 1995], are predefined and applicable to multiple other conversations. A concatenation is an operation over conversations that produces a new conversation specification by connecting the final state of a conversation with the initial state of another conversation. But perhaps one of the most important contributions made by [Nodine and Unruh, 1999] is the introduction of the notion of conversation enforcement. InfoSleuth implements conversations by incorporating a conversation layer in the agent's generic shell and by separating the conversation management with the remote agent from conversational interactions with the local agent.

Conversation enforcement is also brought up as a fundamental issue in this thesis when introducing our model of mediation in Chapter 4. But differently to [Nodine and Unruh, 1999], in [Martín et al., 2000b] conversation management is delegated to a special type of facilitators, the so-called *interagents*, instead of employing an agent's built-in conversation layer. Interagents —like KQML facilitators [Patil et al., 1992]— are autonomous software agents inspired by the *efficient secretary* metaphor already introduced in the Actors model of concurrent computation [Agha, 1986].

Interagents are key for the realisation of open agent systems. Instead of allowing the direct interaction of agents with the rest of agents in the society, we oblige agents to have all their interactions filtered by interagents so that these can validate them and enforce them conveniently. Along this direction, the use of interagents has proved to be valuable in the development of two different types of agent-based systems: the agent-based electronic auction house presented in Chapter 5, and the multi-agent learning framework Plural [Martin et al., 1998], which tackles the problem of sharing knowledge and experience among cognitive agents that co-operate within a distributed case-based reasoning framework.

We model and implement conversation protocols for interagents as a special type of Pushdown Transducer (PDT), which can be seen in turn as a combination of a Finite-State Transducer (FST) and a Pushdown Automaton (PDA). The choice of PDTs as the mechanism for modelling CPs is motivated by several reasons. First, analogously to other finite-state devices a few fundamental theoretical basis make PDTs very flexible, powerful and efficient [Roche and Schabes, 1997]. They have been largely used in a variety of domains such as pattern matching, speech recognition, cryptographic techniques, data compression techniques, operating system verification, etc. They offer a straightforward mapping from specification to implementation. PDTs, unlike other finite state devices, allow us to store, and subsequently retrieve, the contextual information of ongoing conversations. In order to ensure the correct exchange of illocutions during a conversation, we introduce important, desirable properties such as termination, liveness, and lack of deadlocks and race conditions that conversation protocols must verify. Although conversation analysis has not deserved much attention in the literature, we regard it as a fundamental issue since not only is important a conversation protocol specification but its subsequent validation.

Several alternative modelling techniques to FSMs have been also put forth to model agent conversations.

In [Labrou, 1997] Labrou investigates the specification and semantics of KQML, and the logical architecture of KQML–speaking agents. Building on the semantics, he derives conversation policies as formal descriptions of how KQML performatives may be combined into KQML exchanges using a *Definite Clause Grammar* (DCG). Labrou is not only concerned about specifying the permissible sequences of performatives within a conversation, but also about capturing contextual information as constraints and dependencies on the values of the *reserved* parameters (`:reply-with` and `:in-reply-to`) of the performatives involved in the conversations. Conversation protocols are expressed using a DCG in which each KQML message is a terminal, defined as a list of the following values: `:performative_name`, `:sender`, `:receiver`, `:in-reply-to`, `:reply-with`, `:language`, `:ontology`, `IO` (set to 1 for incoming messages and to 0 for outgoing messages).

According to Labrou's approach, in general when an agent sends a message, it has expectations about how the recipient is to respond back. Those expectations are encoded into the message itself, instead of employing a higher-level structure (a conversation item) embedding the message.

In [Moore, 1999], Moore proposes to represent conversations using the state-chart formalism defined originally by Harel [Harel, 1987] to model the behaviour of reactive systems. Moore defends state-charts as an appropriate modelling language to deploy in the research of conversation modelling after considering that there are many commonalities shared between agent communication systems and reactive systems. Certainly, state-charts constitute a highly expressive, modular visual formalism. In [Moore, 1999] it is showed how they can be useful to model conversations at different levels of abstraction, as well as to model sub-conversations and concurrent conversations. Notice however that Moore's proposal is solely presented as a formal specification that is not accompanied and realised by a computational model.

Parunak [Parunak, 1996] introduces Enhanced Dooley Graphs with the purpose of capturing both the possible relationships among utterances, likewise state models, and the actual relationships between utterances once the conversation happened.

Thus, Enhanced Dooley Graphs are generated from an actual interchange among existing agents. Each node in a Dooley Graph [Dooley, 1976] of a discourse is said to represent what Parunak calls a *character*, a participant at a *distinguished* stage of a discourse, i.e. either at the overall conversation, at a nested conversation or at a side-bar conversation.

Due to the inherent difficulty in predicting the behaviour of an agent-based system, Parunak proposes the analysis of the resulting interactions via simulation with the aid of Enhanced Dooley Graphs and their underlying basic Dooley Graphs. A (basic or enhanced) Dooley graph induces a topology whose features provide the basis for obtaining quantitative measures that help analyse the dynamics of the discourse. For instance, the number of separated components in the basic Dooley Graph represents the number of side-bar conversations undertaken by the participants in response to the state of the overall conversation, instead of in reply to specific utterances.

Then the analysis of the size and participants of each component provides further information. Thus, for instance, components of similar size reflect concurrency and parity among participants, whereas the appearance of a large component and several smaller components indicates centralisation and domination by one player. Similar measures can be derived from the lengths of the paths in the graph and the degree of the characters (nodes).

Though valuable as an analysis tool, Parunak does not make clear how to employ Dooley Graphs to guide the subsequent implementation of conversations. Furthermore, although a Dooley Graph captures the rhetorical structure of a conversation, it hides the control flow among the agents where more than one character of an agent is involved. In [Singh, 1998] it is showed how Dooley Graphs can be effectively used for generating the agent skeletons needed for coordinating heterogeneous, autonomous agents. A skeleton, such as defined in [Singh, 1998], is a model of an agent's externally visible actions or events. Notice however that both works depart from the trace generated during an actual conversation to step back to the design of the conversation patterns needed by agents.

Recently agent researchers have resorted to well-known and established models of concurrency endowed with verification mechanisms in order to overcome the weaknesses of FSMs: though FSMs help us depict the flow of communication in a very intuitive way, and are indeed sufficient for expressing many sequential interactions, they lack of expressiveness to model more sophisticated interactions involving some degree of concurrency. More precisely, there is a body of research in conversation modelling that advocates for the use of Petri nets as an effective model for supporting a greater range of interactions. Petri nets are argued to provide many of the benefits of FSMs, and besides they offer greater expression and concurrency. The purpose of the introduction of concurrent models is to allow for the specification of complex multi-agent conversations (not just dialogues) that even allow for changes in the set of participants during the course of interaction.

Along this direction, the Java-based Agent Framework for Multi-agent Systems (JAFMAS) [Chauhan, 1997] uses automata models for specifying conversations —based on COOL—, which can be subsequently converted into a Petri net representation. The conversion of conversations into Petri nets allows for the use of existing Petri net analysis methods to check the consistency and coherency of the conversation. This is achieved by checking the satisfaction of safeness, liveness and no deadlock properties. JAFMAS is planned to have its capabilities extended in order to assist designers in validating conversations [Galan and Baker, 1999]. In this way, conversation specifications will be automatically converted into a single Petri net that is subsequently analysed using an outer Petri net verification tool.

Petri nets are also employed in [Koning et al., 1998] for modelling conversations. However, the approach adopted is slightly different from the approach used in JAFMAS. Thus, in this work a conversation is specified by building a Petri sub-net per agent as well as a sub-net for the medium connecting the agents involved in the conversation. Likewise JAFMAS, the Petri-net corresponding to each agent is constructed from an FSM specification. Therefore, the whole model is obtained by assembling the Petri-net sub-net corresponding to the medium with those corresponding to agents. A distinguishing feature of this approach is that it offers software tools that assist designers in the analysis of specifications in order to check their validity. The validity process involves the checking of some elementary properties: a conversation is required to be *bound* (with a finite number of states), *quasi-livable* (all conversation messages may be eventually used), *deadlock-free* and *terminating*. Perhaps the major drawback of this contribution is that it does not make clear how to practically realise conversation specifications.

A particular type of Petri nets, Coloured Petri Nets (CPNs) [Jen, 1995a], has been also proposed for conversation modelling [Cost et al., 1999, Lin et al., 1999]. Though CPNs are formally equivalent to traditional Petri nets, their richer notation facilitates conversation modelling to designers. The major difference between CPNs and traditional Petri nets roots in their tokens. While tokens in Petri nets are simply markers, in CPNs tokens can be typed, and so they can have data associated to them.

In [Cost et al., 1999] the use of CPNs is proposed as the formal model underlying the *Protolingua* conversation specification language investigated in the framework of the Jackal development environment. Conversations are firstly designed and analysed at a high level with the aid of CPN-ML[4], with the aim of subsequently translating them into Protolingua protocols. In [Lin et al., 1999] CPNs are also employed as a specification and verification tool at the design stage, but this time for the so-called conversation schemas. A conversation schema is defined as a pattern of agent interaction specifying conversations that may be centered on one or more domain-dependent topics.

Due to the intrinsic limitations of the Petri net formalism (in particular, the combinatorial explosion of the network size for actual protocols and the ultra-powerful simulation tools needed), Koning proposes in [Koning, 2000] to alternatively use Lotos as a more appropriate conversation specification language. Lotos is a standard (ISO8807) specification language built upon a process alge-

[4]An extension of ML that allows the specification of CPNs.

bra that provides a high degree of abstraction by making use of abstract types. Lotos allows to model multiprocess systems. A distinguishing feature of Lotos is that Lotos specifications can be easily translated into finite state automata. Then it is possible to make use of validation tools over the resulting automata. This is the main reason why Lotos is claimed to be particularly suited for formal verification. The interesting contribution in [Koning, 2000] roots in the proposal of an automated translation from a logical model of a conversation protocol into a Lotos specification that can be employed to verify the properties of the specified conversation with the aid of the CADP tool box [Garavel, 1996]. Once validated the designed conversation protocol, pieces of code that agents can use in order to exploit the protocols are generated from the logical model.

The main and general criticism to be held up concerning the existing approaches to agent interoperability is that the integration of facilities to support the specification, validation and implementation of conversations into an agent development environment are still at a very early stage.

2.4 Agent-based Computational Markets

In this section we survey the most salient agent-based virtual market places. Next we present the most relevant features of each system, leading such analysis to establishing a brief comparison of the characteristics of each one as shown in Figure 2.4.

Kasbah [Chavez and Maes, 1996, Chavez et al., 1997] is a web-based and agent-based consumer-to-consumer electronic marketplace. Users intending to buy or sell goods can create buying and selling agents that help them transact their products. When creating agents, users must give them some strategic direction, and afterwards send them off into a centralised agent marketplace.

Kasbah agents pro-actively seek out potential buyers or sellers so as to negotiate with them on their owners' behalf. Thus these agents are responsible for automating much of the merchant brokering and negotiation stages for both buyers and sellers. Each agent arriving into the central marketplace is targeted to close an acceptable deal according to a set of user-specified constraints such as desired price, lowest (or highest) acceptable price, and a deadline to complete the transaction.

Negotiation in Kasbah is rather simple. Once buying and selling agents match, buying agents can offer a bid to selling agents that selling agents can either accept or reject. When configuring their buying agents, buyers can choose among three negotiation strategies: anxious, cool-headed, and frugal —corresponding to a linear, quadratic, or exponential function respectively for increasing its bid for a product over time. These are indeed very intuitive negotiation strategies that help users to understand their buying agents' trading behaviour.

Among the innovative features of Kasbah there is the incorporation of a distributed trust and reputation mechanism [Zacharia, 1999]. It can be used by both parties completing a transaction to rate how well the other party managed

Features/**Systems**	**Kasbah**	**Fishmarket**	**AuctionBot**	**Zeus**	**Magnet**	**eMediator**
Distribution of agents	Web-based client, server-based virtual market	Web-based, distributed buyers and sellers	Web-based client, server side agents	Buyers and sellers distributed across a LAN	Buyers and sellers distributed across a LAN	Web-based plus on-site mobile agents
Trading Agents' Support	Limited to choosing a fixed strategy	■ customisable trading strategies ■ API for SW traders	API for SW traders	■ customisable trading strategies ■ embeddable strategies into generic trader	API for SW traders	customisable mobile agents
Negotiation (two-way bargaining)	yes	no	no	yes	yes	no
Auction Types	no	single-sided(Dutch, English,First-price Sealed Bid, Vickrey)	single-sided double-sided	single-sided	Vickrey	single-sided double-sided combinatorial
Seller:buyer cardinality	1:many	1:many	1:many,many:1 many:many	many:many	1:many,many:1	1:many,many:1 many:many
Merchant brokering	Facilitated by dedicated broker agents	Auctioneer chooses goods to offer from available lots	Users choose vendors from category index	Users choose vendors from publicised offers	Yellow pages of services and markets	Users choose vendors from category index
Mobility	no	no	no	no	no	yes
Reputation	Collaborative Reputation Mechanisms	no	no	no	no	Better Businees Bureau
Monitoring & Debugging Facilities	no	yes	no	yes	no	no

his half of the deal (e.f. accuracy of product condition, completion of transaction, etc.). Therefore users can establish a reputation threshold when creating their trading agents that these must utilise for discriminating which agents are eligible for negotiation.

AuctionBot [Wurman et al., 1998] is a web-based, flexible auction server that supports both software and human agents. AuctionBot is capable of managing a large number of simultaneous auctions, which are organised in a hierarchical catalog. The user initiating an auction can position it anywhere within the existing catalog, or extend the catalog to create an appropriate subcategory.

The most distinguishing, interesting feature of AuctionBot is that it manages to support a wide range of auction types by decomposing the auction design space into a set of orthogonal parameters. Based on such parameters, the classic auction types can be generated, plus others that have not been studied yet. The identified parameters are classified in one of the following categories:

- *Bidding restrictions.* These parameters determine acceptable bids. They basically encompass parameters referring to participation (whether the auction allows one buyer or many buyers, one seller or many sellers) and bidding rules (additional restrictions on allowable bids such as, f.i. the bidders' obligation to beat the current price quote).
- *Auction events.* They refer to the frequency and number the matching algorithm is run to produce an allocation, the logical conditions determining the closing of an auction, and the number of frequency of price quotes.
- *Information revelation.* They refer to the degree of information revelation concerning current and past transactions (f.i. trader's identity, timing of upcoming clear,etc.)
- *Allocation policies.* They are the most important parameters because an allocation policy dictates which agents transact, and at which price(s), as a function of the bids. Currently, AuctionBot supports three different allocation policies: the *Mth-* and *(M+1)st*-price policies as generalisation of the first-price and second-price mechanisms to auction multiple units, the *chronological match policy* implementing the sequential trade effect of the Continuous Double Auction, and the *tie-breaking rules* employed for breaking ties (f.i. choose the earliest bid).

AuctionBot provides full support for human agents through a web-based interface and for software agents implementing the AuctionBot API, a communication specification that developers follow to implement software that communicates with the AuctionBot server over TCP. Currently, AuctionBot supplies C++ and Java implementations of the API for developers. Both AuctionBot and Fishmarket were the first auction houses providing full support for the participation of software agents. However they follow different approaches; while AuctionBot allows a software trader to directly connect to the auction server, Fishmarket obliges that trader to have his participation mediated by a special type of facilitator provided by the auction house.

Though empowered to let software agents play, it does not provide any facility to allow users to create customised automated autonomous (or semi-autonomous) trading agents.

To the best of our knowledge, though neither evaluation facilities nor visualisation tools are provided, AuctionBot supports the text-based monitoring of the trading activity.

Zeus [Collins and Lee, 1998] is not a computational market, but an agent building tool-kit that has been successfully employed for constructing a virtual electronic marketplace[5]. The resulting marketplace was developed as a virtual commodity auction with a variable number of agents capable of buying and selling items. The marketplace is open, with an indeterminate number of participating agents spread across a LAN (local area network). This is different from Kasbah, Fishmarket and Auctionbot, which all allow for agents to be distributed over the web.

The implementation of the agents participating in the marketplace was ensembled from the ZEUS *agent component library* with the aid of the ZEUS *agent building tool.* Users are offered the possibility of creating their own agents by customising their bidding and selling strategies via a GUI. Then users can opt for either launching their agents to work autonomously or for instructing them to work semi-autonomously. This facility is also provided by Kasbah and Fishmarket. But the marketplace created with ZEUS was designed to allow individual agent designers to include their own strategies into their agents. In other words, agent designers can create new strategies and embed them into a generic trading agent provided by the marketplace. This approach is between the solutions proposed in Fishmarket and Auctionbot. Thus on the one hand an agent designer is not allowed to directly connect his own agent to the marketplace like AuctionBot does. Instead he is forced to nest his strategies into a generic agent so that the marketplace can secure that the agent behaves according to the rules of the market handled by the generic agent. Notice that differently the so-called interagents employed in Fishmarket remain separated from external agents containing their own strategies, and so strategies and coordination mechanisms are kept separated into two different agents.

An interesting feature of this marketplace is the availability of visualisation tools, inherited from ZEUS, which can be used to view, analyse or debug the marketplace. The visualisation tools succeed in integrating the large amounts of limited, local information generated by each agent into a coherent whole. This type of facility is only provided by Fishmarket.

Lastly, the marketplace also provides means for agents to discover what is on offer, and where on the network the vendor can be found. For this purpose, the ZEUS facilitator agents provide a brokering service as a *classified ads* service, enabling agents wishing to buy or sell particular items to identify potential trading partners.

[5]A similar approach has been recently reported in [Albers et al., 1999] where the agents of an electronic marketplace were designed and developed using DESIRE [Brazier et al., 1998], a compositional development method for multi-agent systems.

MAGNET [Collins et al., 1998] is a computational market designed to act as an intermediary for multi-agent negotiations. It implements a generalised multi-agent market architecture that provides explicit and integrated support for complex multi-agent interactions; from complex negotiations to simple auction protocols. The market place builds upon three fundamental elements: *exchange* —a collection of domain-specific markets set up for the trading of goods and services, along with some generic services —, *markets* —forums for trading in a particular commodity or business area —, and *market session* —agent interactions occurring in the context of a particular market. Market sessions can be stored by the marketplace, and so the state of each transaction can be made persistent over time.

MAGNET allows for the participation of both software and human agents, likewise all systems described so far, spread over a LAN. Software agents are handled likewise AuctionBot, and so they can directly interact with MAGNET as long as they implement an API. Agents can play either as buyers or sellers with respect to any market session. However, there is only a single agent initiating the session for either buying or selling.

Using the CORBA trader service, each exchange maintains "yellow pages" of services and markets for brokering purposes. Furthermore, likewise *Market Maker* —the successor of Kasbah—, an exchange is endowed with a Better Business Bureau service that can provide to agent information about the reliability of other agents based on past performance.

eMediator [Sandholm, 1999b] is a web-based electronic commerce server that includes:

- an auction house, the so-called *eAuctionHouse* that allow users from across the Internet to buy, sell, and set up auctions;
- a leveled commitment contract optimiser that determines the optimal contract price and decommitting penalties for a variety of leveled commitment contracting protocols; and
- a safe exchange planner that enables anonymous exchanges by dividing the exchange into chunks and sequencing those chunks to be delivered safely in alternation between the buyer and the seller.

Undoubtedly the eMediator auction house is the most powerful auction site over the Internet, covering the largest auction space and being the first Internet auction house that supports combinatorial auctions. Due to the complexity of the auction space —the auction setting differs based on whether it is a single or double auction, whether there is one or multiple items, whether there is one or multiple units of each item, and whether the units can be divisible or indivisible— the user is assisted by the auction server in choosing an auction type out of a wide variety of auction types.

Once set up an auction, bidders can participate through either web browsers or on (or near) the host machine of the auction server itself. *eAuctionHouse* provides support for mobile agents that permit a user have agents actively participating in the auction while he is disconnected, and help reduce the network latency. Though having the same motivation than some commercial auction sites [eBay, www,Amazon, www] providing *proxy bidders*, the approach of *eAuctionHouse* is much more sophisticated.

Users can opt for coding their mobile agents themselves —likewise AuctionBot, Fishmarket and ZEUS—or using the support provided by *eAuctionHouse* to generate a mobile agent out of a large variety of mobile agent types (simple information agents reporting about special events, agents that implement dominant strategies, agents that underbid optimally, or agents that speculate in order to mislead the others).

Differently to other auction servers, *eMediator* provides additional services.

On the one hand, *eMediator* introduces the so-called *leveled commitment contracts*; contracts which enclose the level of commitment for each agent specified by decommitment penalties, i.e. the penalties that agents must pay in order to be freed from the obligations of the contract. A leveled commitment contract optimiser determines the optimal contract price and decommitting penalties for a variety of leveled commitment contracting protocols.

On the other hand, in order to guarantee contract execution, i.e. that the seller gets paid and that the buyer gets the goods, a method to tackle this problem without third parties is included into *eMediator*. As part of *eMediator*, a safe exchange planner enables anonymous exchanges by dividing the exchange into chunks and sequencing those chunks to be delivered safely in alternation between the buyer and the seller.

In this thesis, in Chapter 5, we present **FM96.5 (Fishmarket)**, an open, flexible, robust, agent-based auction house where heterogeneous (software and human) agents may trade. Along with *AuctionBot* [Wurman et al., 1998] it was the first auction house providing support to software agents. In order to assist agent developers in the construction of their agents, FM96.5 encloses a library of agent templates in several languages. Furthermore, it also offers the possibility of generating agents with customised trading strategies likewise [Collins and Lee, 1998] and [Chavez and Maes, 1996].

Considering auction protocols, FM96.5 includes, as a distinguishing feature from the rest of auction houses, a sound and fair implementation of a real-time Dutch auction protocol which is not included in other auction marketplaces. Although as a matter of fact originally it only included Dutch auctions, it had its capabilities extended in order to accommodate other classic auction types, namely English, First-price sealed bid and Vickrey.

One capital aspect which does not seem to be conveniently considered by most of the marketplaces described so far is the notion of accountability of the market activity. Likewise *Zeus* [Collins and Lee, 1998], FM96.5 bargains for monitoring facilities aimed at keeping track of all events taking place in the market as well as at integrating into a coherent whole the vast amount of information generated by individual agents in order to picture the behaviour of the system in such a manner that humans can better understand it.

But surely the most differentiating feature of FM96.5 with respect to the rest of computational marketplaces surveyed in this section roots in its inception as an *open agent-mediated electronic institution* founded on a formal specification language as described in Chapter 3 and a strong agent-oriented computational model as presented in Chapter 4.

To the best of our knowledge, FM96.5 is the only marketplace reported to explicitly exhibit organisational structure. Such organisational structure takes inspiration on the traditional trading institution, the fish market, presented in Chapter 1. Thus, based on a highly mimetic strategy and according to the formal model of electronic institutions presented in Chapter 3, FM96.5 is conceptually conceived as a virtual marketplace where simultaneous *scenes* occur, but with some causal continuity. Each scene represents a particular activity in the marketplace and involves a group of agents playing different roles that interact by means of illocutions according to a well-defined protocol. Such protocol is intended to capture the possible dialogues that agents may have in the course of the activity represented by the scene. Scenes in turn are connected composing a network of scenes, the so-called *performative structure*, which captures the existing relationships among activities. Such network can be also regarded as a map showing the possible activity roads, paths, that agents may follow within the marketplace depending on the role that they play. Finally, on top of the performative structure there is a set of constraints, the so-called *normative rules* that produce limitations or enlargements in the acting possibilities of agents depending on the actions that they take. For instance, an agent purchasing a good is committed to pay for it, and so it will not be allowed to abandon the marketplace till paying off.

Alternatively, Padget et al. [Padget and Bradford, 1998] develop an interesting π-calculus model of the fish market. Although their approach demonstrates the value and effectiveneess of π-calculus as a formal specification language for multi-agent systems, they take an agent-centered approach. Thus, they focus on the specification of the social behaviour of each one of the agents participating in the marketplace. There is a high ressemblance between their resulting specification and the way we specify institutional roles in our computational model, as described in Chapter 4.

FM96.5 is computationally realised by means of two special types of agents: *institutional* agents and *interagents*. Both types of agents are owned by the institution and are responsible for guaranteeing its sound operation by means of their coordinated activity. Notice that typically institutional agents, interagents and external trading agents will be running distributedly.

On the one hand, institutional agents are those to which the institution delegates its services. In fact, an institutional agent acquires as responsibilities the tasks associated to a particular institutional role when adopting the role (f.i. an auctioneer is responsible for auctioning goods, a sellers' admitter for registering goods, an accountant for the book-keeping of traders' accounts, etc.)

On the other hand, interagents are key in FM96.5 since their mediation guarantees that external trading agents are subject to the explicit —and enforceable—

behavioural conventions of the institution. But more generally, interagents are reported to be useful along a variety of directions: they allow to achieve the openness of the market, and so allowing the participation of heterogeneous agents; they contribute to the accountability processes undertaken by the market auditor (monitoring agent) by book-keeping agents' trace information; they locally manage trading agents' exceptions in order to prevent that an agent eventually jeopardises the global market activity; and lastly they are ready to accommodate societal changes since they are capable of handling institutions' specifications.

Furthermore, not only are interagents useful for the deployment of electronic institutions, but also of agent-based systems in general.

Summarising, notice that on the one hand an organisation-centered approach is adopted in order to structure FM96.5, and besides additional computational mechanisms are deployed to create the infrastructure of the electronic institution so that it can be open to external trading agents.

A brief comparison of the features of the marketplaces presented so far is shown in the table in Figure 2.4. The criteria used for comparison are as follows: *Distribution of agents* refers to the organisational architecture of the marketplace, *Trading agents' support* refers to the type of support provided to trading agents, *Negotiation* refers to the support for different negotiation protocols, *Auctioning* refers to the support for auction protocols, *Seller:Buyer cardinality* refers to the number of potential sellers who will be in competition for an individual transaction, *Merchant brokering* refers to the process whereby users identify who has a desired item for sale, *Mobility* refers to the capability of the maketplace of hosting mobile agents, *Reputation services* refers to the availability of reputation mechanisms, and *Monitoring and debugging* refers to the availability of monitoring and debugging facilities.

2.5 Agent Test-beds

Controlled experimentation is at the heart of Artificial Intelligence from its inception. It is fundamentally founded on two research tools [Hanks et al., 1993]: *benchmarks*, standardised tasks, and *test-beds*, environments in which AI programs can be studied. Both must be regarded as part of the apparatus of empirical AI, exclusively devoted to support and facilitate experimentation. Therefore, it is up to researchers to analyse experimental results and subsequently discern whether a general formal theory may be postulated.

One of the main purposes of benchmarks and test-beds is to provide metrics for comparing competing systems. In this direction, benchmark problems have been largely exploited by AI researchers by means of competitions. Recall for instance the *Robocup* initiative [Kitano et al., 1997] that encourages both agent researchers and robotics researchers to make their agents and robots play soccer; or the *AAAI Mobile Robot Competition* [Kortenkamp et al., 1997] where autonomous mobile robots try to show their skills in office navigation and in cleaning up the tennis court; or the Santa Fe Institute tournaments [Andrews and Prager, 1994] where the contenders competed for developing optimised trading strategies; and even automated theorem proving systems have been pitched against each other [Suttner and Sutcliffe, 1996]. Nowadays Robocup is undoubtedly in vogue in the agent community, motivating an important body of work.

Considering uniquely test-beds, they have been largely employed by AI researchers as a stepping-stone between formal theory and applications. In general, their main role is to provide support for operationalising theoretical ideas so that they can be tested, analysed and possibly validated. Ideally the use of test-beds can serve researchers to either validate their hypothesis or, alternatively, to elaborate new hypothesis founded on their empirical observations. In what follows we put benchmarks aside to survey existing test-beds for distributed artificial intelligence research.

DAI has a long tradition of reported, controlled experimentations and test-beds since its early days. Fundamentally, DAI test-beds have been exploited along two directions:

- to experiment with agent architectures; and
- to analyse the performance of coordination, communication and negotiation mechanisms.

In order to understand the support provided by DAI test-beds to researchers, we divide the facilities that they may provide into three principal types [Decker, 1996]:

- *Domain facilities* for the representation and simulation of the problem being solved. One of the key features of this type of facilities is the capability of scenario generation, essential for the repeatability of empirical experiences, that highly depends on the degree of experiment parametrizability offered by the test-bed. Another distinguishing feature is the type of domain modelled, i.e. whether the test-bed focus on providing tools to model many domains or simply concentrates on a single representative domain. We will be referring differently to both types of test-beds as *general* or *domain-independent* and *domain-specific* respectively.
- *Development facilities* that may include:
  - a development language;
  - a pre-specified agent architecture;
  - support for agent communication;
  - graphic tools to visualise and monitor the execution of the system; and
  - debugging tools to analyse the behaviour of the system and detect deviating, unexpected behaviours.

  Test-beds that do not commit to any particular agent architecture and do not provide any development language for building agents are said to remain *architecturally-neutral* since they may include agents possibly written in different languages and run by different architectures.

- *Evaluation facilities* that permit to collect data so that it can be displayed and analysed to study and understand both each agent's performance and the performance of the system as a whole. They are typically tied to debugging and graphical facilities. Notice that the capabilities of scenario generation and repeatability are fundamental in evaluating. For a complete evaluation, researchers will require to generate and repeatedly evaluate carefully constructed scenarios that highlight certain problems and randomly generated scenarios that help test average performance. Undoubtedly the most important part of evaluation is the analysis of results, but agent test-beds have not historically supported analysis tools.

Therefore, the completeness of a test-bed (from a researcher's point of view) depends on the tools, if any, provided within the facilities' types identified above.

Once presented what a test-bed is, in what follows we discuss how a test-bed can be used by surveying a collection of test-beds employed for research in DAI and multi-agent systems. Along our discussion we shall attempt at identifying the types of facilities provided by each test-bed.

DVMT [Lesser and Corkill, 1983] is the oldest, and probably largest and most-cited DAI test-bed. It is a domain-specific test-bed that simulates a network of vehicle monitoring agents responsible of analysing acoustically sensed data to identify, locate, and track patterns of vehicles moving through a two-dimensional space. Each agent is equipped with problem-solving capabilities an is associated with a sensor or sensors. Each agent is responsible for a specific area and has as its goal to recognise and eliminate erroneous sensor data as it integrates the correct data into an answer map. At this aim agents can engage in several types of cooperative interactions.

DVMT agents use the Generic Blackboard Architecture (GBB) as their base to construct blackboard-based agents, but allow for researchers to define their own communication protocols. DVMT provides tracing facilities that permit the global and local visualisation of the system, and even it is rarely one of the few test-beds providing automated tools for analysing the performance of the system.

Tile-world [Pollack and Ringuette, 1990a] is a test-bed whose agents are simulated robots that push tiles around to fill holes in a rectangular two-dimensional grid that they occupy composed of tiles, some obstacles, and some holes. The experiment designer configures the world setting up the parameters that the subsequent simulation employs to make appear holes, obstacles and tiles. The agent's goal is to maximise his score by filling holes with tiles. Its evaluation facilities allow to track and measure each agent's performance during a trial. Although no development facilities are provided, Tile-world is distributed with an embedded, configurable IRMA agent architecture.

Tile-world is intended to support controlled experiments with agent architectures situated in dynamic and unpredictable environments.

MICE [Montgomery et al., 1992] is another grid-oriented test-bed intended to host research into coordinating the problem-solving behaviour of multiple autonomous agents. The basic layout of MICE consists of various agents and a grid. The experimenter only defines an agent's sensing and effecting capabilities and the effect of actions taken simultaneously by the agents. Mice is domain-specific, hence it allows to model many domains, and architecturally neutral. MICE might be viewed more as a framework for building test-beds rather than a simulator in and of itself (in fact versions of Tile-world and Phoenix have been built with this platform).

Phoenix [Cohen et al., 1989] is a test-bed for implementing and testing multiple autonomous agents in a complex fire-fighting scenario. The Phoenix environment employs a fairly complex fire simulator based on weather conditions, Yellowstone National Park's physical features, and the features of the terrain (particularly burn rates). The Phoenix test-bed allows multiple agents (fire-bosses and bulldozers) to operate semi-autonomously in the highly dynamic environments generated by the simulator. For this purpose, it includes an experiment-running facility that offers a language for specifying scripts for changes in weather, fires starting, and other events. Phoenix uses Lisp as its implementation language and includes a flexible high-level agent architecture. Due to the environment's complexity and size, Phoenix offers extensive and detailed debugging and graphical visualisation tools that can be employed for monitoring agents' behaviour, producing data files that can be read by data-manipulation and statistical packages.

Truckworld [Nguyen et al., 1993] is a test-bed designed to test theories of reactive execution and to research in theories of reasoning about dynamic and uncertain worlds. An agent is a truck with two arms, two cargo bays, several sensors, and various other components that operates in worlds consisting of roads that connect locations populated with objects (f.i. fuel drums, tire chains, vending machines, etc.). The simulator is in charge of periodically generating exogenous events (f.i. rain storms). Each agent directly communicates with the simulator to perform actions and get sensor reports via communication devices provided by the simulator itself.

Truckworld is in between general test-beds such as Tile-world and domain-specific test-beds such as Phoenix.

MACE [Gasser et al., 1987] was one of the first general (domain-independent) test-beds for modelling multi-agent systems, as well as one of the first truly concurrent distributed object systems. As a distinguishing feature, MACE exploits the use of agents at all stages of system construction, user interaction, and management of experiments, as well as the basis for the modelled system itself. For this purpose, MACE provides a language for describing and instantiating agents, as well as support for a problem-solving architecture. Hence the test-bed and the experiment are an integrated multi-agent organisation for interacting with the experimenter and for testing hypotheses about the structure of organisations of artificial agents.

Another major contribution of MACE is the inclusion of explicit social modelling concepts drawn from sociological theory.

First, it employs the notion of recursive composition of agents so that a group can itself be treated as an agent with distributed internal structure. In this way, agents, groups and groups of groups have all agency.

Secondly, it incorporates the notion of *social world* proposed by symbolic interactionists[6]. Social worlds are operationalised in MACE as knowledge-based agent boundaries. Then each agent contains a set of acquaintances that define its boundaries of communication and interaction, and hence the social structure.

Lastly, social knowledge and social structure is founded in MACE on the modelling of other agents. MACE includes facilities to model other agents' features (such as roles, goals, skills, etc.) in order to structure individual interactions, and so to establish social structure defined as patterns of interaction. Observe that modelling others has become one of the main issues of MAS and DAI research.

Tæms [O'Hare and Jennings, 1996](Task Analysis, Environment Modeling, and Simulation) is a powerful framework with which to model complex computational task environments for exploring issues concerning coordination, scheduling, and organisation. The framework allows to both analyse and quantitatively simulate the behaviour of single or multi-agent systems with respect to interesting characteristics of computational task environments of which they are part. At this aim, it allows to build models of task environments at multiple levels of abstraction. The central idea motivating Tæms is that the design of coordination mechanisms cannot solely rely on the principled construction of agents alone, but also on the structure and other features of the task environment (f.i. the presence of uncertainty and high variance in a structure). In turn, this structure should be also used as the central guide to the design of coordination mechanisms, and so must be part of any theory of coordination.

Tæms is endowed with a language for building general models of a large variety of different domains that can be either simulated or mathematically analysed. Thus it exists as both a language for stating hypotheses or theories and as a system for simulation. Since it focuses on domain modelling, it has opted for remaining architecturally neutral as to agents, and so no development language or agent architecture is provided. Complementarily, Tæms is also endowed with extensible trace facilities, with facilities for the graphical display of domain scenarios (task structures) and with support for data collection and analysis.

AGenDA [Fis, 1996] is a test-bed for the design of DAI applications consisting of two different levels:

- *Architectural level.* It involves a methodology for designing agents founded on the InteRRaP agent architecture [Muller and Pischel, 1994]. General templates for agents are provided that have to be filled by the designer of a DAI system with the domain-specific instantiation.

- *System level.* It is covered by the MAGSY system [Fischer, 1993]. MAGSY provides a frame-based knowledge representation formalism and a set of general-purpose inference mechanisms. Furthermore, it provides facilities

[6]Individual people negotiate their lives by operating within social worlds which constrain both what they need to know and with whom they interact.

supporting the construction, visualisation, evaluation and debugging of DAI scenarios; from the lowest level of communication up to higher levels of complex protocols.

AGenDA is reported to have successfully hosted the development of two actual applications, namely FORKS, an automated loading dock where forklift robots load and unload trucks in a coordinated way, and MARS, which models an agent society consisting of transportation companies and their trucks that have to carry out transportation orders.

CooperA [Som, 1996](Cooperating Agents) is a generic test-bed for distributed knowledge-based applications that arises from the need to meet in the most effective way the requirements of a specific real-life, critical application. CooperA permits to investigate cooperative and coordinated problem solving by sets of coarse-grained knowledge-based agents. As development facilities, it uses a general language (Knowledge Craft's CRL-Lisp, a Lisp extended with a frame knowledge representation language) and a well-defined agent structure. Furthermore, it provides translation mechanisms and semantic mapping dictionaries for heterogeneous agents and a uniform message structuring protocol for communication. A special user interface agent provides a number of facilities such as tracing of executions, continuous updating of the status of agents' tasks, and others.

CADDIE [Farhoodi et al., 1991](Control And Direction of Distributed Artificial Intelligent Agents) is a test-bed for evaluating organisational structures and procedures. Instead of simulating specific domains, CADDIE allows to build particular scenarios in a chosen domain of interest. It also provides an architecture for describing agents and organisations of agents and their communications.

EOS [Dor, 1994] is a test-bed devoted to founding the study and development of computational models of sociological theories. Simulations run by EOS intend to obtain a link between cognitive and environmental factors and the emergence of hierarchical groupings among small sets of agents and development tools based on general languages such as C++ and Prolog.

AuctionBot [Wurman et al., 1998] Though not claimed as a test-bed, AuctionBot has been used as a test-bed by being extensively used for investigating both computational market mechanisms and agent strategies. On the one hand, its flexible specification of auctions in terms of orthogonal parameters makes it a useful tool for agent researchers exploring the design space of auction mechanisms. On the other hand, it provides full support for trading agents implementing the AuctionBot API, a communication specification that developers follow to implement software that commmunicates with the AuctionBot server over TCP. Currently, AuctionBot supplies C++ and Java implementations of the API for developers. To the best of our knowledge, though neither evaluation facilities nor visualisation tools are provided, AuctionBot supports the text-based monitoring of the trading activity.

In Chapter 6 we present **FM**, a test-bed for trading agents in auction-based markets. The framework, conceived and implemented as an extension of FM96.5 (a Java-based version of the Fishmarket auction house), allows to define trading scenarios based on several types of auctions (Dutch, English, First-price Sealed-bid and Vickrey. FM provides the environment wherein agent designers can perform *controlled experimentation* in such a way that a multitude of experimental market scenarios – that we regard as *tournament* scenarios due to the competitive nature of the domain – of varying degrees of realism and complexity can be specified, activated, and recorded; and trading (buyer and seller) heterogeneous (human and software) agents compared, tuned and evaluated. Such competitive situations are argued to constitute convenient problem domains in which to study issues related with agent architectures in general and auction strategies in particular.

To the best of our knowledge, AuctionBot and FM are the only test-beds for trading agents in auction-based tournaments. Apart from the originality of the test-bed domain, there are several features that differentiate it from the rest of test-beds. Firstly, FM is open, likewise the marketplace itself and unlike most test-beds which are closed systems, and so agents owned by different parties can take part in a same experiment. Secondly, FM allows to handle eventual scalability problems by allowing experimenters to distribute their agents across a network, and allowing remote agents to participate in a tournament by employing a minimum runtime environment. Lastly, FM96.5 is wrapped into FM being in fact one of its tournament modes.

We have observed that in the last few years there has not been much activity regarding the development of DAI test-beds. This fact is mainly motivated by the evidence that the development of DAI experimental platforms manifests as a long, tedious process characterised by the need of an enormous development effort. Although there is an increasing number of agent development environments, they are fundamentally conceived to support the construction of agent-based applications.

Chapter 3

A Formal Specification of Electronic Institutions

In this chapter we argue that open multi-agent systems can be effectively designed and implemented as institutionalised electronic organisations (*electronic institutions*) composed of a vast amount of heterogeneous (human and software) agents playing different roles and interacting by means of illocutions. We take the view that the design and development of electronic institutions must be guided by a principled methodology. Along this direction, we advocate for the presence of an underlying formal method that underpins the use of structured design techniques and formal analysis, facilitating development, composition and reuse. For this purpose we propose a specification formalism for electronic institutions that founds their design, analysis and development.

3.1 Organisations, Institutions and Electronic Institutions

In Chapter 1 we identified as our main goal the design and implementation of multi-agent systems. We also argued in favour of borrowing and incorporating social concepts and structures that have proven valuable to articulate human societies in order to reduce our endeavour. In what follows, the reasons why we resort to social notions are discussed.

Firstly, we start out considering organisations and institutions in the context of human societies.

Following the rational view of organisations provided by [Etzioni, 1964] "organisations are social units (or human groupings) deliberately constructed and reconstructed to seek specific goals". Organisations include political, economic, social and educational bodies (f.i. political parties, trade unions, clubs, universities). In [Scott, 1992] *goal specifity* and *formalisation* are identified as the two main characteristics of organisations, meaning by formalisation the attempt to standardise and regulate the behaviour of the roles interacting within an organisation. Meaning by roles standardised patterns of behaviour required of all agents playing a part in a given functional relationship in the context of an organisation.

Organisations must conform to the rules of institutions (such as the Constitution, the common law, etc.) in order to receive legitimacy and support. Institutions [North, 1990] represent the rules of the game in a society, including any (formal or informal) form of constraint that human beings devise to shape human interaction. They are the framework within which human interaction takes place, defining what individuals are forbidden and permitted and under what conditions. Furthermore, institutions are responsible for ascertaining violations and the severity of the punishment to be enacted. They are created and altered by human beings.

While organisations are concerned with achieving efficient functional structures, institutions are concerned with realising normative environments. From this follows that institutionalised organisations are organisations whose activities are regulated by means of some normative environment.

When considering open multi-agent systems, adopting an organisation-centered perspective may eventually lead us to functionally structure the system in the most effective way. And yet this is not enough. External agents are assumed to be neither benevolent nor cooperative. Their eventual faulty, deviating or malicious behaviour may jeopardise the system. Thus there is also the need for deploying a normative environment similar to those provided by human institutions. Therefore, we uphold that open multi-agent systems can be effectively designed and implemented as *institutionalised agent organizations* —henceforth referred to as *electronic institutions* or *e-institutions* for shorter.

Since a distinguishing feature of institutions is the clear distinction drawn between the rules and the players, in this chapter we focus on the macro-level (societal) aspects referring to the rules and structure of electronic institutions, instead of the micro-level (internal) aspects of agents. Due to the high complexity of this task and to the critical nature of open multi-agent systems applications, we advocate for adopting a principled, engineering approach founded on a formal specification of electronic institutions, as a follow-up of [Rodríguez-Aguilar et al., 2000,Sierra and Noriega, 1998], that founds their design, analysis and subsequent development. As a result, we obtain a formal, graphical specification language for electronic institutions.

Notice that the kind of electronic institutions that we will be considering are populated by a vast amount of heterogeneous (human and software) agents playing different roles and interacting by means of illocutions [Searle, 1969]. In other words, we make the assumption that any interacting activity taking place within our electronic institutions is purely dialogical.

The rest of the chapter is organised as follows. In Section 3.2 we argue on the need for a formal method that underpins the specification, analysis and development of electronic institutions. Next, in Section 3.3 we provide a mathematically sound, unambiguous, abstract definition of electronic institutions. Section 3.4

analyses how the rules contained in the structural specification affect the dynamics of electronic institutions once populated by agents. In order to illustrate how to specify in practice electronic institutions, we offer in Section 3.5 two examples corresponding to the institutions introduced in Chapter 1. In Section 3.6 we summarise and highlight the major benefits arising from our contribution, and discuss the open issues that can be faced taking our proposal as the starting point.

3.2 A Formal Specification Approach

Formal methods in software building comprise two main activities, namely *formal specification* and *verified design*, that is the precise specification of the behaviour of a piece of software, so that such software can be subsequently produced, and the proof of whether the implementation complies with the specification. Thus, the role of any formal method is to provide a clear and precise description of *what* a system is supposed to do, rather than a formulation of *how* it operates [Diller, 1990]. The presence of an underlying formal model will support the use of structured design techniques and formal analysis, facilitating development, composition and reuse.

In this section we take such formal approach with the purpose of specifying electronic institutions. Nonetheless, instead of using other formal techniques such a Z, CSP, Petri Nets or others, our formal specification will be based on a purposely devised specification language enabled to produce both visual and textual specifications of electronic institutions. In other words, such specification language shall serve to produce sound an unambiguous specifications of the *rules* of the game of an electronic institution. It is apparent that at a further stage the specification language can be extended in order to allow for the specification of the *players* of the game. Such extension is expected to support the specification of micro (internal) aspects of agents, i.e. their architectures and logics. As a result, it would be possible the complete specification (infrastructure and agents) of an electronic institution.

Why a new language? Let us refer back to our discussion about the subject in Chapter 1. There we argued that although using a general purpose-specification languages such as Z, π-calculus, or CSP represents a straightforward solution, these are limited by their lack of expressiveness to cover all agent issues. For instance, Z has been reported to be inappropriate for modelling interactions, being CSP or Petri Nets more appropriate for this purpose. And yet Petri Nets have been recently criticized for the combinatorial size for actual protocols and the ultra-powerful simulation tools needed. Therefore, we have opted for inferring the features of the language required from the requirements obtained when analysing electronic institutions. As a future step, we might consider whether there exists a natural mapping to a well-know formalism that helps us to carry out formal reasoning.

For the moment, in this work we solely focus on the specification of the rules of electronic institutions. As a first step, we identify the core notions on which our current conception of electronic institution, and so our specification language, is to be founded:

- *Agents and Roles.* Agents are the players in an electronic institution, interacting by the exchange of illocutions, whereas roles are defined as standardised patterns of behaviour. The identification and regulation of roles is considered as part of the formalisation process of any organisation [Scott, 1992]. Any agent within an electronic institution is required to adopt some role(s). A major advantage of using roles is that they can be updated without having to update the actions for every agent on an individual basis. Recently, the concept of role is becoming increasingly studied by software engineering researchers [Riehle and Gross, 1998, Jacobson et al., 1996], and even more recently by researchers in the agents' community [Odell et al., 2000,Becht et al., 1999,Kendall, 1998,Wooldridge et al., 1999,Buhr et al., 1997,Belakhdar and Ayel, 1996]. Hereafter we will differentiate between institutional and non-institutional (external) roles as well as institutional and non-institutional (external) agents. Whereas institutional roles are those enacted to achieve and guarantee institutional rules, non-institutional roles are those requested to conform to institutional rules. From this follows that institutional agents are those that adopt institutional roles, while non-institutional agents are those playing non-institutional roles.

- *Dialogical framework.* Some aspects of an institution such as the objects of the world and the language employed for communicating are fixed, constituting the context or framework of interaction amongst agents. In a dialogical institution, agents interact trough speech acts (illocutions). Institutions establish the acceptable speech acts by defining the ontology[1] and the common language for communication and knowledge representation which are bundled in what we call dialogical framework. By sharing a dialogical framework, we enable heterogeneous agents to exchange knowledge with other agents.

- *Scene.* Interactions between agents are articulated through agent group meetings, which we call *scenes*, with a well-defined communication protocol. We consider the protocol of a scene to be the specification of the possible dialogues agents may have. Notice however that the communication protocol defining the possible interactions within a scene is role-based instead of agent-based. In other words, a scene defines a role-based framework of interaction for agents.

- *Performative structure.* Scenes can be connected, composing a network of scenes, the so-called performative structure, which captures the existing relationships among scenes. The specification of a peformative structure

[1]Here we adhere to the definition of Alberts in [Alberts, 1993]: *An ontology for a body of knowledge concerning a particular task or domain describes a taxonomy of concepts for that task or domain that define the semantic interpretation of the knowledge.*

contains a description of how agents can legally move from scene to scene by defining both the pre-conditions to join in and leave scenes. Satisfying such conditions will fundamentally depend on the roles allowed to be played by each agent and his acquired commitments through former utterances. The execution of a performative structure is to contain the multiple, simultaneous, ongoing activities, represented by scenes, along with the agents participating in each activity. Agents within a performative structure may be possibly participating in different scenes, with different roles, and at the same time.

- *Normative Rules.* Agent actions in the context of an institution have consequences, usually in the shape of compromises which impose obligations or restrictions on dialogic actions of agents in the scenes wherein they are acting or will be acting in the future. The purpose of normative rules is to affect the behaviour of agents by imposing obligations or prohibitions. Notice that institutinal agents are committed to undertake the required actions so that to ensure that non-insitutional agents abide by institutional rules.

Next, in the rest of the section, we offer a detailed analysis of the notions introduced above that helps us gradually construct a formal specification of an electronic institution. Firstly, in Section 3.3 we concentrate on specifying an electronic institution at the *structural* level. Our approach is analogous to the definition of directed graphs, non-deterministic finite automata or Petri nets. Formally, they are defined as many-tuples, but usually they are graphically represented by drawings. Once specified an electronic institution, in Section 3.4 we propose an execution model that explains the dynamics of agents within an institution.

3.3 Specifying Electronic Institutions

First of all, and for notational purposes, we introduce some definitions that will apply henceforth. Let $Agents = \{a_1, \ldots, a_n\}$ be a finite set of agent identifiers, and $Roles = \{r_1, \ldots, r_m\}$ a finite set of role identifiers respectively. At the specification level we regard agents and roles simply as symbols and type names. Then by $a_i : r_j$ we mean that agent a_i is of type r_j. Moreover, we define the sets $V_A = \{x_1, \ldots, x_p\}$ and $V_R = \{\rho_1, \ldots, \rho_q\}$ as a finite set of agent variables and a finite set of role variables respectively.

3.3.1 Roles

We have already identified the notion of role as central in the specification of electronic institutions. Roles allow us to abstract from the individuals, the agents, that get involved in an institution's activities. In order to take part in an electronic institution, an agent is obliged to adopt some role(s). Thereafter an agent playing a given role must conform to the pattern of behaviour attached to that particular role. Therefore, all agents adopting a very same role are guaranteed to have the same rights, duties and opportunities. Notice that we can think of roles as agent types. An agent thus have associated a finite set of dialogic actions. Such actions are intended to represent the capabilities of the role. For instance, an agent playing the buyer role is capable of submitting bids, another agent playing the auctioneer role can offer goods at auction, and the agent acting as the boss of the market is authorised to declare the closing of the market at any time.

There are several issues concerning the agent/role and role/role relationships. First, we must establish how agents are authorised to perform roles. As an agent may possibly play several roles at the same time, role/role associations standing for conflict of interests must be defined with the purpose of protecting the institution against an agent's malicious behaviour. For instance, though an agent might act as a bank teller and as a costumer, we should not allow it to play both roles at the same time.

Thus, in general, the management of agent/role and role/role relationships becomes a fundamental issue for electronic institutions. *Role-based Access Control Models* (RBAC Models) [Sandhu et al., 1996] developed in the computer security arena offer well-founded mechanisms for handling both types of relationships. Fundamentally, RBAC has been successfully employed for managing the security administration of organisations [Barkley et al., 1997,Barkley, 1997]. But more interestingly, RBAC has been accurately formalised [Ferraiolo et al., 1999, Gavrila and Barkley, 1998, Ferraiolo et al., 1995]. In order to solve the agent/role and role/role relationships, we borrow and adapt the formalisation of the rules for RBAC offered by the NIST RBAC model [Gavrila and Barkley, 1998,Ferraiolo et al., 1995,Ferraiolo et al., 1999], which extends the basic RBAC model by adding role hierarchies, cardinality, and conflict of interests relationships.

Building upon RBAC, in what follows we develop our own model based on role hierarchies, cardinality and conflict of interests relationships. We will employ our model for two basic purposes:

- to grant role permissions to agents when joining an institution; and
- to regulate agents' role changes through the several activities taking place within an institution.

Although RBAC manages three types of associations, namely associations between users (agents) and roles, associations between roles, and associations between roles and permissions [Ferraiolo et al., 1999], in this section we solely concentrate on the role/role associations and complementarily, further on in Section 3.4.1, we deal with agent/role associations when considering the dynamics of electronic institutions.

When analysing human institutions we observe that the roles played by humans are hierarchically organised. Consider for instance the roles played by the medical staff in a hospital. The head of any department must be also a doctor. Thus, for instance the head of orthopaedics is capable of diagnosing any patient likewise his subordinates. However, only the head is entitled to either approve or disapprove some tests and surgeries, while his subordinates are only allowed to perform them. Analogously, any doctor, no matter which speciality, is capable of testing a patient's blood pressure likewise any nurse. However, a doctor is capable of diagnosing a patient. In general, roles further up in any given hierarchy subsume the capabilities of related roles further down in the hierarchy. Figure 3.1[2] depicts an example of a hierarchy involving medical staff roles.

There are many ways of constructing a role hierarchy to express different types of relationships among roles. Here we consider that roles are organised as a partially ordered set (poset), represented as a pair $\mathcal{R} = \langle Roles, \succeq \rangle$, reflecting a role hierarchy. The relation $\succeq \subseteq Roles \times Roles$ is reflexive, antisymmetric, and transitive. Then if $r \succeq r'$ holds we say that r subsumes r', or alternatively, that r is a super-type of r'. We interpret this relationship as follows: an agent playing role r is also enabled to play role r'.

We also realise that there are conflicting roles within an institution. For instance, in the fish market the *boss* and *buyer* roles are mutually exclusive in the sense that no one can *ever* act as the boss of the market and as a buyer. In a bank, the *teller* and *auditor* roles are also considered as mutually exclusive. We define a policy of static separation of duty (SSD) to mean that roles specified as mutually exclusive cannot be both authorised to an agent.

We represent the static separation of duties as the relation $ssd \subseteq Roles \times Roles$. A pair $(r, r') \in ssd$ denotes that r, r' cannot be authorised to the very same agent. Observe that the static separation of duties will be verified when assigning roles to an agent entering the institution. An agent can apply for several roles when entering an insitution, though only the non-conflicting roles can be assigned.

Summarising, the adoption of a policy of separation of duties will allow us to express constraints on the role/role associations indicating conflicts of interests.

Finally, we explicitly state the requirements identified so far as necessary for managing the role/role relationships:

(i) The static separation of duties relation is symmetric. Formally, for all $r_i, r_j \in Roles \quad (r_i, r_j) \in ssd \Rightarrow (r_j, r_i) \in ssd$.

(ii) If a role subsumes another role, and that role is in static separation of duties with a third role, then the first role is in static separation of duties with the third one. Formally, for all $r, r_i, r_j \in Roles, r \succeq r_i, (r_i, r_j) \in ssd \Rightarrow (r, r_j) \in ssd$.

(iii) If a role subsumes another role, then the two roles cannot be in static separation of duties. Formally, $r_i \succeq r_j \Rightarrow (r_i, r_j) \notin ssd$.

Observe that from the last requirements the following properties can be inferred:

(i) For all $r \in Roles \Rightarrow (r, r) \notin ssd$ since the $\succeq$ relation is reflexive.

[2]The example has been borrowed from [Ferraiolo et al., 1995].

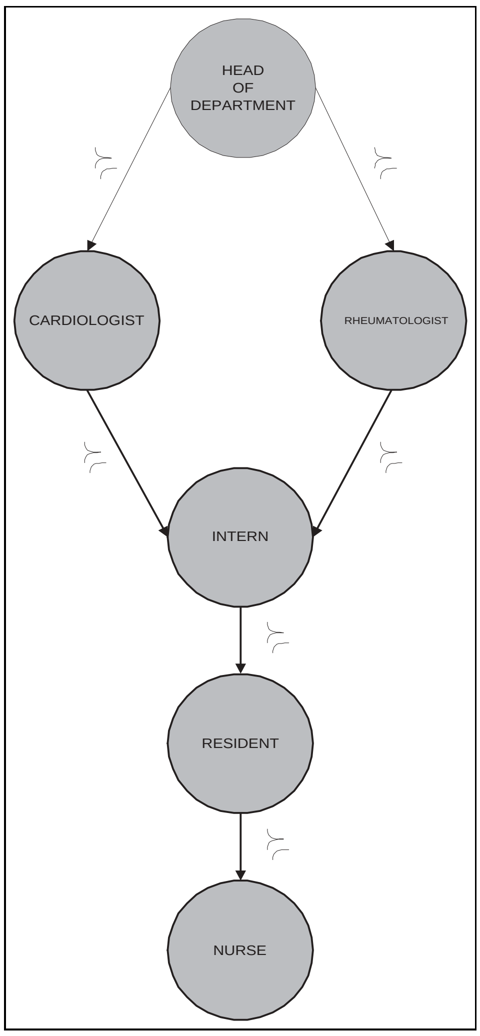


Figure 3.1: Example of Role Hierarchy.

(ii) If $(r_i, r_j) \in ssd$ then for all $r \succeq r_i$ and for all $s \succeq r_j \Rightarrow (r, s) \in ssd$.

(iii) If exists r such that $r \succeq r_i$ and $r \succeq r_j \Rightarrow (r_i, r_j) \notin ssd$.

3.3.2 Dialogical Framework

In the most general case, each agent immersed in a multi-agent environment is endowed with its own inner language and ontology.

In order to allow agents to successfully interact with other agents we must address the fundamental issue of putting their languages and ontologies in relation. For this purpose, we propose that agents share when communicating the so-called *dialogical framework* [Noriega and Sierra, 1996], composed of a communication language, a representation language for domain content and an ontology [Gruber, 1993a]. By sharing a dialogical framework, we enable heterogeneous agents to exchange knowledge with other agents.

Definition 3.3.1. We define a *dialogical framework* as a tuple $DF = \langle O, L, I, CL, Time \rangle$ where

- O stands for an ontology;
- L stands for a representation language for domain content;
- I is the set of illocutionary particles;
- CL is the (agent) communication language;
- $Time$ is a discrete and totally ordered set of instants.

Within a dialogical framework the representation language (KIF [Genesereth and Fikes, 1992], first-order logic, etc.) allows for the encoding of the knowledge to be exchanged among agents using the vocabulary offered by the O ontology. The propositions built with the aid of L, the "inner" language, are embedded into an "outer language", CL, which expresses the intentions of the utterance by means of the illocutionary particles in I. We take this approach in accord to speech act theory [Searle, 1969], which postulates that utterances are not simply propositions that are true or false, but attempts on the part of the speaker that succeed or fail.

We consider that CL *expressions* are constructed as formula of the type $\iota(\alpha_i : \rho_i, \alpha_j : \rho_j, \varphi, \tau)$ where $\iota \in I, \alpha_i$ and α_j are terms which can be either agent variables or agent identifiers[3], ρ_i and ρ_j are terms which can be either role variables or role identifiers, $\varphi \in L$ and τ is a term which can be either a time variable or a value in $Time$. We say that a CL expression is an *illocution* when α_i, α_j are agent identifiers, ρ_i and ρ_j are role identifiers and τ is a value in $Time$ time-stamping the illocution. We say that a CL expression is an *illocution scheme* when some of the terms corresponds to a variable. This distinction will be valuable when specifying scenes in the following section.

[3]We shall also employ the particle *all* as a value of α_j to specify a broadcast of an illocution to all agents participating in a scene playing a particular role.

We shall employ question marks to differentiate variables in CL expressions, including the expression corresponding to the contents, from other terms. During the execution of a scene, variables in CL expressions will be bound to concrete values. Thus we need to differentiate free variables from bound variables. Henceforth by $?t$ we shall denote a free variable t whereas by $!t$ we denote the value bound to the t variable.

Considering the examples introduced in Chapter 1 (for both cases, we take first-order logic as the representation language for domain content, as well as the *conferences* ontology for the conference centre environment, and the *e-auctions* vocabulary for the fish market). We have the following examples corresponding to CL expressions:

- $request(?x_i : pra, ?x_j : pra, appointment(?t), ?t')$ is an illocution scheme.
- $request(KQLAT : pra, marc : pra, appointment(tomorrow), 5)$ is an illocution that instantiates the schema above. We interpret this CL expression as a request sent by agent $KQLAT$ playing the *pra* (personal representative assistant) role to agent *marc* playing the same pra role for arranging an appointment to meet *tomorrow*. Furthermore, the time value indicates that the request was uttered by agent $KQLAT$ at time 5.
- $request(auct\#01 : auctioneer, KQLAT : buyer, bid(1\$), 6)$ is an illocution uttered by the auctioneer agent $auct\#01$ to ask buyer $KQLAT$ whether it wants to submit a bid at 1$.
- $refuse(KQLAT : buyer, auct\#01 : auctioneer, bid(1\$), 7)$ is another illocution, now sent by buyer $KQLAT$ to the auctioneer agent $auct\#01$ refusing the proposal to bid at 1$.

Finally, we would like to stress the importance of the dialogical framework as the component containing the ontologic elements on the basis of which any agent interaction can be specified as illustrated next when introducing the notion of scene. Thus, a dialogical framework must be regarded as a necessary ingredient to specify scenes.

3.3.3 Scene

Before formally defining a scene, we must precisely understand what a scene is. Recall that the whole activity within an electronic institution was described above as a composition of multiple, well-separated, and possibly concurrent, dialogic activities, each one involving different groups of agents playing different roles. For each activity, interactions between agents are articulated through agent group meetings, which we call *scenes*, that follow well-defined protocols. We consider the protocol of each scene to model the possible dialogical interactions between roles instead of agents. In other words, scene protocols are patterns of multi-role conversation. Scene protocols are state-based specifications characterising scenes as sequences of states.

A distinguishing feature of scenes is that they allow that agents (possibly) either join in or leave at some particular moments of an ongoing conversation.

The formal definition that follows is intended to contain all the elements that we will subsequently need to specify the dynamics, the execution of a scene.

Definition 3.3.2. Formally, a scene is a tuple:

$$S = \langle R, DF, W, w_0, W_f, (WA_r)_{r\in R}, (WE_r)_{r\in R}, \Theta, \lambda, min, Max\rangle$$

where

- R is the set of roles of the scene;
- DF is a dialogical framework;
- W is a finite, non-empty set of scene states;
- $w_0 \in W$ is the initial state;
- $W_f \subseteq W$ is the non-empty set of final states;
- $(WA_r)_{r\in R} \subseteq W$ is a family of non-empty sets such that WA_r stands for the set of access states for the role $r \in R$;
- $(WE_r)_{r\in R} \subseteq W$ is a family of non-empty sets such that WE_r stands for the set of exit states for the role $r \in R$;
- $\Theta \subseteq W \times W$ is a set of directed edges;
- $\lambda : \Theta \longrightarrow CL$ is a labelling function;
- $min, Max : R \longrightarrow I\!N$, $min(r)$ and $Max(r)$ are respectively the minimum and maximum number of agents that must and can play the role $r \in R$;

satisfying the requirements below:

(i) (W, Θ) is a connected graph.

(ii) Every non-final state $w \notin W_f$ is reachable from the initial state and connected to some final state $w' \in W_f$. More formally $\forall w \notin W_f \quad \exists w_0, \ldots, w_m \in W \mid (w_i, w_{i+1}) \in \Theta$ $(i = 0, \ldots, m-1)$ such that $w_m \in W_f$ and $\exists i \quad w = w_i$.

(iii) $W_f \subseteq (WE_r)_{r\in R}$

(iv) $\forall \theta \in \Theta, \lambda(\theta)$ is a CL expression representing an illocution scheme.

(v) $\forall r \in R \quad min(r) > 0 \Rightarrow w_o \in WA^r$.

(vi) $Max(r) \geq min(r) \geq 0$ $\forall r \in R$.

These variables will be bound to concrete values during the execution of the scene. For example, agent variables in an illocution scheme will be bound respectively to the identifier of the agent that has uttered the illocution and to the identifier of the agent who has received the illocution. Then at each moment the bindings of the variables will be the context of the scene execution. Notice that there is a major constraint applying to agent variables within a scene: there cannot be two different agent variables taking on the same value. This is to say that we will restrict agents to play a single role within each scene.

Next, we describe two examples of scenes extracted from the case studies presented in Chapter 1, that we subsequently employ to illustrate how designers can construct graphical specifications of scenes, and how such specifications, in turn, can be naturally translated into a textual, machine–readable format.

Case Studies

First, concerning the fish market, we have selected the scene representing the main activity within the marketplace: the auctioning of goods in the auction room. This scene is governed by the auctioneer making use of the downward bidding protocol (DBP) that next we state explicitly:

[**Step 1**] The auctioneer chooses a good out of a lot of goods that is sorted according to the order in which sellers deliver their goods to the sellers' admitter.

[**Step 2**] With a chosen good, the auctioneer opens a *bidding round* by quoting offers downward from the good's starting price, previously fixed by the sellers' admitter, as long as these price quotations are above a *reserve price* set by the seller.

[**Step 3**] For each price called by the auctioneer, several situations might arise during the open round:

Multiple bids. Several buyers submit their bids at the current price. In this case, a collision comes about, the good is not sold to any buyer, and the auctioneer restarts the round at a higher price. Nevertheless, the auctioneer tracks whether a given number of successive collisions is reached, in order to avoid an infinite collision loop. This loop is broken by randomly selecting one buyer out of the set of colliding bidders.

One bid. Only one buyer submits a bid at the current price. The good is sold to this buyer if his credit can support his bid. Whenever there is an unsupported bid the round is restarted by the auctioneer at a higher price, the unsuccessful bidder is punished with a fine, and he is expelled out from the auction room unless such fine is paid off.

No bids. No buyer submits a bid at the current price. If the reserve price has not been reached yet, the auctioneer quotes a new price which is obtained by decreasing the current price according to a price step. If the reserve price is reached, the auctioneer declares the good *withdrawn* and closes the round.

[**Step 4**] The first three steps repeat until there are no more goods left.

Secondly, concerning the conference centre environment we have selected the activity in which two *attenders* engage to negotiate upon a set of topics for discussing in a future, eventual appointment, the so-called *appointment proposal* scene [Plaza et al., 1999]. In fact, the responsibility of participating in the activity is delegated to the attenders' *personal representative agents*(PRA). The scene occurs as follows:

[**Step 1**] One PRA, the proposer, initiates the exchange by sending an appointment proposal to another PRA with a set of initial interest topics. This set is intended to be a subset of its owner's (the attender's) profile of interests.

[**Step 2**] The recipient evaluates the proposal to decide whether to accept it, refuse it, or responding with a counter-proposal containing a different set of topics.

[**Step 3**] When the proposer receives a counter-proposal, it evaluates it to decide whether accept it, refuse it, or responding back with another counter-proposal.

The negotiation process above loops until an agreement is reached or one of the PRAs decides to withdraw.

According to the definition of scene provided above, in Table 3.1, Table 3.2 and Table 3.3 we present the definition of the *appointment proposal* and *auction* scenes. In both cases we leave out the time value contained in any CL expression. The example demonstrates that, in practice, it is cumbersome to specify scenes in this manner. It would be more intuitive to construct graphical specifications of scenes.

Graphical Representation

The purpose of the formal definition above is to give a a sound definition of scene. However in practice scenes will be graphically specified. For instance, Figures 3.3 and 3.2 depict the graphical specifications of the scenes presented above. Here we adopt an analogous approach to the definition of directed graphs, non-deterministic finite automata, or Petri nets, which are all formally defined, but usually represented by drawings containing the elements identified by their respective formal definitions.

As to the *auction* scene (see Figure 3.3), we observe that the specification of the role set requires the participation of exactly one auctioneer(a) and, at least, two buyers(min(b) = 2), although up to ten buyers might be allowed to participate(Max(b) = 10). The graph depicts the states of the scene, along with the edges representing the legal transitions between scene states, and labelled with

$R = \langle pra \rangle$

$DF = \langle conferences, fol, \{commit, request, refuse\}, ACL, I\!N \rangle$

$W = \{w_0, w_1, w_2, w_{f1}, w_{f2}\}$

$W_f = \{w_{f1}, w_{f2}\}$

$WA_{pra} = \{w_0\}$

$WE_{pra} = \{w_{f1}, w_{f2}\}$

$\Theta = \{(w_0, w_1), (w_1, w_2), (w_2, w_1), (w_1, w_{f2}), (w_2, w_{f2}), (w_2, w_{f1}), (w_1, w_{f1})\}$

$\lambda((w_0, w_1)) = request(?x : pra, ?y : pra, app(?t))$

$\lambda((w_1, w_2)) = request(?y : pra, ?x : pra, app(?t'))$

$\lambda((w_2, w_1)) = request(?x : pra, ?y : pra, app(?t))$

$\lambda((w_1, w_{f2})) = commit(?y : pra, ?x : pra, app(!t))$

$\lambda((w_2, w_{f2})) = commit(?x : pra, ?y : pra, app(!t'))$

$\lambda((w_2, w_{f1})) = refuse(?x : pra, ?y : pra, app(!t'), reason(?r))$

$\lambda((w_1, w_{f1})) = refuse(?y : pra, ?x : pra, app(!t), reason(?r))$

$min(pra) = 2$

$Max(pra) = 2$

Table 3.1: Appointment Proposal Scene Specification

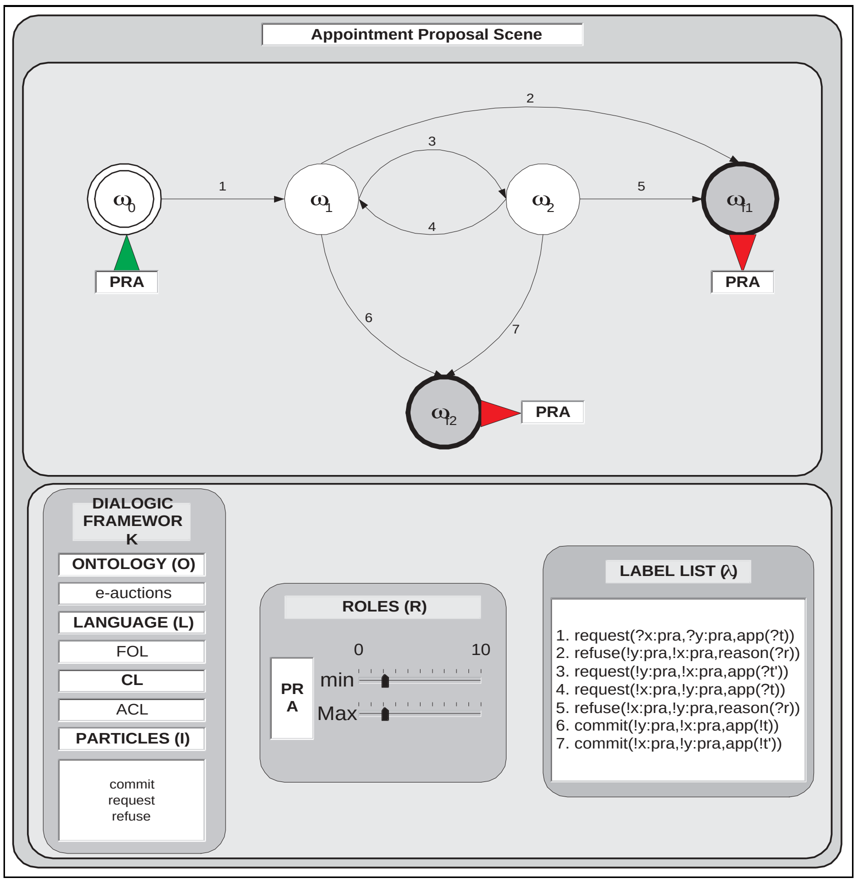

Appointment Proposal Scene
2
3
1
ω_0
ω_1
4
ω_2
5
ω_{f1}
PRA
PRA
6
7
ω_{f2}
PRA
DIALOGIC FRAMEWORK
ONTOLOGY (O)
e-auctions
LANGUAGE (L)
FOL
CL
ACL
PARTICLES (I)
commit
request
refuse
ROLES (R)
0
10
PRA
min
Max
LABEL LIST (λ)
1. request(?x:pra,?y:pra,app(?t))
2. refuse(!y:pra,!x:pra,reason(?r))
3. request(!y:pra,!x:pra,app(?t'))
4. request(!x:pra,!y:pra,app(?t))
5. refuse(!x:pra,!y:pra,reason(?r))
6. commit(!y:pra,!x:pra,app(!t))
7. commit(!x:pra,!y:pra,app(!t'))

Figure 3.2: Graphical Specification of the Appointment Proposal Scene

$R = \langle a(auctioneer), b(buyer)\rangle$

$DF = \langle e - auctions, fol, \{inform, commit, request, question, refuse\}, ACL, \mathbb{N}\rangle$

$W = \{w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, , w_9, w_{10}, w_{11}, w_{f1}\}$

$W_f = \{w_{f1}\}$

$WA_a = \{w_0\}$

$WA_b = \{w_0, w_1\}$

$WE_a = \{w_{f1}\}$

$WE_b = \{w_1, w_{f1}\}$

$\Theta = \{(w_0, w_1), (w_1, w_2), (w_1, w_{f1}), (w_2, w_3), (w_3, w_4), (w_4, w_5), (w_5, w_5), (w_5, w_6),$

$(w_5, w_7), (w_7, w_7), (w_6, w_1), (w_7, w_8), (w_8, w_1), (w_7, w_9), (w_9, w_5), (w_7, w_{10}),$

$(w_{10}, w_5), (w_7, w_{11}), (w_{11}, w_5), (w_{11}, w_{f1})\}$

$min(a) = 1$

$Max(a) = 1$

$min(b) = 2$

$Max(b) = 10$

Table 3.2: Auction Scene Specification (i)

$$\lambda((w_0, w_1)) = inform(?x : a, all : b, open\_auction(?n))$$

$$\lambda((w_1, w_{f1})) = request(!x : a, all : b, end\_auction(!n, ?reason))$$

$$\lambda((w_1, w_2)) = inform(!x : a, all : b, open\_round(?r))$$

$$\lambda((w_2, w_3)) = inform(!x : a, all : b, to\_sell(?good\_id))$$

$$\lambda((w_3, w_4)) = inform(!x : a, all : b, buyers(?b1, ..., ?bn))$$

$$\lambda((w_4, w_5)) = inform(!x : a, all : b, offer(!good\_id, ?price))$$

$$\lambda((w_5, w_5)) = inform(!x : a, all : b, offer(!good\_id, ?price))$$

$$\lambda((w_5, w_6)) = inform(!x : a, all : b, withdrawn(!good\_id, !price))$$

$$\lambda((w_5, w_7)) = commit(?y : b, !x : a, bid(!good\_id, !price))$$

$$\lambda((w_7, w_7)) = commit(?y : b, !x : a, bid(!good\_id, ?price))$$

$$\lambda((w_6, w_1)) = inform(!x : a, all : b, end\_round(!r))$$

$$\lambda((w_7, w_8)) = inform(!x : a, all : b, sold(!good\_id, !price, ?buyer\_id)$$

$$\lambda((w_8, w_1)) = inform(!x : a, all : b, end\_round(!r))$$

$$\lambda((w_7, w_9)) = inform(!x : a, all : b, collision(!price))$$

$$\lambda((w_9, w_5)) = inform(!x : a, all : b, offer(!good\_id, ?price))$$

$$\lambda((w_7, w_{10})) = inform(!x : a, all : b, sanction(?buyer\_id))$$

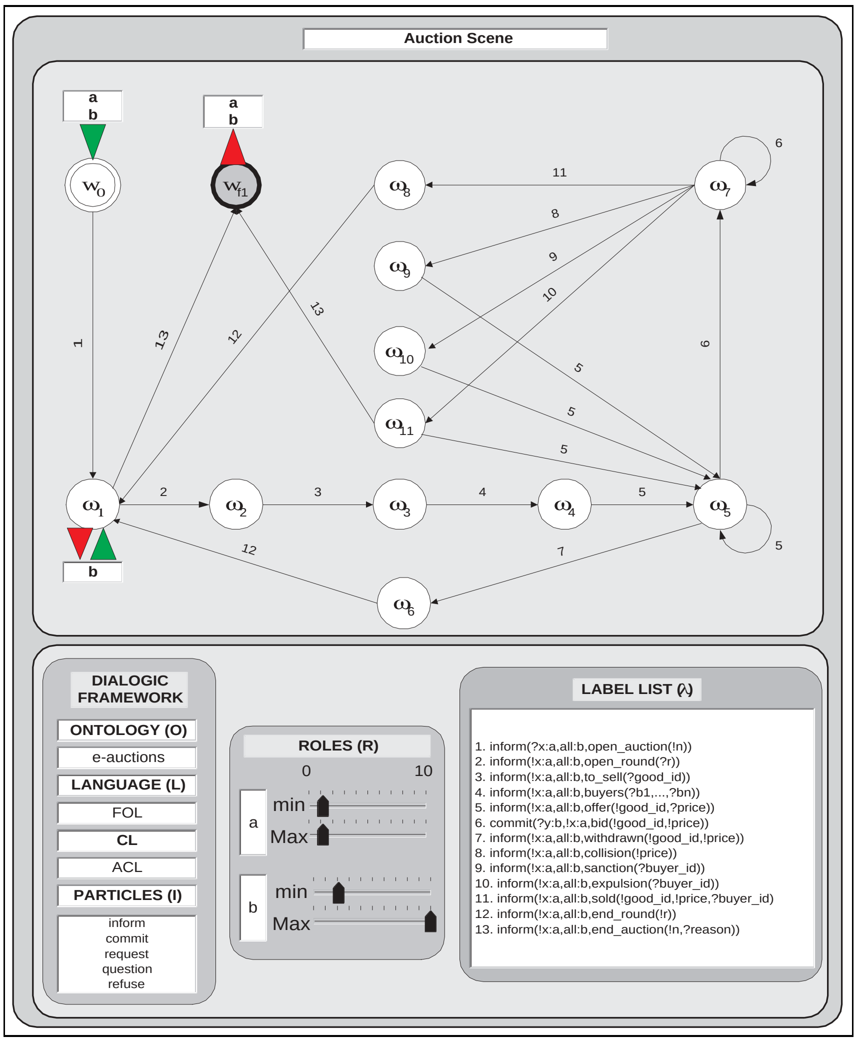
$$\lambda((w_{10}, w_5)) = inform(!x : a, all : b, offer(!good\_id, ?price))$$

$$\lambda((w_7, w_{11})) = inform(!x : a, all : b, expulsion(?buyer\_id))$$

$$\lambda((w_{11}, w_5)) = inform(!x : a, all : b, offer(!good\_id, ?price))$$

$$\lambda((w_{11}, w_{f1})) = inform(?x : a, all : b, end\_auction(!n, ?reason))$$

Table 3.3: Auction Scene Specification (ii)



1. inform(?x:a,all:b,open_auction(!n))
2. inform(!x:a,all:b,open_round(?r))
3. inform(!x:a,all:b,to_sell(?good_id))
4. inform(!x:a,all:b,buyers(?b1,...,?bn))
5. inform(!x:a,all:b,offer(!good_id,?price))
6. commit(?y:b,!x:a,bid(!good_id,!price))
7. inform(!x:a,all:b,withdrawn(!good_id,!price))
8. inform(!x:a,all:b,collision(!price))
9. inform(!x:a,all:b,sanction(?buyer_id))
10. inform(!x:a,all:b,expulsion(?buyer_id))
11. inform(!x:a,all:b,sold(!good_id,!price,?buyer_id)
12. inform(!x:a,all:b,end_round(!r))
13. inform(!x:a,all:b,end_auction(!n,?reason))

Figure 3.3: Graphical Specification of an Auction Scene

illocution schemes of the communication language of the dialogical framework. The information contents of such schemes is expressed in first-order logic (*FOL*) making use of the concepts in the *e-auctions* ontology. The type of performative is specified in the set I —which contains the types of performatives identified in [Cohen and Levesque, 1995]— listed in the *Particles* area.

Notice that some states are identified as access and exit states. Apart from the initial and final states, the w_1 state is labelled as an access and exit state for buyers —meaning that after either a collision, sanction or expulsion, new buyers might be admitted into the scene— while ω_3 is uniquely an access state.

Figure 3.2 shows the specification of the *appointment proposal* scene in the conference centre environment. This scene is restricted to a couple of *PRA* roles, but each PRA must be played by a different agent. Observe that the dialogical framework of the scene only differs in the use of a different ontology (*conferences*) with respect to the auction scene.

Graphical specifications are expected to be constructed by a scene designer. However, such specifications are deemed to contain multiple errors. A straightforward procedure for checking whether a graphical specification is well-formed can be readily obtained by conveniently adapting a basic breadth–first search algorithm to run over the graph (W, Θ) resulting from the specification.

Textual Representation

At this point there is the matter of generating a textual, machine-readable, representation of scenes. Such a textual representation is to be conveyed to agents intending to participate in a scene, so that they can interpret the rules of the game. Thus, once specified, and subsequently validated a scene, a textual representation in the chosen language can be constructed. Here we choose XML(eXtensible Markup Language) [Connolly, 1997] because it offers a unique combination of flexibility, simplicity, and readability by both humans and machines, and because we regard it as an excellent format for interchanging data.

Below we provide the XML-based specification of the *appointment proposal* scene depicted in Fig. 3.2, identified as *aps*, based on the document type definition enclosed in Appendix A.

```
<?xml version ="1.0" standalone = 'no'?>
<!DOCTYPE SCENE SYSTEM "http://www.iiia.csic.es/\~jar/AMEI/dtd/scene.dtd">
<SCENE ID="aps">
      <DESCRIPTION> Appointment Proposal Scene </DESCRIPTION>
      <DF>
         <ONTOLOGY> conferences </ONTOLOGY>
         <LANGUAGE> fol </LANGUAGE>
         <CL> acl </CL>
         <PERFORMATIVES>
            <PNAME> request  </PNAME>
            <PNAME> commit   </PNAME>
            <PNAME> refuse   </PNAME>
         </PERFORMATIVES>
```

```
</DF>
<ROLES>
   <ROLE>
      <ROLEID> PRA </ROLEID>
      <MIN> 2 </MIN>
      <MAX> 2 </MAX>
   </ROLE>
</ROLES>
<STATE ID= "w0" TYPE="initial">
   <ACCESS>
      <ROLEID> PRA </ROLEID>
   </ACCESS>
   <EDGELIST>
      <EDGE ID="t1" HREF="#w1">
         <ILLOCUTION>
            request(?x:pra,?y:pra,app(?t))
         </ILLOCUTION>
      </EDGE>
   </EDGELIST>
</STATE>
<STATE ID= "w1" TYPE="intermediate">
   <EDGELIST>
      <EDGE ID="t3" HREF="#w2">
         <ILLOCUTION>
            request(!y:pra,!x:pra,app(?t’))
         </ILLOCUTION>
      </EDGE>
      <EDGE ID="t2" HREF="#wf1">
         <ILLOCUTION>
            refuse(!y:pra,!x:pra,reason(?r))
         </ILLOCUTION>
      </EDGE>
      <EDGE ID="t6" HREF="#wf2">
         <ILLOCUTION>
            commit(!y:pra,!x:pra,app(!t))
         </ILLOCUTION>
      </EDGE>
   </EDGELIST>
</STATE>
<STATE ID= "w2" TYPE="intermediate">
   <EDGELIST>
      <EDGE ID= "t4" HREF="#w1">
         <ILLOCUTION>
            request(!x:pra,!y:pra,app(?t))
         </ILLOCUTION>
      </EDGE>
      <EDGE ID= "t5" HREF="#wf1">
         <ILLOCUTION>
            refuse(!x:pra,!y:pra,reason(?r))
         </ILLOCUTION>
```

```
            </EDGE>
            <EDGE ID= "t7" HREF="#wf2">
               <ILLOCUTION>
                  commit(!x:pra,!y:pra,app(!t'))
               </ILLOCUTION>
            </EDGE>
         </EDGELIST>
      </STATE>
      <STATE ID= "wf1" TYPE="final">
         <EXIT>
            <ROLEID> PRA </ROLEID>
         </EXIT>
      </STATE>
      <STATE ID= "wf2" TYPE="final">
         <EXIT>
            <ROLEID> PRA </ROLEID>
         </EXIT>
      </STATE>
</SCENE>
```

3.3.4 Performative Structure

In the context of an institution, certain relationships among scenes, activities, hold, and therefore they must be effectively captured. The notion of performative structure is the most complex and interesting of this formalism, since it models the relationships among scenes. Notice that although conversations (scenes) are currently admitted as the unit of communication between agents, limited work has been done concerning the modelling of the relationships among different scenes. This issue arises when these conversations are embedded in a broader context, such as, for instance, organisations and institutions. If this is the case, it does make sense to capture the relationships among scenes. Thus, while a scene models a particular multi-agent (dialogic) activity, more complex activities can be specified by establishing relationships among scenes that allow:

- to capture *causal dependencies* among scenes (f.i. a patient cannot undergo an operation without being previously diagnosed by a doctor);
- to define *synchronisation* mechanisms involving scenes (f.i. within an exchange house, we might require the presence of a certain number of traders when leaving the registration scene before allowing them to start off a negotiation scene);
- to establish *parallelism* mechanisms involving scenes (f.i. in an auction house, several auctions —possibly devoted to the auctioning of heterogeneous types of goods under the rules of various, different auction protocols— might be run at the same time);
- to define *choice points* that allow roles leaving a scene to choose their destination (f.i. an agent attending a conference is expected to opt for one

of various, simultaneous talks);

- to establish the *role flow policy* among scenes, i.e. which paths can be followed by the roles leaving a scene and which target scenes they can reach. In a conference centre, a speaker, after finishing off his talk, is permitted to leave the conference room and make it for another conference room to attend an ongoing talk. Notice that, for this particular case, the migration of this speaker also involves the adoption of another role.

In general, the activity represented by a performative structure can be depicted as a collection of multiple, concurrent scenes. Agents navigate from scene to scene constrained by the rules defining the relationships among scenes. Moreover, the very same agent can be possibly participating in multiple scenes at the same time. Hence, it is our purpose to propose a formal specification of performative structures expressive enough to facilitate the specification of such rules.

Structure

From a structural point of view, performative structures' specifications must be regarded as networks of scenes. How does this network affect agents' behaviour? At execution time, a performative structure becomes populated by agents that make it evolve whenever these comply with the rules encoded by the specification. Thus, groups of agents may jointly start new scene executions, activities, that we refer to as active scenes. Starting a scene execution must be regarded as the *instantiation* of a particular scene node in the performative structure. Thereafter such active scenes may possibly become available to other agents (depending on whether they offer access states), which may apply for entering. But also some agents participating in active scenes may leave them either to start other scene executions or join in other active scenes. At the same time, some active scenes may finish and therefore the activity that they represent. Summarising, an agent participating in the execution of a performative structure devotes his time to jointly start new scene executions, to enter active scenes where the agent interacts with other agents, to leave active scenes to possibly enter other scenes, and finally to abandon the performative structure.

At this point we must notice that the way agents move from scene to scene depends on the type of relationship holding among the source and target scenes. As mentioned above, sometimes we might be interested in forcing agents to synchronise before jumping into either new or existing scene executions, or offer choice points so that an agent can decide which target scene to incorporate into, and so on. Summarising, in order to capture the type of relationships listed above we consider that any performative structure contains a special type of scene, the so-called *transition* scenes[4], devoted to mediate different types of

[4]Henceforth *transitions* for shorter. Although transitions are a particular class of scenes, hereafter we will be using the terms separately to distinguish non-transition scenes from transition scenes.

connections among scenes. Scenes and transitions are connected by means of *arcs*. Each scene may be connected to multiple transitions, and in turn each transition may be connected to multiple scenes. In both cases, the connection between a scene and a transition is made by means of a directed arc. Then we can refer to the source and target of each arc. And given either a scene or a transition, we shall distinguish between its incoming and outgoing arcs. Notice that there is no direct connection between two scenes, or, in other words, all connections between scenes are mediated by transitions.

Figure 3.4 depicts the graphical elements that we will employ to represent the components of a performative structure introduced so far. From the point of view of the modeller, such graphical components are the pieces that serve to construct graphical specifications of performative structures.

Agents will be moving from a scene instance (execution) to another by traversing the transition connecting the scenes and following the arcs that connect transitions and scenes.

In order to traverse a given transition towards new or existing scene executions, an agent must first leave the scene instance in which it's taking part following some arc(s) which connects the scene with the transition when arriving at an exit state. While sometimes the agent should be given the possibility of choosing which path to follow, eventually we may need to enforce the agent to follow all possible paths. Thus, in the context of a performative structure, we classify the exit states of scenes into two classes: *and* and *or*. For the first type of exit state, an agent is forced to follow all available arcs, whereas it is the agent's choice which arc to follow for the second type of exit states. Figure 3.4 shows how we will represent the two different types of outputs from scenes.

What does make transitions different from the rest of scenes? Transitions must be regarded as a kind of routers. Therefore, instead of modelling some activity, they are intended to route agents towards their destinations in different ways, depending on the type of transition.

The arcs connecting transitions to scenes play a fundamental role. Notice that as there might be multiple (or perhaps none) scene executions of a target scene, it should be specified whether the agents following the arcs are allowed to start a new scene execution, whether they can choose a single or a subset of scenes to incorporate into, or whether they must enter all the available scene executions.

We define a set of different types of transitions and arcs whose semantics will highly constrain the mobility of agents among the scene instances (the ongoing activities) of a performative structure. The differences between the diverse types of transitions that we consider are based on how they allow to progress the agents that they receive towards other scenes. Let us divide each transition into two parts: the *input*, through which it receives agents from the incoming arcs, and the *output*, through which agents leave following the outgoing arcs towards other scenes. Then, the following classification of transitions is based on the behaviour that they exhibit on their input and output sides:

- **Sync/Parallel**: They establish synchronisation and parallelism points since agents are forced to synchronise at their input to subsequently follow the outgoing arcs in parallel.

- **Choice/Choice**: They behave in an asynchronous way at the input (agents are not required to wait for others in order to progress through), and as choice points at the output (agents are permitted to select which outgoing arc, which path, to follow when leaving).
- **Sync/Choice**: They are a hybrid of the two types of transitions above: on the one hand, likewise *sync/parallel* transitions, they force agents to synchronise on the input side, while on the other hand, likewise *choice/choice* transitions, they permit agents to individually make the choice of which path to follow when leaving.
- **Choice/Parallel**: They are also a hybrid of *sync/parallel* and *choice/choice* transitions. Agents are not required to wait for others on the input side, but they are forced to follow all the possible outgoing arcs.

According to this classification, we define $\mathcal{T} = \{sync/parallel, choice/choice, sync/choice, choice/parallel\}$ as a set of transition types.

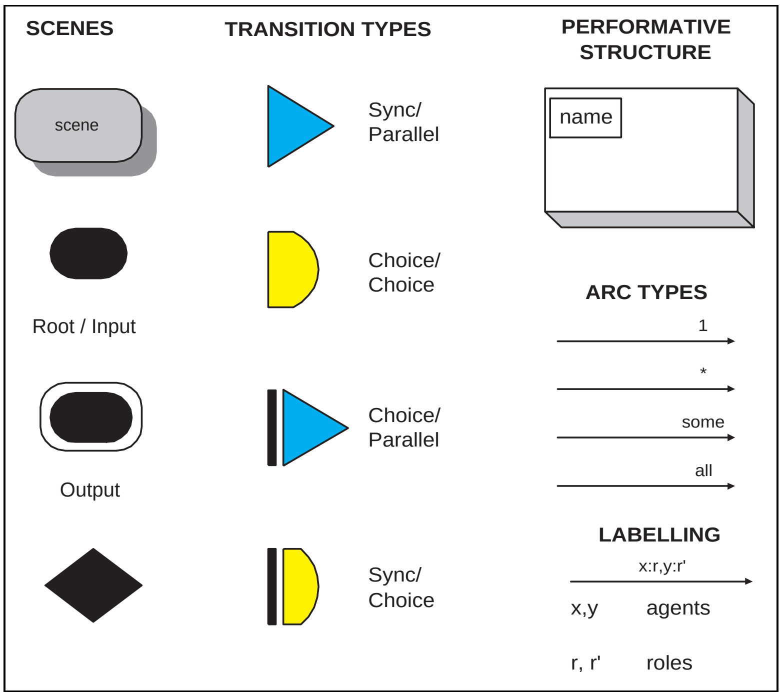


Figure 3.4: Graphical Elements of a Performative Structure

Depending on the path followed by the agents when traversing a transition, they may either start scenes or incorporate to one or more ongoing scenes. Thus there are also different types of paths, arcs, for reaching scenes after traversing transitions. We define $\mathcal{E} = \{1, some, all, *\}$ as the set of arc types. Following a *1-arc* constrains agents to enter a single scene instance of the target scene, whereas a *some-arc* is less restrictive and allows the agents to choose a subset of scene instances to enter, and an *all arc* forces the agents to enter all the scene instances to which the paths lead. Finally, a **- arc* fires the creation of a new scene instance of the target scene.

Furthermore, each arc will be labelled with the collection of roles that are allowed to follow the arc such as depicted in Figure 3.4.

Before stating a concrete definition of performative structure, there is a last element to be considered. Notice that although two scenes may be connected by a transition, the eventual migration of agents from a source scene instance to a target scene instance not only depends on the role of the agents but also on the results achieved by agents in the source scene. Thus, for instance, in the fish market, although a registration scene is connected to an auction scene, the access of a buying agent to the execution of the auction scene is forbidden if it has not successfully completed the registration process when going through a registration scene. In a conference centre environment, two agents are not allowed to meet to talk at a meeting room unless they have previously arranged and committed to some appointment. This fact motivates the introduction of constraints over the arcs connecting scenes and transitions. We will require that agents satisfy the constraints, conditions, over the arc solicited to be followed when attempting to reach a destination scene. Therefore, conditions must also appear in our formal definition of performative structure.

Finally, we bundle all the elements introduced above to provide a formal definition of a performative structure specification:

Definition 3.3.3. A performative structure is a tuple

$$PS = \langle S, T, s_0, s_\Omega, E, f_L, f_T, f_E, C, \mu, ML \rangle$$

where

- S is a finite, non-empty set of scenes;
- T is a finite and non-empty set of transitions;
- $s_0 \in S$ is the *root* scene;
- $s_\Omega \in S$ is the *output* scene;
- $E = E^I \bigcup E^O$ is a set of arc identifiers where $E^I \subseteq S \times T$ is a set of edges from scenes to transitions and $E^O \subseteq T \times S$ is a set of edges from transitions to scenes;
- $f_L : E \longrightarrow 2^{V_A \times R}$ is the labelling function;
- $f_T : T \longrightarrow \mathcal{T}$ maps each transition to its type;
- $f_E : E^O \longrightarrow \mathcal{E}$ maps each arc to its type;

- $C : E \longrightarrow ML$ maps each arc to a meta-language expression of type boolean, i.e. a formula representing the arc's constraints that agents must satisfy to follow the arc;
- $\mu : S \longrightarrow \mathbb{N}$ sets the upper bound on the number of allowed simultaneous scenes for the scene scheme represented by each scene node; and
- ML is a meta-language.

In order to present the requirements to be satisfied by the elements above, some previous definitions are needed. Let $s \in S$ be a scene such that $s = \langle R, DF, W, w_0, W_f, (WA_r)_{r\in R}, (WE_r)_{r\in R}, \Theta, L, min, Max\rangle$. Let var and $roles$ be two functions over arcs, defined respectively as $var : E \longrightarrow 2^{V_A}$ and $roles : E \longrightarrow 2^{Roles}$, which return for each arc the set of variables and the set of roles contained in the arc label. Furthermore, to differentiate the elements of two scenes s and s' we will use a super-index s or s'.

Then the following requirements must be satisfied by every performative structure:

(i) The root scene of a performative structure is not connected to the output of any transition. Formally, $\nexists t \in T \quad | \quad (t, s_0) \in E^O$.

(ii) The output scene is not connected to the input of any transition. Formally, $\nexists t \in T \quad | \quad (s_\Omega, t) \in E^I$. This requirement and the requirement above indicate that any performative structure starts at the root scene and finishes at the output scene.

(iii) The output of every transition is connected to some scene. Formally, $\forall t \in T \quad \exists s \in S \bigcup \{s_\Omega\} \quad | \quad (t, s) \in E^O$.

(iv) The input of every transition is connected to some scene. Formally, $\forall t \in T \quad \exists s \in S \bigcup \{s_0\} \quad | \quad (s, t) \in E^I$.

(v) All the agent variables labelling the incoming arcs of a given transition must also appear on the outgoing arcs of the transition: for any $t \in T$ $\bigcup_{(s,t)\in E^I} var((s,t)) = \bigcup_{(t,s')\in E^O} var((t,s'))$

(vi) All the agent variables labelling the incoming arcs of a given transition are different: for all $t \in T$, for all $(s,t), (s',t) \in E^I \quad var((s,t)) \bigcap var((s',t)) = \emptyset$

(vii) For every scene there is a path from the root scene to the output scene passing through the scene. For all $s \in S$ there is a path $s_0, t_1, \ldots, , s_m, t_m, s_\Omega$ such that $(s_i, t_i) \in E^I, (t_i, s_{i+1}) \in E^O \quad \forall i = 0, \ldots, m-1$ and $s_i = s$ for some i.

(viii) For every scene, the roles labelling its outgoing arcs must belong to its role set and for every role in its role set there is at least one outgoing arc labelled with it, i.e. for all $s \in S \quad \bigcup_{(s,t)\in E^I} roles((s,t)) = R^s$.

(ix) For every scene, the roles labelling its incoming arcs must belong to its role set and for every role of its role set there is at least one incoming arc labelled with it, i.e. for every scene $s \in S$ $\bigcup_{(t,s) \in E^O} roles((t,s)) = R^s$.

(x) For every scene, there must be at least one access state for every role labelling an incoming arc, i.e. for every arc $(t, s) \in E^O$, and for every role $r \in roles((t, s))$ there is an access state $w \in WA_r$.

(xi) For every *sync/parallel* transition, every outgoing arc has to be connected to a scene containing at least one access state for all the roles labelling the arc. More formally, for every (t, s) such that $f_T(t) =$ *sync/parallel* then $\bigcap_{r \in roles((t,s))} WA_r \neq \emptyset$.

(xii) For every arc of type $*$ the initial state of the scene must belong to the set of access states for each role labelling the arc. Formally, for each arc $(t, s) \in E^O$ such that $f_E((t, s)) = *$ we require that $w_0 \in \bigcap_{r \in roles((t,s))} WA_r$.

From the definition above, we can simply view a performative structure as a network of scenes interconnected by different types of transitions. The specification of a performative structure amounts to select a collection of scenes, such as those depicted by Figures 3.3 and 3.2, and next create sound connections among them. Complementarily, the limits on the number of scenes, μ, need to be defined. Figures 3.12 and 3.13 show how the graphical elements in Figure 3.4 are combined in order to produce the graphical specification of the performative structures corresponding to the fish market and to the conference centre environment. Observe that the scenes that Figures 3.3 and 3.2 represent do appear in the resulting specifications as two particular nodes.

Notice that we demand any performative structure to contain a root and an output scene. The output scene does not model any activity, and so it must be regarded as the exit point of the performative structure. As to the root scene, it must be regarded as the starting point of any agent accessing the performative structure. Departing from the root scene, agents will make for other scenes within the performative structure.

Containers

It is time to introduce a special type of scene, the so-called *container* scene (or container for shorter), depicted in Figure 3.4. Analogously to a root scene, there is no activity within containers. Formally, we can think of containers as scenes with a single state which is both initial and final. In practice, they are employed to mark execution points to which we may force agents return. Therefore they are useful for specifying iterative activities as shown further on through the examples in Section 3.4.3. In practice, agents will not need to stop by these containers and will consider them as part of the arc to be followed. In this way, we avoid to introduce connections among transitions.

Textual Representation

Analogously to scenes, here we present part of the XML-based specification of the performative structure depicted in Figure 3.13 —identified as *comris-01*— based on the document type definition enclosed in Appendix A.

```
<?xml version ="1.0" standalone = 'no'?>
<!DOCTYPE PS SYSTEM "http://www.iiia.csic.es/~jar/AMEI/dtd/ps.dtd">
   <PS ID="comris-01">
      <DESCRIPTION>
         Comris Performative Structure
      </DESCRIPTION>
      <NODE ID="node-01" MAX="100" HREF="scenelib.xml#aps">
         <ARCLIST>
            <ARC ID="arc-01" HREF="#transition-01">
               <LABEL>
                  <AGENTVAR> x </AGENTVAR>
                  <ROLEID> pra </ROLEID>
               </LABEL>
            </ARC>
         </ARCLIST>
      </NODE>
      <NODE ID="node-02" MAX="100" HREF="scenelib.xml#aps">
      </NODE>
      <TRANSITION ID="transition-01" TYPE="AND">
         <ARCLIST>
            <ARC ID="arc-02" HREF="#node-02">
               <LABEL>
                  <AGENTVAR> x </AGENTVAR>
                  <ROLEID> pra </ROLEID>
               </LABEL>
            </ARC>
         </ARCLIST>
      </TRANSITION>

<!--More specifications of scenes and transitions follow below-->


</PS>
```

3.3.5 Normative Rules

Such as depicted, a peformative structure constrains the behaviour of any participating agents at two levels:

(i) *intra-scene:* Scene protocols dictate for each agent role within a scene what can be said, by whom, to whom, and when.

(ii) *inter-scene:* The connections among the scenes of a performative structure define the possible paths that agents may follow depending on their roles. Furthermore, the constraints over output arcs impose additional limitations to the agents when attempting to reach a target scene.

And yet, we understand that an agent's actions within a scene may have consequences that either limit or enlarge its subsequent acting possibilities.

Such consequences have effect along two different directions. On the one hand some actions will introduce subsequent acting commitments that have to be interpreted as acting obligations. On the other hand, other consequences occurring locally within a scene may enlarge or restrict the number of paths that an agent can follow in the performative structure because they affect the satisfaction and dissatisfaction of the constraints labelling paths. Both types of consequences will be required to be kept by an institution for each agent on an individual basis. In general, for a given agent we shall refer to such consequences as the agent *obligations* within the institution.

For instance, a trading agent winning a bidding round within an auction house is obliged to subsequently pay for the acquired good. Considering the performative structure in Figure 3.12 that implies that the trading agent has to move at some time to the buyers' settlements scene to pay for the acquired good. Notice that although the auction scene is connected to the output scene, the path is disallowed to agents unless they fulfil their pending payments.

In order to represent the deontic notion of obligation, we set out the predicate *obliged* as follows:

$obliged(x, \phi, s)$ = agent x is obliged to utter an illocution satisfying ϕ in scene s.

where ϕ is taken to be an illocution scheme. Notice that the formulae expressing obligations will be considered as part of a meta-language.

Next we introduce a special type of rules, the so-called *normative rules*, intended to provide a means of capturing which agent actions (illocutions) have consequences in terms of obligations. The institution is responsible for guaranteeing that every single agent participating in its activity abides by normative rules. Notice that the behaviour of an institution shall be highly influenced by normative rules since these constitute its main enforcement component.

Syntactically, in its most general form, normative rules will be composed of and meta-language formulae:

$$\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi_{n+1} \wedge \ldots \wedge \phi_r$$

where $\phi_1 \wedge \ldots \wedge \phi_r$ are meta-language formulae. While some of these rules will be devoted to the triggering of obligations, others will be used for inferring facts that will be subsequently employed to determine the access of an agent to other scenes.

In order to undo a pending obligation ascribed to an agent, the following basic rule applies:

$$obliged(x, \phi, s) \wedge done(\phi, s) \Rightarrow \neg obliged(x, \phi, s)$$

where $done(\phi, s)$ is a meta-language formula expressing that an illocution with scheme ϕ has been uttered in scene s.

Notice that normative rules cannot be expressed by any other elements or structures introduced so far. In order to understand them better, we must view them as being part of the meta-level of a performative structure.

Notice also that normative rules do not have to do with agents' decision machinery, but with their social, external behaviour. Thus their only purpose is to formally underpin a normative environment, along the lines of human institutions.

In what follows we encode some of the normative rules derived from the description of the *fish market*, the case study institution presented in Section 1.3.
Rule: Price call. After calling out a given offer($price$) in the auction room scene($auction\_room$) for a given good ($good\_id$), an auctioneer($x : a$) is obliged to call out a new offer ($price - price\_step$) whenever no bids have been received at $price$ and the new offer price is below the good's reserve price (established when a seller ($z : s$) delivers the good to a sellers' admitter($y : sa$)).

Rule: Price call

$done(inform(?x : a, all : b, offer(?good\_id, ?price)), auction\_room) \quad \wedge$
$done(commit(?y : sa, ?z : s, sell(!good\_id, ?starting\_price, ?reserve\_price, ?price\_step)), good\_delivery) \quad \wedge$
$!price - !price\_step > !reserve\_price \quad \wedge$
$bids(!good\_id, !price) = 0 \quad \Rightarrow$
$obliged(!x, inform(!x : a, all : b, offer(!good\_id, !price - !price\_step)), auction\_room)$

Rule: Good adjudication. An auctioneer ($x : a$) is obliged to declare that a given good is sold to a buyer ($y : b$) if he is the only bidder at the current offer ($price$) and his credit ($credit(y)$) is bigger than his bid.

Rule: Good adjudication

$done(inform(?x : a, all : b, offer(?good\_id, ?price)), auction\_room) \quad \wedge$
$\exists! \quad ?y \quad done(commit(?y : b, !x : a, bid(!good\_id, !price)), auction\_room) \quad \wedge$
$credit(!y) \geq !price \quad \Rightarrow$
$obliged(!x, inform(!x : a, all : b, sold(!good\_id, !y, !price)), auction\_room)$

Rule: Bidders' collision. An auctioneer ($x : a$) is obliged to declare a collision at the current offer price ($price$) when there are at least two bidders at the current offer price.

Rule: Bidders' collision

$done(inform(?x : a, all : b, offer(?good\_id, ?price)), auction\_room) \quad \wedge$
$\exists \quad ?y, ?z \quad done(commit(?y : b, !x : a, bid(!good\_id, !price)), auction\_room) \quad \wedge$
$done(commit(?z : b, !x : a, bid(!good\_id, !price)), auction\_room) \quad \wedge$
$credit(!y) \geq !price \quad \wedge$
$credit(!z) \geq !price \quad \Rightarrow$
$obliged(!x, inform(!x : a, all : b, collision(!good\_id, !price)), auction\_room)$

Rule: Bidder sanction. An auctioneer is obliged to sanction any given bidder ($y : b$) when he submits a bid at the current offer price and his credit ($credit(y)$) cannot support his offer.

Rule: Bidder sanction

$done(inform(?x : a, all : b, offer(?good\_id, ?price)), auction\_room) \quad \wedge$
$\exists! \quad ?y \quad done(commit(?y : b, !x : a, bid(!good\_id, !price)), auction\_room) \quad \wedge$
$done(commit(!y : b, !x : a, bid(!good\_id, !price)), auction\_room) \quad \wedge$
$credit(!y) < !price \quad \Rightarrow$
$obliged(!x, inform(!x : a, all : b, sanction(!y)), auction\_room)$

Rule: Good withdrawal. An auctioneer is obliged to withdraw a good at auction whenever there are no bids at the current offer price($bids(good\_id, price) = 0$) and the next offer price to be called out ($price - price\_step$) is below the reserve price ($reserve\_price$).

Rule: Good withdrawal

$done(inform(?x : a, all : b, offer(?good\_id, ?price)), auction\_room) \quad \wedge$
$done(commit(?y : sa, ?z : s, sell(!good\_id, ?starting\_price, ?reserve\_price, ?price\_step)), good\_delivery) \quad \wedge$
$!price - !price\_step < !reserve\_price \quad \wedge$
$bids(!good\_id, !price) = 0 \quad \Rightarrow$
$obliged(!x, inform(!x : a, all : b, withdrawn(!good\_id, !price)), auction\_room)$

Rule: Buyers' payment. If a buyer ($buyer\_id : b$) acquires a good during a bidding round (the auctioneer announces that the good is sold to him), he is obliged to subsequently pay for the good to a buyers' accountant ($y : bac$).

Rule: Buyers' payment

$done(inform(?x : a, all : b, sold(?good\_id, ?buyer\_id, ?price)), auction\_room) \quad \Rightarrow$
$obliged(!x, pay(!buyer\_id : b, !y : bac, sale(!good\_id, !buyer\_id, !price)), buyer\_settlements)$

Rule: Sellers' payment. If an auctioneer commits to sell the good at auction to a given buyer, this is obliged to to pay for the good to a buyers' accountant in a *settlements* scene.

Rule: Sellers' payment

$done(inform(?x : a, all : b, sold(?good\_id, ?buyer\_id, ?price)), auction\_room) \quad \wedge$
$done(pay(?buyer\_id : b, ?y : bac, sale(!good\_id, !buyer\_id, !price)), buyer\_settlements) \quad \Rightarrow$
$obliged(!y : sac, pay(!y : sac, seller(!good\_id), sale(!good\_id, !buyer\_id, !price)), seller\_settlements)$

Rule: Auction room closing. The auctioneer is obliged to end the auctioning when required by the boss of the market (x : *boss*) once the current bidding round ends up.

Rule: Auction Room Closing

$done(request(?x : boss, ?y : a, close\_market(now)), closing) \quad \wedge$
$done(inform(!y : a, all : b, open\_auction(?n)), auction\_room) \quad \wedge$
$\neg done(inform(!y : a, all : b, end\_auction(!n)), auction\_room) \quad \wedge$
$done(inform(!y : a, all : b, end\_round(?r)), auction\_room) \quad \wedge$
$\neg done(inform(?! : a, all : b, end\_round(!r + 1)), auction\_room) \quad \Rightarrow$
$obliged(!y, inform(!y : a, all : b, end\_auction(!n)), auction\_room)$

In order to provide participants with a complete specification of an electronic institution, normative rules must be published along with the specification of a performative structure and its scenes. In this way, an agent would be able to understand from the specification what can be said, to whom and when, along with the eventual consequences of all actions undertaken within the institution. In this way, an agent can decide on the convenience of participating in any given institution.

3.3.6 Electronic Institution

Finally, we can define an electronic institution by choosing a performative structure and defining the rest of its components. These are the institutional roles, the hierarchy between roles, the policy of duties and the normative rules. Institutional roles define a set of roles that can not be played by external agents. They are like the employees in a human institution. Notice that the hierarchy of roles and the policy of duties are applied to the set of all roles within the institution which are all the roles appearing in the different scenes composing the performative structure.

Definition 3.3.4. An electronic institution is defined as a tuple

$$\langle \mathcal{PS}, IR, \succeq, ssd, N_{PS}, ML \rangle$$

where

- PS stands for a performative structure;
- IR is a subset of roles representing the institutional roles;

- $\succeq$ stands for the hierarchy partial order over the set of all roles;
- ssd is the set of static separation of duties between roles; and
- N_{PS} stands for a set of normative rules.
- ML is the meta-language employed for encoding the normative rules in N_{PS}.

Notice that the specification of an electronic institution must be regarded as a compositional activity to be undertaken by the institution designer. In general, the designer must go through the following steps.

- define a role set including the cardinality of each role and the policies of separation of duties;
- select or construct a set of specifications of dialogical frameworks;
- select or construct a set of scenes;
- select or construct a performative structure; and
- define a set of normative rules.

We say that the designer can opt for either selecting or constructing some specifications because there might exist specifications already available to be reused for composing new specifications. Thus we can consider that specifications of dialogical frameworks, scenes, and peformative structures are to be naturally organised into specification libraries.

Once completed a specification, it must go through a validation process that checks its well-formedness. Ultimately, if such a specification proves to be correct, its equivalent textual representation must be generated in order to be manipulated by the agents intending to participate in the institution. We postpone our example of textual representation of a given specification till Section 3.4, where complete graphical specifications of the case studies introduced in Chapter 1 are provided.

In the light of the complexity of the whole process, it is apparent the need of tools that assist the institution designer through the specification, validation, and translation of an electronic institution into a machine-readable format so that it can be easily parsed by agents.

3.4 Execution Model

Notice that through Section 3.3 we have pursued to provide a static, structural definition of electronic institution. Now we draw our attention towards the dynamics of an electronic institution when it becomes populated by agents. For this purpose, we take the view that scenes, transitions and performative structures as objects whose evolution depends on the dialogic actions of the agents immersed in the institution. Then we provide a functional specification of the operations over scenes, transitions and performative structures that shall ideally help us understand the dynamics of electronic institutions.

3.4.1 Agents and Roles

An institution comes to life when populated by agents. But in order for an agent to join in an electronic institution it is obliged to adopt some role(s). Once adopted a particular role, an agent is enforced to conform to the pattern of behaviour established for the role. We consider that an agent can adopt one or more roles, and in turn a single role can be adopted by one or more agents. And so we establish a many-to-many relationship between agents and roles.

When entering an institution, an agent must request for the authorisation of the role(s) that it intends to play within the institution. Then the institution assigns a subset of these roles to the agent taking into account the existence of conflicting roles (pairs of mutually exclusive roles). Recall that roles are organised into a hierarchy. Thus, for each assigned role an agent is also authorised to eventually play the roles below in the hierarchy. For instance, in Figure 3.1, if an agent is *assigned* (and so authorised) the role *specialist*, it is also *authorised* to play the role *doctor*.

We impose as a constraint that an agent can only play a single role within a scene, and such role must belong to its set of authorised roles. However since an agent can possibly participate in multiple scenes, it can play various roles simultaneously. Then we will mean by *active* roles the set of roles currently played by an agent within the institution.

Notice that so far we have distinguished *assigned*, *authorised* and *active* roles. The following functions will be employed for referring to them separately:

- $assigned\_roles : Agents \longrightarrow 2^{Roles}$. $assigned\_roles(a)$ denotes the set of roles assigned to agent a.
- $authorised\_roles : Agents \longrightarrow 2^{Roles}$. $authorised\_roles(a)$ denotes the set of roles authorised to agent a. We say that a role r is authorised for an agent a if either r is assigned to a, or r is subsumed by another role that is assigned to a.
- $active\_roles : Agents \longrightarrow 2^{Roles}$. $active\_roles(a)$ denotes the set of roles currently played by agent a.

While $assigned\_roles$ is conceived as a static assignment which is kept while an agent takes part in an institution's activities, $active\_roles$ is expected to be varying as the agent takes on new roles or ends up playing other roles.

Next, we make explicit the requirements needed for handling agent/role associations[5]:

- An agent can only play authorised roles.

[5]Notice that the first two requirements concern in fact role/role associations, but have been included here for explanatory purposes.

- Any two roles assigned to the same agent do not subsume one another.
- Any two roles authorised to the same agent do not violate the static separation of duties, i.e. an agent cannot be authorised to play mutually exclusive roles.

Put formally:

Requirement 1. For all $a \in Agents, active\_roles(a) \subseteq authorised\_roles(a)$.

Requirement 2. For all $a \in Agents, \forall r_i, r_j \in Roles, r_i, r_j \in assigned\_roles(a) \Rightarrow \neg(r_i \succeq^+ r_j)$ where $\succeq^+$ stands for the transitive closure of $\succeq$.

Requirement 3. For all $a \in Agents, \forall r_i, r_j \in Roles, r_i, r_j \in authorised\_roles(a) \Rightarrow (r_i, r_j) \notin ssd$.

Notice that requirements 2,3 can be easily enforced when managing the access of an agent to the institution. Differently, requirement 1 is demanded to be held during an institution's life-cycle. As we briefly sketched in the introduction of this section, the activity of an institution is composed of multiple, concurrent scenes populated by agents that move around by jumping into and out of scenes. Consequently, requirement 1 will have to be be satisfied in the transit of agents among scenes. Therefore, both requirements will be revisited later on when presenting the dynamics of performative structures.

3.4.2 Scene

Notice that the specification of scene is purely static, simply establishing a pattern of multi-role conversation (no agents, but the roles that they might take on, are considered). Then there remains to explain how such a specification is adequately exploited for modelling the behaviour of a scene, i.e. the dynamics of a multi-agent conversation.

For this purpose, a scene must be executed. Any scene execution is firstly started by a group of agents intending to engage in a conversation following the rules specified by the scene. Here we assume that during a scene execution each agent plays a single role. Once started a scene execution, it will evolve as the participating agents legally utter illocutions that make the conversation, the scene execution, progress. Furthermore, an execution will also vary as some participating agents leave when the conversation reaches exit states and non–participating agents join in when the conversation reaches access states.

Recall that the arcs connecting the states of a scene are labelled with illocution schemes containing variables. Such variables correspond to agent variables, role variables and variables in the contents of the illocution scheme. During a scene execution, the variables in an illocution scheme are bound to the values of the uttered illocution matching the scheme. In order to keep track of the context of a scene, a list of bindings is maintained keeping the bound variables along with their values. Notice however that we may eventually need that some of these bindings change dynamically. In this way the very same variable may be bound to a different value at different stages of the execution of a scene.

We distinguish two cases regarding the change of the binding for a particular variable. Firstly, when the utterance of an illocution brings the scene state to a past state that closes a cycle. In such a case, the context prior to the beginning of the cycle is recovered in the sense of free and bound variables. Secondly, when the value of an already bound variable is overwritten as introduced in Section 3.3.2.

Considering a cycle, the list of bound variables is composed of variables bound before and during the cycle. Variables bound during a cycle become free again once the cycle completes, i.e. the scene goes back to the visited state at which the cycle started. For the purpose of controlling cycles, a special list of visited states is used, characterised by not containing visited states belonging to a cycle —except for scene states marking the starting point of a cycle.

Complementarily, in order to recover the context when a cycle is closed, a list of bound variables is kept for each scene state containing the variables bound when each state was last visited.

Building upon the intuitions provided above, next we describe the execution of a scene by providing the possible operations that transform it from one state to another. In this way we manage to specify the dynamic evolution of a scene execution composed of multiple interacting agents. As we intend to construct a formal specification, the operations are specified by giving their pre-conditions and post-conditions, and not by giving a procedure for how those operations are to be carried out. Hence, we show under what conditions each operation preserves the scene execution consistency. If a scene execution is in a consistent state, and a certain set of conditions is satisfied by (the arguments of) an operation, then it remains in a consistent state after the operation is performed. For each operation, we specify its arguments, semantics, and consistency preserving conditions. In the semantics specification of each operation, a primed variable denotes its value after the operation has been performed[6].

In order to keep track of the state of a scene execution we introduce the notion of *scene execution descriptor*(or scene descriptor for shorter) departing from the definition of scene:

Definition 3.4.1. Given a scene $s = \langle R, DF, W, w_0, W_f, (WA_r)_{r \in R}, (WE_r)_{r \in R}, \Theta, \lambda, min, Max \rangle$, an execution of s will be characterised by a tuple $\Sigma_s =< A, \rho, \sigma_s, w, N, (V_{w_i})_{w_i \in W} >$ where

- $A \subseteq Agents$ is the set of participating agents;
- $\rho : A \longrightarrow R$ maps each participating agent to its role in the scene;
- σ_s contains all the variable bindings at each moment during the scene execution;
- w is the scene state;

[6]Likewise Z specifications.

- N is the list of visited states without cycles, i.e. without containing visited states belonging to a cycle —except for scene states marking the starting point of a cycle;
- V_{w_i} stands for the list of bound variables last time the state w_i was visited.

Notice that an important restriction applies to the number of roles that an agent can play within a scene, since each agent can only play a single role (following the formal definition above, the ρ function maps each agent to a single role).

In order for a scene to start executing, there must be a group of agents for which the following pre-conditions apply:

- the roles requested to be played by the agents do not exceed the cardinality of the scene roles and the minimum for each role is reached;
- the requested roles are indeed authorised to each agent; and
- the initial state of the scene is an access state for all the requested roles.

If this is the case, a scene can start being executed and so the scene starts at w_0 —the initial state— and σ_s and V_{w_0} are initialised as empty lists while N is initialized to w_0. Furthermore, the variables of the scene are free and so they can be subsequently bound to any value of its type.

More formally, the creation of an execution descriptor of a scene s with an initial group of agents $Ag \subseteq Agents$ such that each agent $a \in Ag$ requests for playing a role $\rho_{Ag}(a)$ can be specified by means of the *start* operation as follows:

$\mathbf{start}(\mathbf{s}, \mathbf{Ag}, \rho_{\mathbf{Ag}})$

Pre-conditions.
(c0) $\forall r \in R \quad min(r) \leq \quad \| \{a \in Ag | \ \rho_{Ag}(a) = r\} \| \quad \leq Max(r)$
(c1) $\forall a \in Ag, w_0 \in WA_{\rho_{Ag}(a)}$
(c2) $\forall a \in Ag, \rho_{Ag}(a) \in authorised\_roles(a)$

Post-conditions.
$w' = w_0$
$A' = Ag$
$\rho' = \rho_{Ag}$
$\sigma'_s = \{\}$
$N' = \{w_0\}$
$V'_{w_0} = \{\}$

whose output is a new execution descriptor Σ_s such that $\Sigma_s = \langle A', \rho', \sigma'_s, w', N', (V'_{w_i})_{w_i \in W} \rangle$.

Once started the execution of a scene, the participating agents are allowed to interact with one another by uttering legal illocutions. By legal we mean illocutions that comply with the rules of the scene, which explicitly state what can be said, by whom and when. Valid interactions among the agents make the conversation evolve, and hence its underlying activity. For each non-final state w there is at least one state w_j such that the illocution schema labelling (w, w_j) states which role can speak, to whom, an what can be said. Notice that the variables in the illocution schemes labelling the arcs become bound as illocutions are uttered during the execution of scenes. In other words, the context of the scene kept in σ_s, is constructed as illocutions are uttered. Then an illocution u uttered at state w is valid if it matches the label of (w, w_j) once its bound variable are substituted considering the bindings in σ_s. Summarising, the variables in the illocution scheme already bound are substituted by their values in the list of bindings. whereas the remaining (free) variables are assigned their values from illocution u generating new bindings which are used for updating the context. This new bindings are kept in a list that we denote as σ_u.

Then the state of the conversation switches from w to w_j. Observe that not only a state transition occurs, but also an eventual change in the context captured by the *update_context* function that we thoroughly detail further on. Below, the operation *update_state* specifies the changes in an execution descriptor Σ_s when a illocution u is uttered:

$\mathbf{update\_state(\Sigma_s, u)}$
Pre-conditions. (c0) $w \notin W_f$ (c1) $u \in CL$ (c2) $\exists (w, w_j) \in E \mid \lambda(w, w_j) = \iota(\alpha_i : r_i, \alpha_j : r_j, \phi, t)$ and u matches $\sigma_s(\lambda(w, w_j))$ **Post-conditions.** $w' = w_j$ $\langle \sigma'_s, N', V'_{w_j} \rangle = update\_context(\sigma_u)$

Next we describe how the list of bindings (σ_s), the list of visited states (N) and the list of bound variables of state w_j (V_{w_j}) are updated when a new illocution is uttered and the scene evolves to the w_j state.

Let σ_u be the list of new bindings generated by the utterance of illocution u. More precisely , σ_u contains the bindings for the free variables of the illocution along with the overwritten variables. The purpose of the process that follows is to update σ_s with the information in σ_u taking into account whether the transition corresponds to a loop that brings the scene to a visited state. If so the states composing the loop are removed from the list of visited states N and the bindings of the variables not appearing in V_{w_j} are removed from σ_s —in this way the context corresponding to w_j is recovered. If not w_j is added to N as a visited state and the variables in σ_s are assigned to V_{w_j}.
where

Function 1 *update_context*(σ_u)

```
1: $\sigma_s \Leftarrow \sigma_u \; op \; \sigma_s$
2: if $w_j \in N$ then
3:     $N \Leftarrow w_0 \cdots w_j$
4:     $\sigma_s \Leftarrow \sigma_s \; proj \; V_{w_j}$
5: else
6:     $N \Leftarrow N w_j$
7:     $V_i \Leftarrow Var(\sigma_s)$
8: end if
9: return $\langle \sigma_s, N, V_{w_j} \rangle$
```

- σ_u op σ_s stands for the union of the bindings in both lists taking into account that if there is a variable appearing in both, the binding in σ_u is taken because this means that the variable has been overwritten;
- σ_s proj V_{w_j} stands for the projection of the variables in V_{w_j} over the bindings in σ_s. As a result, this operation keeps in σ_s the bindings for the variables in V_{w_j}, removing the others; and
- Var(σ_s) returns the bound variables, i.e. the variables in σ_s.

We can see an example of the execution of the scene depicted in Figure 3.2 in Table 3.4. There we consider two agents identified as $pra1$ and $pra2$ playing both the pra role in the scene. Then the scene starts with $Ag = \{pra1, pra2\}$. Since neither Ag nor ρ change during the execution of the scene, they are not shown in Table 3.4. The table presents how the rest of variables change when the scene evolves when the illocutions listed below are uttered. Notice that an *update_state* operation occurs after each of the following illocution:

(i) $request(pra1, pra2, app(e - commerce))$ at w_0

(ii) $request(pra2, pra1, app(auctions))$ at w_1

(iii) $request(pra1, pra2, app(strategies))$ at w_2

(iv) $accept(pra2, pra1, app(strategies))$ at w_1

The first column of the table represents the state of the scene, σ_i the bindings generated after each illocution, σ_s contains all the bindings of the scene, V_w the list of bound variables that are kept at the current scene state and finally N represents the list of visited states without cycles.

The first row of the table shows how the variables are initialised once scene starts. Recall that in the beginning all scene variables are free. After the first illocution occurs three new bindings are generated for each one of the variables appearing in the illocution as shown in the σ_i column. Furthermore, the list of scene bindings σ_s is updated considering the new bindings in σ_i. As a result of the first illocution, the scene evolves to the state w_1, which is then added to N.

At w_1 there are three outgoing arcs each one labelled with a different illocution scheme. After replacing the occurrences of the bound variables in such illocution schemes using the values in σ_s, the next illocution to be uttered is due to match one of the following expressions:

- $request(pra2, pra1, app(?t'))$
- $commit(pra2, pra1, app(e - commerce))$
- $refuse(pra2, pra1, reason(?r))$

t	w	σ_i	σ_s	V_w	N
0	w_0	$\{\}$	$\{\}$	$\{\}$	$\{w_0\}$
1	w_1	$\{x/pra1\ y/pra2\ t/e\}$	$\{x/pra1\ y/pra2\ t/e\}$	$\{x\ y\ t\}$	$\{w_0 w_1\}$
2	w_2	$\{t'/a\}$	$\{x/pra1\ y/pra2\ t/e\ t'/a\}$	$\{x\ y\ t\ t'\}$	$\{w_0 w_1 w_2\}$
3	w_1	$\{t/s\}$	$\{x/pra1\ y/pra2\ t/s\}$	$\{x\ y\ t\}$	$\{w_0\ w_1\}$
4	w_{f_1}	$\{\}$	$\{x/pra1\ y/pra2\ t/s\}$	$\{x\ y\ t\}$	$\{w_0\ w_1\ w_{f_1}\}$

Table 3.4: Example of the evolution of the context during the execution of the scene in Figure 3.2 (the following abrreviations apply: e = e-commerce, a = auctions, s = strategies)

Notice that given the current state and context only the $pra2$ agent can utter an illocution whose addressee must be the $pra1$ agent. It has three possibilities: to propose new topics for the appointment, to accept the topic proposed by $pra1$ or simply to refuse the proposed appointment. In case it chooses to make a new proposal, this can include new topics because t' is a free variable. On the other hand, if it decides to accept the current proposal, the topic of the appointment will correspond to that proposed by $pra1$ to which t is bound.

In our example agent $pra2$ chooses to propose *auctions* as an alternative topic for the appointment, making the scene evolve to the w_2 state. This illocution generates a new binding for t', which is the only free variable. This binding is kept by σ_i and subsequently added to σ_s. As the occurring transition does not close a cycle, w_2 is added to the list of visited states.

Now there is an interesting aspect concerning the illocution scheme $request(!x : pra, !y : pra, app(?t))$ that allows $pra1$ to make a new proposal for the appointment topic. If the notation $?t$ is not used for t, this would be substituted by the value in σ_s corresponding to the first proposal submitted by $pra1$. Introducing $?t$ implies that t is handled as a free variable. Thus agent $pra1$ can propose new topics for the appointment and a new binding can be generated for t and kept in σ_i. And so the binding for t in σ_s is replaced for the new one generated in the last illocution.

Going back to our example, when the scene reaches w_2, $pra1$ proposes *strategies* as a new topic for the appointment, making the scene evolve to w_1. By going back to w_1 a cycle is completed and detected because w_1 belongs to N. Then, only the bindings for the variables in V_{w_1} are kept in σ_s while the rest of bindings are removed. In our case, $V_{w_1} = \{x, y, t\}$ and so the bindings for these variables are stored in σ_s while the binding for t' is removed. Thus t' becomes a free variable again allowing agent $pra2$ to propose new topics for the appointment. Furthermore N is updated by removing from it the states forming the cycle. Thus the w_2 state is removed from N.

Finally agent $pra2$ accepts to hold an appointment to discuss about the *strategies* topic proposed by $pra1$, and so the scene reaches its final state.

A distinguishing feature of scenes is that agents may enter or leave through the access and exit states.

Any agent can join in an ongoing scene whenever the scene execution is unfinished (the conversation has not reached a final state yet), the roles requested to be played by the agent do not exceed the cardinality of the scene roles, and the state of the scene is an access state for the role requested by the agent. In the most general case, the access of a group of agents $Ag \subseteq Agents$ to a scene execution represented by Σ_s such that each agent $a \in Ag$ requests for playing a role $\rho_{Ag}(a)$ and is authorised to play the roles $authorised\_roles(a)$, can be specified by means of the operation *add_agents* as follows:

add_agents$(\Sigma_\mathbf{s}, \mathbf{Ag}, \rho_\mathbf{Ag})$

Pre-conditions.
(c0) $w \notin W_f$
(c1) $\forall r \in R \quad \| \{a \in A | \; \rho(a) = r\} \| + \| \{a' \in Ag | \rho_{Ag}(a) = r\} \| \leq Max(r)$
(c2) $\forall a \in Ag, w \in W_\alpha^{\rho_{Ag}(a)}$
(c3) $\forall a \in Ag, \rho_{Ag}(a) \in authorised\_roles(a)$
(c4) $\forall a \in Ag, a \notin A$

Post-conditions.
$A' = A \cup Ag$
$\rho'(a) = \rho(a) + \rho_{Ag}(a)$

whose output is an execution descriptor $\Sigma'_s = \langle A', \rho', \sigma_s, w, N, (V_{w_i})_{w_i \in W} \rangle$.

We say that a scene being executed is *accessible* to Ag iff the conditions $c0, c1, c2c3$ and $c4$ above hold.

As an example of access state, in Figure 3.3 the w_1, w_3 states allow for the incorporation of more buyers during a bidding round.

On the other hand, any participating agent can leave a scene whenever the scene execution is unfinished, the role to be played out by the agent do not violate the cardinality of the scene roles, and the state of the execution state corresponds to an exit state for the role of the leaving agent. Notice that the variables in σ_s corresponding to the agents leaving the scene must be unbound.

In the most general case, given an execution descriptor Σ_s, the exit of a group of participating agents Ag can be specified by means of the operation *substract_agents* as follows:

$\mathbf{substract\_agents(\Sigma_s, Ag)}$
Pre-conditions.
(c0) $w \notin W_f$
(c1) $Ag \subseteq A$
(c2) $\forall a \in Ag, w \in W_{\omega}^{\rho(a)}$
(c3) $\forall a \in Ag, min(\rho(a)) \leq \lvert\{a' \in A - \{a\} \mid \rho(a') = r\}\rvert$
Post-conditions.
$A' = A - Ag$
$\rho'(a) = \rho(a) \quad a \in A - Ag$
$\sigma'_s = unbind(\sigma_s, Ag)$

whose output is a scene execution descriptor $\Sigma'_s = \langle A', \rho', \sigma'_s, w, N, (V_{w_i})_{w_i \in W} \rangle$. The function *unbind* above unbinds the variables bound to the agents leaving the scene.

We say that a scene execution is *open* for Ag iff the conditions $c0, c1, c2$ and $c3$ above hold.

To finish, once a final state is reached during the execution of the scene, the scene can be closed whenever no agents remain within, as specified by the *close* operation:

$\mathbf{close(\Sigma_s)}$
Pre-conditions.
(c0) $w \in W_f$
(c1) $A = \emptyset$
Post-conditions.
$\Sigma'_s = \emptyset$

To summarise, the general dynamics, the *life-cycle* of a scene execution can be specified via the following regular expression:

$$start.(update\_state|add\_agents|substract\_agents)^*.close$$

It is time to take into account that multiple scene executions corresponding to the very same scene specification might be concurrently running. For instance, multiple executions of the scene in Figure 3.3 could be running to allow for the parallel auctioning of goods. Furthermore, other executions corresponding to other scenes may also be running.

In the context of an electronic institution, the specification of the performative structure captures the relationships holding among the scenes composing the institution. Such relationships must be considered when agents are in play to understand how and under what conditions agents can either start a scene execution or move from a scene execution to another.

3.4.3 Performative Structure

Notice that the specification of performative structure provided in Section 3.3.4 is static, simply defining a network of scenes interconnected by means of transitions. As a matter of fact, no agents are considered to be part of the definition. Thus, there remains the matter of explaining how agents interact and how their behaviour becomes affected and constrained by a performative structure specification.

At the outset any performative structure uniquely contains two special, everlasting scenes, namely the root and output scenes. The root scene must be regarded as the starting point of any performative structure that all agents have to enter in order to act within the institution. No activity whatsoever takes place within this scene: agents enter, stand still, and leave to start new scenes or join ongoing scenes (if any).

On the other hand, the output scene is a special scene that agents make for when intending to leave the performative structure. We consider that there is a special transition directly connecting the root and output scenes that can be followed by any role.

This is the initial picture of a peformative structure. From this starting point, Once started the execution of a performative structure, it may evolve due to the dynamics of agents whenever these comply with the rules encoded by the specification. Thus, groups of agents may jointly start new scene executions, activities, that become part of the set of active scenes. Thereafter such active scenes may possibly become available to other agents (depending on whether they offer access states), which may apply for entering. But also some agents participating in active scenes may leave them either to start other scene executions, join in other active scenes or simply leave the performative structure. At the same time, some active scenes may finish and therefore the activity that they represent. Summarising, an agent participating in the execution of a performative structure devotes his time to jointly start new scene executions, to enter active scenes where the agent interacts with other agents, to leave active scenes to possibly enter other scenes, and finally to abandon the performative structure.

In what follows we formalise all the operations involved in the dynamics of a performative structure. In formal terms, we base our explanation on defining how scene and transition variables become bound and unbound. Thus, for instance, consider a group of agents intending to jump from one scene execution to another through a given transition. First, they must leave their scenes to enter the transition. Next, they must be bound to the transition variables. Afterwards, if their target scenes are ready to let them in they are allowed to traverse the transition towards their destinations. This process can be seen as the creation and destruction of bindings for the scenes and transition involved. Summarising, the flow of agents within a performative structure is achieved by doing and undoing bindings within scenes and transitions. An important aspect to consider in our discussion concerning both scene and transition variables is their local scope.

Analogously to the approach followed for scenes, we start by introducing the notions of performative structure execution descriptor (or performative structure descriptor for shorter). Thereafter we will complete our functional specification by defining a collection of operations over a performative structure execution.

Definition 3.4.2. Let $PS = \langle S, T, s_0, s_\Omega, E, f_L, f_T, f_E, C, \mu \rangle$ be a performative structure. We define an execution descriptor of PS as a tuple $\Pi_{PS} = \langle A, (\Sigma_s)_{s \in PS}, (\tau_t)_{t \in T} \rangle$ where

- $A \subseteq Agents$ is the finite set of agents participating in the performative structure execution;
- Σ_s stands for the set of all execution descriptors of scene s, denoting $(\Sigma_s)_{s \in PS}$ the set of all scene execution descriptors, henceforth referred to as the set of *active scenes* and denoted simply as Σ; and
- τ_t stands for the execution descriptor of transition t.

For notational purposes, henceforth Σ will stand for the set of all active scenes, and Σ_s and Σ_s^i will stand for the set of all executions of scene s and the i-th execution of scene s respectively. Within Σ we will distinguish two particular scene executions: Σ_{s_0} and Σ_{s_Ω} corresponding to the single executions of the root scene and output scene respectively.

Observe that our definition of performative structure contains transition descriptors whose formal definition and used is provided further below. Now, complementarily to our definition of performative structure descriptor, we introduce additional, auxiliary definitions that help us express the global state of a performative structure.

Thus the active roles played by an agent in a peformative structure execution are those played by the agent within each scene execution wherein it participates. Recall that given a scene execution $\Sigma_s^i = \langle A, \rho, \sigma_s, w, S, (V_{w_j})_{w_j \in W} \rangle$, the role played by an agent $a \in A$ can be readily obtained as $\rho(a)$. Next we define the marking of a performative structure execution as a function that returns the agents immersed in each scene execution, along with the role that each one plays.

Definition 3.4.3. We define the marking of a performative structure execution described by $\Pi_{ps} = \langle A, (\Sigma_s)_{s \in PS}, (\tau_t)_{t \in T} \rangle$ as a mapping $M : (\Sigma_s)_{s \in PS} \longrightarrow 2^{A \times Roles}$.

Then, considering also the local state of each scene execution, we can obtain in a straightforward manner the global state of a performative structure execution.

Definition 3.4.4. We define the state of a performative structure execution described by $\Pi_{ps} = \langle A, (\Sigma_s)_{s \in PS}, (\tau_t)_{t \in T} \rangle$ as a function $\Gamma : (\Sigma_s)_{s \in PS} \longrightarrow 2^{A \times Roles} \times W_{PS}$, where $W_{PS} = \bigcup_{s \in S} W_s$ is the set of all states for all the scene specifications composing the performative structure specification.

In what follows we take a formal view to specify how agents can legally move around within a performative structure.

Since agents jump out of and into scene executions by following different arcs, paths, and by traversing transitions, we start by explicitly stating how transitions can be traversed. It is fundamental to understand how transitions work since they let agents flow among scenes. Informally, we consider that a transition can be traversed when it becomes *enabled*. At that precise moment the transition can *fire* (or *occur*), and so the agents waiting within the transition may go through following the output arcs that lead to some scene executions lying on the output side. In formal terms, enabling a transition means that its variables are bound to a given set of agents which can leave the transition to either start or join in their destination scenes. Notice that we are assuming that any agent intending to either start or join in a scene execution must solicit his destination: the scene execution along with the role to be played. For now, we do not consider how agents make or refrain from their requests since our focus is a functional, not procedural, specification. We simply consider that such requests can be incorporated or refrained from a transition. The way requests for scenes are produced and handled is left to Chapter 4, where a computational model that fully captures our formal specification is presented.

Next we introduce the notion of transition execution along with the operations that allow to add/refrain requests for scenes to/from transitions.

Definition 3.4.5. Let $\Pi_{ps} = \langle A, (\Sigma_s)_{s \in PS}, (\tau_t)_{t \in T} \rangle$ be a performative structure descriptor. We define the execution descriptor of transition $t \in T$ as a tuple $\tau_t = \langle Ag, Q, \sigma \rangle$ where:

- Ag is a finite set of agents within the transition;
- Q is a finite set of tuples of the form $\langle a, s, r, \Sigma^i_{s'}, r' \rangle$ being a an agent identifier[7], r the role played by a in his source scene, and r' the role intended to be played in the target scene $\Sigma^i_{s'}$; and
- $\sigma : Var_t \longrightarrow Ag$ is the function binding transition variables in $Var_t = \bigcup_{(s,t) \in E} var((s,t))$ to agents in the Ag set.

Given an agent a that played the r role in his source scene, the addition of a request of a to enter a set of scene executions $\delta_{s'} \subseteq \Sigma_{s'}$ playing role r' can be specified by means of the *add_request* operation as follows:

$\mathbf{add\_request}(\tau_t, \mathbf{a}, \mathbf{s}, \mathbf{r}, \delta_{\mathbf{s'}}, \mathbf{r'})$
Pre-conditions. (c1) $r' \in authorised\_roles(a)$ (c2) $\nexists \; r_j \in Roles, \Sigma^i_{s'} \in \delta_{s'}$ such that $(a, r_j) \in M(\Sigma^i_{s'})$ (c3) $\exists x \in var((s,t)) \mid (x, r) \in f_E((s,t))$ and $(x, r') \in f_E((t, s'))$ **Post-condition.** $Q' = Q \cup \{\langle a, s, r, \delta_{s'}, r' \rangle\}$

[7]Henceforth we shall be using the terms *agent* and *agent identifier* indistinctly.

At this point, notice that it is possible to join several *threads* of a very same agent when he reaches a transition. Notice that a transition execution keeps a single instance of an agent. Thus an agent reaching a given transition following different arcs (and possibly playing different roles) must decide on progressing through the transition playing a single or multiple roles. In other words, within transitions an agent can rejoin several of his roles at will.
Refraining from an annotated request is accomplished by removing the request from the transition:

$\mathbf{refrain\_request}(\tau_t, \mathbf{a}, \mathbf{s}, \mathbf{r}, \delta_{\mathbf{s}'}, \mathbf{r}')$
Pre-conditions. $\langle a, s, r, \delta_{s'}, r' \rangle \in Q$ **Post-conditions.** $Q' = Q - \{\langle a, s, r, \delta_{s'}, r' \rangle\}$

Both operations produce a new transition execution descriptor $\tau_t' = \langle Ag, Q', \sigma \rangle$.

Considering all the definitions introduced so far, we are ready to consider the enablement and subsequent occurrence of transitions. Importantly, in the definitions that follow we exclusively consider sync/parallel transitions. Why? Notice that we could even do without *choice/choice*, *sync/choice* and *choice/parallel* transitions since they all can be expressed in terms of *sync/parallel* transitions. Figures 3.5, 3.6 and 3.7 illustrate this observation. The graphical specifications at the top of each box are translated below into equivalent graphical specifications using exclusively *sync/parallel* transitions. At this point, we may wonder about the use of having available such a collection of transition types. The answer is that they provide us with more expressiveness and contribute to the simplification and readability of resulting specifications.

Since transition enabling and occurrence can be best explained in terms of the binding of transition variables, we start out defining how transition variables and transitions themselves are bound to agent identifiers.

First, the definition of transition variable binding simply states that an agent identifier a can be bound to a transition variable x whenever the agent has requested to follow all the transition output arcs whose labels contain the variable. Depending on the type or arc followed, the agent must select a new scene execution (i), a single scene execution (ii), all current scene executions (iii) or a subset of scene executions (iv).

Definition 3.4.6. Let $\tau_t = \langle Ag, Q, \sigma \rangle$ be a transition execution and x a variable such that $(x, r) \in f_L((s,t))$ for some scene s and role r. We say that x can be bound to agent a if $\forall (t, s') \in E$ such that $(x, r') \in f_L((t, s'))$ $\exists (a, s, r, \delta_{s'}, r') \in Q$ satisfying that

(i) $type((t,s')) = * \Rightarrow \delta_{s'} = \emptyset$

(ii) $type((t,s')) = 1 \Rightarrow \| \delta_{s'} \| = 1$ and $\delta_{s'} \in \Sigma_{s'}$

(iii) $type((t,s')) = all \Rightarrow \delta_{s'} = \Sigma_{s'}$

(iv) $type((t,s')) = some \Rightarrow \delta_{s'} \subseteq \Sigma_{s'}$

When a given variable x is bound to some agent a, we write that $\sigma(x) = a$.

In order for a transition execution to be bound, all we require is that all transition variables are conveniently bound as stated by the following definition.

Definition 3.4.7. Let $\tau_t = \langle Ag, Q, \sigma \rangle$ be a transition execution for a transition $t \in T$ such that $Var_t = \{x_1, \ldots, x_n\}$. We say that τ_t is bound if each transition variable is bound to a different agent identifier so that all agents following the same arcs choose the same target scene executions and the conditions over the transition output arcs apply. More formally,

- $\forall x \in Var_t \quad \exists a \in Ag$ such that x is bound to a and $\forall (t,s) \in E, \forall x_i, x_j \in var((t,s)) \quad \delta_s^{\sigma(x_i)} = \delta_s^{\sigma(x_j)}$, where $\delta_s^{\sigma(x_i)}$ and $\delta_s^{\sigma(x_j)}$ stand respectively for the destination scenes requested by the agents bound to x_i and x_j; and
- $\forall (t,s) \in E \quad \models_{\{\sigma(x) | x \in Var_t\}} C((t,s))$.

We denote that a transition execution τ_t is bound to a set of agents A' as $B[Var_t] = A'$.

A bound transition is enabled when the agents bound to the transition variables can join in the requested target scene executions. In other words, the *bound* agents can be removed from the transition to be subsequently added to their destination scenes.

Definition 3.4.8. Let $\tau_t = \langle Ag, Q, \sigma \rangle$ be a transition execution whose variables $x_1, \ldots, x_n \in Var_t$ are bound to $a_1, \ldots, a_n \in Agents$ with roles $r'_1, \ldots, r'_n$ so that the bindings $\sigma(x_1) = a_1, \ldots, \sigma(x_n) = a_n$ apply. Let δ^{a_i} stand for the target scene executions of a_i so that $\langle a_i, s_i, r_i, \delta^{a_i}, r'_i \rangle \in Q$. We say that τ_t is enabled iff $\forall (t,s) \in E \quad \cup_{x \in var((t,s))} \delta_s^{\sigma(x)}$ are all accessible to $\{\sigma(x_i) : r_i \quad | \quad x_i : r_i \in f_L((t,s))\}$.

After a transition is enabled it *occurs*(it is *fired*). Then the agents bound to the transition variables enter their chosen destination scenes. Depending on the semantics of the type of path followed by the agents, they will either start a new scene —for *-arcs—, or incorporate into a single scene —for 1-arcs—, some —for some-arcs—, different scenes, or all scenes —for and-arcs— of a given scene node. Therefore, basically the agents will be either starting new scene executions or incorporating to existing scene executions.

The occurrence of a transition involves a change in the state of a performative structure because of the starting of scene executions and the incorporation of some agents to some scene executions.

The following operation formally defines the processes involved in firing a transition.

fire_transition$(\Pi_{\mathbf{PS}}, \tau_{\mathbf{t}})$

Pre-conditions.
(c1) $\tau_t = \langle Ag, Q, \sigma \rangle$ enabled by $A' = \{a_1, \ldots, a_n\} \subseteq Ag$ so that the bindings $\sigma(x_1) = a_1, \ldots, \sigma(x_n) = a_n$ apply and $\langle a_i, s_i, r_i, \delta^{a_i}, r'_i \rangle \in Q \ \forall a_i \in A'$

Operations.
$add\_agents(\Sigma^i_s, Ag, \rho) \ \forall (t,s) \in E$ such that $type((t,s)) \neq *, \forall \Sigma^i_s \in \delta$ where $Ag = \{\sigma(x) \mid (x,r) \in f_L((t,s))\}$, $\delta = \cup_{x \in var((t,s))} \delta_s^{\sigma(x)}$
and $\rho(\sigma(x)) = r$ where $(x,r) \in f_L((t,s))$

$start(s, Ag, \rho) \forall (t,s) \in E$ such that $type((t,s)) \neq *$
where $Ag = \{\sigma(x) \mid (x,r) \in f_L((t,s))\}$ and $\rho(\sigma(x)) = r$ where $(x,r) \in f_L((t,s))$

$refrain\_request(\tau_t, \langle a_i, s_i, r_i, \delta^{a_i}, r'_i \rangle) \ \forall a_i \in A'$

$unbind(x) \ \forall x \in Var_t$

Figures 3.8, 3.12 and 3.13 contain examples of graphical specifications containing the several types of transitions explained above that will be conveniently dissected along Sections 3.4.3 and 3.5.

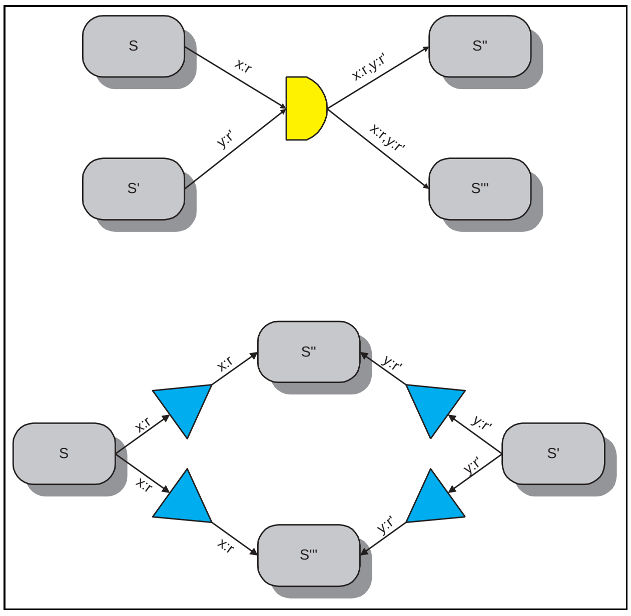


Figure 3.5: Transition Equivalences: Choice/Choice to Sync/Parallel

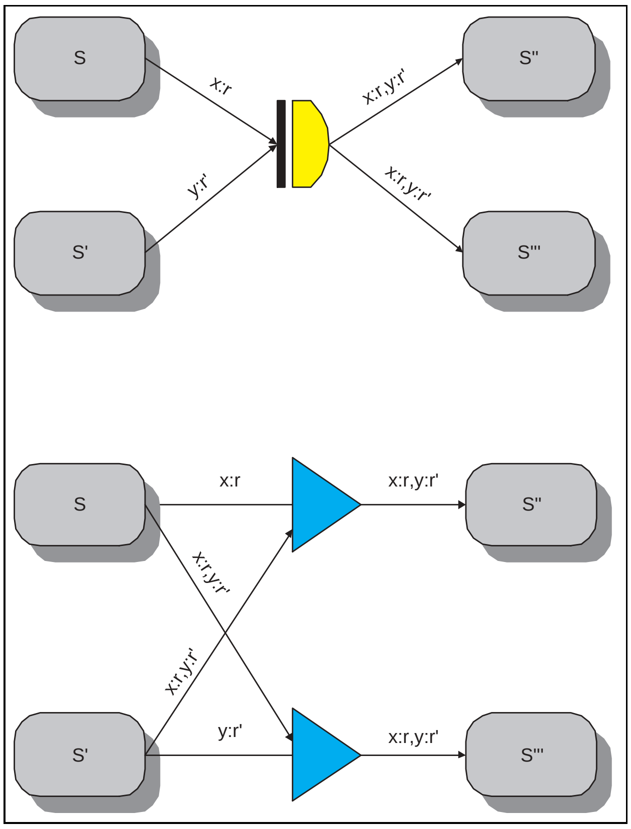


Figure 3.6: Transition Equivalences: Sync/Choice to Sync/Parallel.

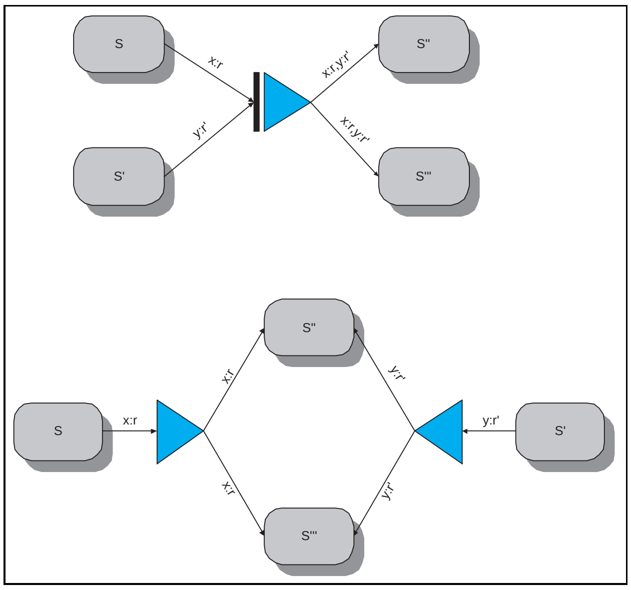


Figure 3.7: Transition Equivalences: Choice/Parallel to Sync/Parallel.

Examples

In what follows we attempt at illustrating how agents traverse the several types of transitions introduced above, and how they incorporate into other scenes depending on the types of paths connecting the transitions with the output scenes nodes. Figure 3.8 depicts some possible combinations of transitions and arcs for an electronic auction house inspired on the case study introduced in Chapter 1. Figure 3.8 (a) contains the specification of a choice point (represented as an *choice/choice* transition) for an agent playing either the role of buyer or seller. Connected to the output of the transition, there might be several Dutch auction scenes running simultaneously. An agent traversing the *choice/choice* transition following the arc of type *some* must select which of such scenes to enter. Below Figure 3.8 (b) shows how a buyer agent and a seller agent synchronise at the input of an *sync/parallel* transition, and then they traverse it together towards the very same Double auction scene that they must have previously chosen as destination. In this way we ensure that any Double auction scene is integrated by the same number of buyers and sellers. Recalling our definition of *sync/parallel* transitions, it is compulsory that both agents agree on their destination prior to traversing the transition, otherwise they remain waiting within the transition for any other, accompanying agent aiming at the same destination. Similarly in Figure 3.8 (c) agents also synchronise on the input side likewise those in Figure 3.8 (b), with the exception that they are allowed to choose several Double auction scenes as their destination. Finally, the case depicted by Figure 3.8 (d) shows

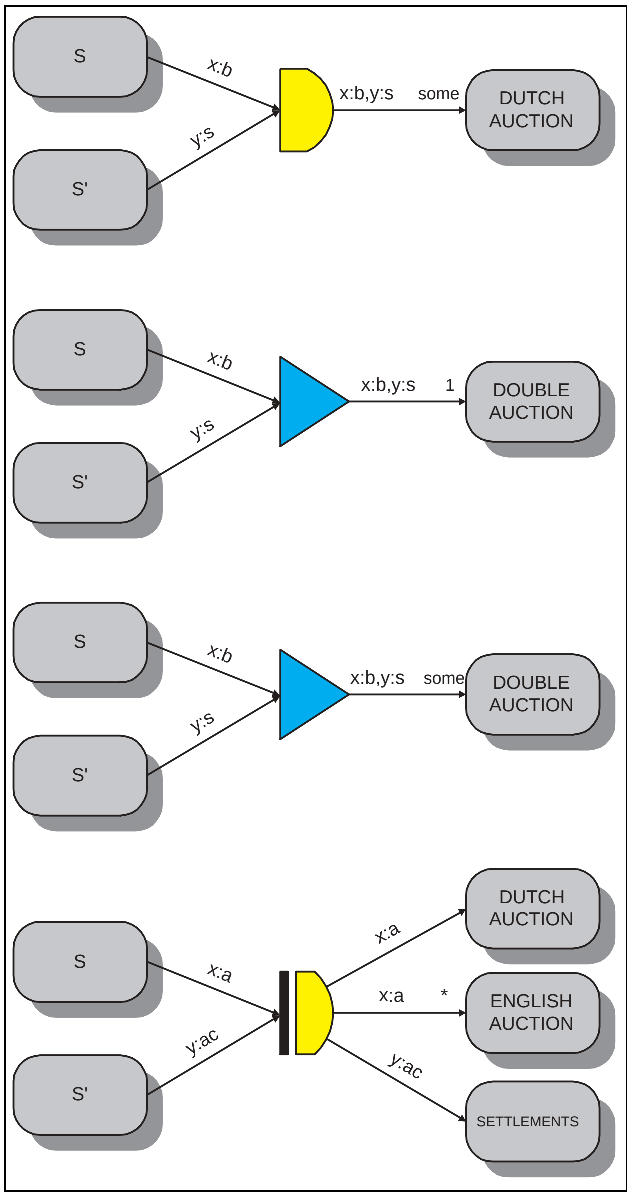


Figure 3.8: Examples

how an auctioneer (x:a) and an accountant (y:ac) synchronise on the input side of an *sync/choice* transition. Once traversed the transition they independently choose their destinations. Thus, the auctioneer might decide to either open a Dutch auction scene or an English auction scene, whereas there is no choice here available for the accountant.

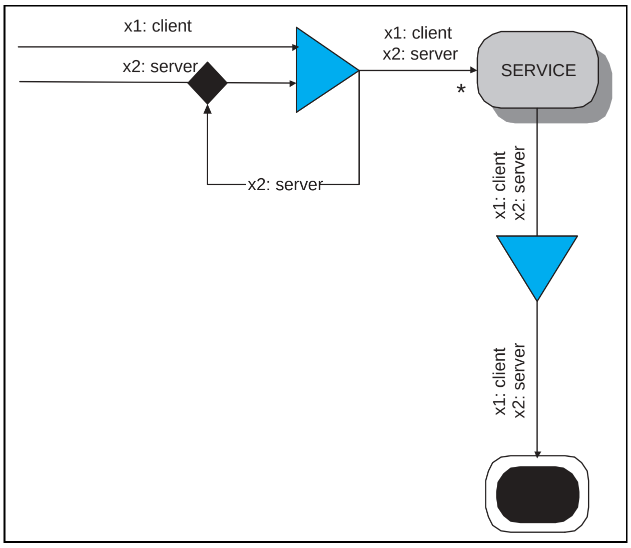


Figure 3.9: A Client-server Model

Now let us consider a radically different example which is depicted in Figure 3.9: a performative structure that models the behaviour of a multi-threaded client-server application. Here we will start understanding the use of *container* scenes. According to the specification having a client agent and a server agent they both proceed through the *sync/parallel* transition and start an instance of the scene labelled as *service* where the client agent is expected to receive a service by interacting with the server agent according to the scene protocol. Once the service is over, both client and server make it for an output scene. The interest of this example roots in the behaviour of the server agent. Notice that a container scene has been introduced before the transition for the agent server. Recall that the purpose of a container scene is to mark execution points to which we plan to eventually send back agents. From the point of view of a server agent bound to y, the container scene has no effect. The server understands that traversing the transition spawns two execution threads: one devoted to serve the client and another one which must loop back to the transition to wait for another client.

Lastly we present another example which shows a more complex pattern of iterative structure. Further on in Section 3.5 we shall find the usefulness of the very same structure when specifying the auctioning of goods in the fish market. Departing from the specification provided in Figure 3.10 we wonder whether we

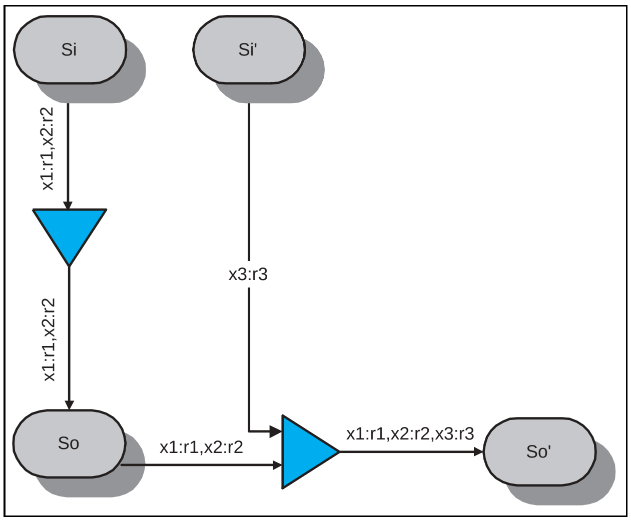


Figure 3.10: Loop Specification (1)

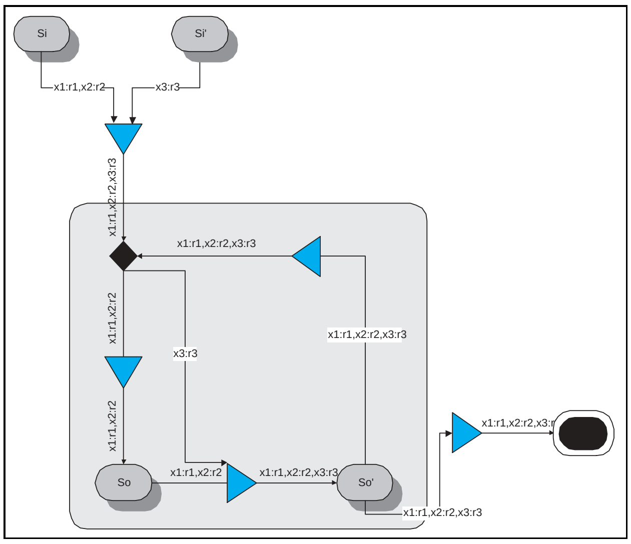


Figure 3.11: Loop Specification (2)

can make the roles $r1, r2, r3$ loop back after finishing in So'. The answer is no. Although we can add a new *sync/parallel* transition that links $r1, r2$ to So, the problem is that we cannot do the same for $r3$. Although we need to return $r3$ to the execution point prior to the traversing of the transition in order to meet again $r1$ and $r2$, we cannot make $r3$ go back to Si' because that obliges $r3$ to re-play in an activity that it intentionally left. Figure 3.10 proposes a solution by adding a *container* scene that marks the execution point where the iterative activity originates.

3.4.4 Electronic Institution

Up to now, we have considered how agents evolve within a performative structure execution by means of their illocutionary actions. Nonetheless, nothing has been said about how the consequences of their actions affect and condition their current and subsequent behaviour. Recall from Section 3.3.5 that agents commit to obligations that either restrict or enlarge their subsequent acting possibilities in the context of an electronic institution. We have not detailed yet how an institution keeps track of agents' pending obligations. Furthermore, neither have we made explicit how agents are allowed to join in or leave an institution.

Fundamentally, the execution of an electronic institution amounts to:

- determining how agents enter and leave;
- executing a performative structure; and
- keeping track of the obligations adopted by its players (agents).

On the one hand, accessing and leaving an institution will depend on the role hierarchy and the set of static separation of duties defined for the institution, along with agents' pending obligations. On the other hand, the determination of individual (per agent) obligations shall be based on the triggering of the institution's normative rules considering the illocutions uttered by each agent within each scene execution.

Next we depart from our definitions of electronic institution and performative structure execution to introduce our formal view of an electronic institution execution.

Definition 3.4.9. Given an electronic institution $E = \langle \mathcal{PS}, IR, \succeq, ssd, N_{PS}, ML \rangle$, we define an execution descriptor of E as a tuple $\Upsilon = \langle \Pi_{PS}, \kappa, H \rangle$ where

- $\Pi_{PS} = \langle A, (\Sigma_s)_{s \in PS}, (\tau_t)_{t \in T} \rangle$ an execution descriptor of the PS performative structure;
- $\kappa : A \longrightarrow 2^{ML}$ is a function that assigns to each agent a list of pending obligations[8]; and

[8]According to the defintion in Section 3.3.5.

- H is a list of triples $\langle \iota, \sigma, t \rangle$ —where ι stands for an illocution and σ and t stand respectively for the scene where the illocution was uttered and the time at which the utterance took place— containing the trace (history of utterances) of the institution.

In order to produce new agents' obligations, normative rules are triggered fed with the illocutions in H, i.e. the illocutions uttered in the institution. New obligations will be kept by κ on an individual basis. Recall from Section 3.3.5 that once an obligation is fulfilled by an agent it is disregarded as a pending obligation according to the following rule:

$$obliged(x, \phi, s) \wedge done(\phi, s) \Rightarrow \neg obliged(x, \phi, s)$$

Next, we define the operations needed to add agents to (grant access to agents) and remove agents from (allow agents to leave) an electronic institution.

Given an electronic institution execution descriptor Υ, an agent a can be added to an electronic institution with a set of roles R whenever a is not already participating in the performative structure execution and there are not conflicting roles in R. If so, the agent is authorised to play the roles in R along with the roles subsumed by the roles in the same set R.

$\mathbf{add\_agent(\Upsilon, a, R)}$
Pre-conditions. (c1) $a \notin A$ (c2) $\nexists (r, r') \in R \times R \quad \vert \quad (r, r') \in ssd$ **Post-conditions.** $A' = A \cup \{a\}$ $assigned\_roles(a) = R$ $active\_roles(a) = \emptyset$ $authorised\_roles(a) = R \cup \{r' \in Roles \quad \vert \quad \exists r \in R \quad and \quad r \succeq r'\}$

An agent a can be removed from an electronic institution execution Υ when it is no longer in any scene execution but only in the output scene and a has no pending obligations. In other words, agents are only allowed to leave when they have fulfilled all their pending obligations.

Recall that by looking at the marking of the performative structure execution we can readily find out the scene executions wherein agents are active.

$\mathbf{remove\_agent}(\Upsilon, \mathbf{a})$
Pre-conditions. (c1) $a \in A$ (c2) $\kappa(a) = \emptyset$ (c3) $\exists r \in authorised\_roles(a)$ such that $(a,r) \in M(\Sigma_{s_\Omega})$ (c4) $\nexists \Sigma_s^i \in (\Sigma_s)_{s \in PS}$ and $\Sigma_s^i \neq \Sigma_{s_\Omega}$ such that $(a,r) \in M(\Sigma_s^i)$ for some $r \in authorised\_roles(a)$ **Post-conditions.** $\Sigma'_{s_\Omega} = substract\_agents(\Sigma_{s_\Omega}, \{a\})$ $A' = A - \{a\}$

3.5 Practical Specifications

The purpose of this section is to practically illustrate how to specify electronic institutions employing the case studies presented in Chapter 1. For both cases, we concentrate on the graphical specification of their performative structures.

3.5.1 The Fish Market

Firstly, let us consider the modelling of the fish market as our initial case, whose complete graphical specification is shown in Figure 3.12. First of all, the boss starts a *registry* scene through which any agent, no matter the role, departing from the root scene must go. The *registry* scene is employed by the *boss* of the market to keep a directory (*white pages*) of all agents entering and eventually participating in the market. Notice that for this example we make the boss service agent registrations iteratively. When a registration process is completed, the boss may loop back to open a new registry scene execution or go ahead to start a closing scene execution to close down the market session.

When the rest of institutional agents arrive, namely a *buyer admitter*, an *auctioneer (auc)*, a *seller admitter (sa)*, a *buyers' admitter (ba)*, a *sellers' accountant (sac)*, and a *buyers' accountant (bac)*, they all synchronise to start the scene executions corresponding to the tasks that they will be responsible for,namely the admission of trading agents, the auctioning of goods and the accounting of transactions. But before that, they jointly start an execution of the *opening* scene, the starting point of the market activity. There the institutional agents acknowledge one another before engaging in the services that they are in charge of. Simultaneously (because of the *and* output followed), institutional agents also make for the transition which leads to the *closing* scene, where they will stand still until the boss of the market shows up to communicate the closing of the market session and start out the closing processes.

And then it is time to start the market activity involving trading agents. But first buyers and sellers arriving into the market are required to have their participation granted in the market session by successfully completing an ad-

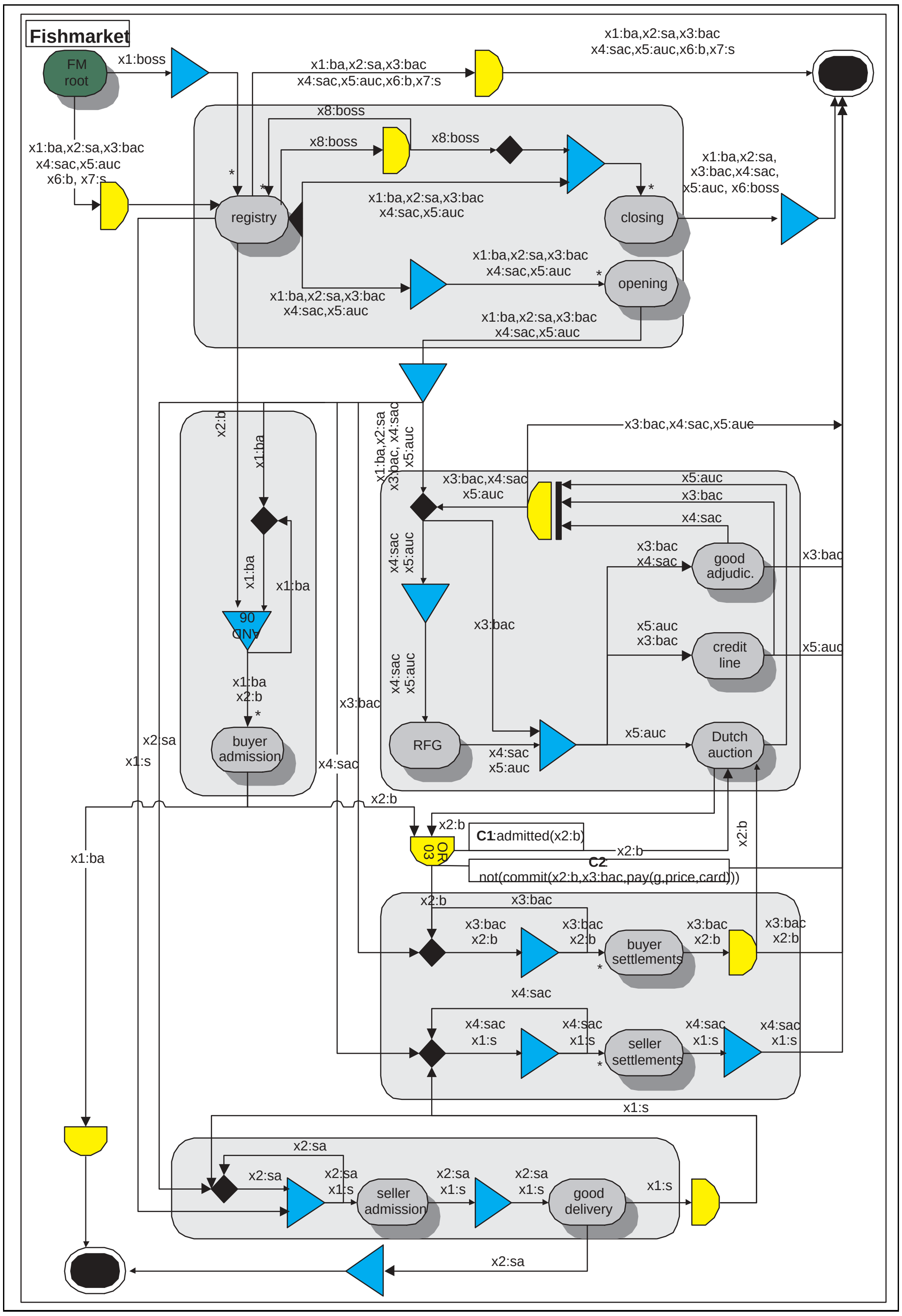


Figure 3.12: Graphical Specification of the Fishmarket Performative Structure.

mission process. In order to specify the admission of both buyers and sellers we make use of the client-server model depicted in Figure 3.9. When leaving the opening scene, both buyer admitter and seller admitter go over two containers which lead to the transitions where they wait for buyers and sellers to arrive. When a trading agent shows up, the admitter and the trading agent move forward towards an *admission* scene wherein the admission process takes place. In parallel, the admitter goes back to the transition leading to the admission scene to wait for other trading agents. As to buyer agents, when succeeding in the admission process, they can make its way to the *Dutch auction* scene, and otherwise they can only leave the institution following the path that leads to the output scene. Sellers, on their side, after their successful admission are allowed to deliver their goods at the *good delivery* scene, where the goods composing the lots that they deliver are typified by the seller admitter before conveying them to the auctioneer so as to be put at auction. If either the admission or the good delivery do not succeed, the seller agent is forced to leave the market by making its way to the output scene.

Say that there are already buyers ready to take part in the auctioning of goods and, very importantly, goods to be put at auction. In fact, the auctioneer does not start a *Dutch auction* scene execution till it receives some lot(s) of goods to put at auction from the sellers' accountant (sac) at the *RFG* (request for goods) scene. Having goods to sell, the auctioneer starts the *Duth auction* and, at the same time, the *credit line* scene and the *good withdrawal* scene. The purpose of the *credit line* scene, occupied by the auctioneer and the buyers' accountant, is to determine the validity of the bids submitted by the buyers participating in a bidding round within the *Dutch auction* scene. Thus the buyers' accountant must check whether every bid received by the auctioneer can be backed up by the bidder. If not, the buyers' accountant might either impose a penalty on the bidder, or even order its sending-off from the *Dutch auction* scene. Notice that buyers are allowed to update their credit at the *buyer settlements* scene. In parallel, the auctioneer communicates the result of every bidding round — either a sale or a withdrawal due to the reaching of the reservation price— to the sellers' accountant at the *good adjudication* scene.

When there are no more goods to be sold, all the institutional agents leave their scenes and synchronise at an *sync/choice* transition to either start out the auctioning of a new lot of goods or they simple leave the institution by jumping into the output scene. In general, the process may be started over unless the boss has already declared the closing of the market at the *closing* scene.

When completing their transactions, buyers and sellers have to respectively make effective the payment of their purchases and collect their earnings before leaving the market. On the one hand, sellers are not allowed to leave the market until all their goods have been auctioned unless the boss suspended the market in the middle of the auctioning of a lot of goods. Anyhow they can collect their earnings at the *sellers' settlements* scene. On the other hand, buyers are not allowed to leave the market till they make effective their pending payments and collect the acquired goods at the *buyers' settlements* scene.

Finally, when the boss declares the closing of the market all scene are closed and all agents synchronise to leave for the output scene.

3.5.2 The Conference Centre

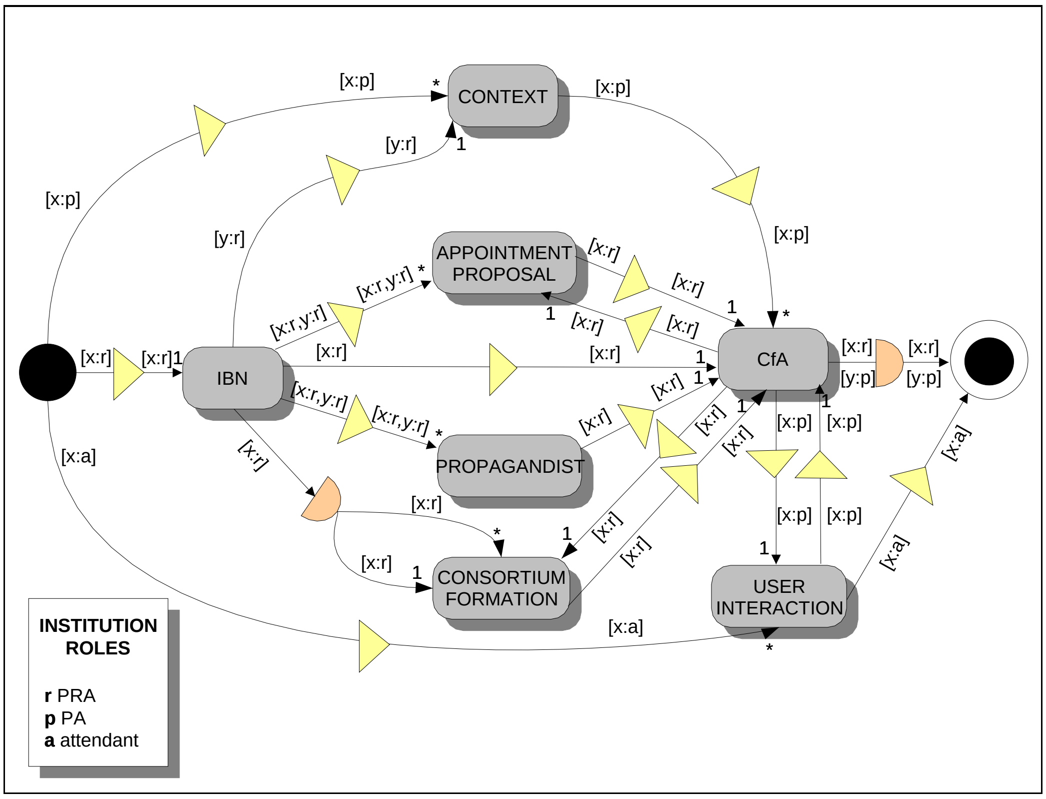


Figure 3.13: Graphical Specification of the Conference Centre Environment Performative Structure.

A conference takes place in a physical setting, the conference centre, where different activities take place in different locations by people that adopt different roles (speaker, session chair, organization staffer, etc.). During the conference, people pursue their interests moving around the physical locations, and engaging in different activities. In a moment in time people are physically distributed along the conference, possibly interacting with other people. We can easily think about the spatial proximity relations that exist among people in this physical space. However, if we think about an *informational space* where the past background and current interests of the conference attendees are represented, we could think of a new kind of *proximity* relation that is a function of the similarity among people's interests and backgrounds.

We can imagine software agents inhabiting the virtual space that take up some specific activities on behalf of some interest of an attendee in the conference. Specifically, a Personal Representative Agent (PRA) is an agent inhabiting the virtual space that is in charge of advancing some particular interest of a conference attendee by searching for information and talking to other software agents.

Attendees have to instruct their PRAs specifying a presentation (e.g. a topic and possibly a collection of related subtopics), an appearance (a collection of features describing the view an agent wants to offer to the other agents) for interacting with other PRAs, and a collection of tasks (e.g. meeting people, attending to talks, making appointments, etc) in which the PRA can participate for achieving the attendees' interests. The collection of tasks is provided by the Conference Centre definition as a set of scenes and roles in which a PRA can participate.

When a PRA has gathered information that may be of interest to an attendee it will try to *push* that information to that attendee. We can imagine another kind of agent the Personal Assistant (PA) agent which filters the information to an attendee. For each conference attendee there could be a PA agent in the virtual space that decides which information is more relevant for the attendee at any moment taking into account the attendee physical context. PRAs could only communicate to the attendee throw her PA. Attendees could receive those informations either by going to computer screens distributed along the conference building or by a wearable computer device, nicknamed *the parrot*, that runs a speech-generation program.

In the Conference Centre there are three types of agents: *Attendees*, *Personal Representative Agents* (PRAs), and *Personal Assistants*. Below we describe the roles that can take each of them:

Attendees are people participating in the conference. During the conference people can adopt different roles such as speaker, listener, session chair, panelist, workshop organizer, or conference staffer.

PRAs are software agents that are in charge of pursuing interests of attendees. Each attendee can launch several PRAs, each of them pursuing a different interest. During the conference a PRA can adopt different roles—and several at the same time if the PRA is participating in several scenes. The PRA roles are information gatherer, proposer, advertiser, information filterer, and context manager. These roles are determined by the roles that her attendee can adopt—i.e. only PRAs belonging to a workshop organizer, a demonstrator, or a both owner can adopt the role of advertisers.

PAs are software agents that are in charge of elaborating the information that is delivered to the attendees. Each attendee has one PA. The PA roles are context pusher and deliverer.

The Conference Centre supports six scenes. It is important to remark here that, in order to perform these scenes, the PRAs use *both* the conference centre ontology and the context information to infer the situation of the user. That is to say, knowing that the user is in a particular place, the current time, and the activity scheduled by the Conference for that place at that time, the PRA can infer the social activity in which the user is involved. We will briefly summarize the conference centre scenes:

• *Interest-Based Navigation* (IBN): The participants in an IBN scene are two PRAs adopting the role of information gatherers. In this scene PRAs establish initial conversations among them for estimating the interestingness of the attendees or conference events they represent. We say that in the IBN scene PRAs construct the *interest landscape*s of their attendees. The interest landscape holds all the information considered as useful for the interest of the attendee and is used and refined in the further scenes. When a PRA in a IBN scene assesses a conference event with a high interestingness valuation, the information is directly delivered to the attendee by means of the CfA scene. This delivery strategy was adopted for biasing the future decisions of the attendee.
In CAPIA advertisers, this task has been specialized for attracting persons that might be interested in the conference events (exhibition booths or conference sessions) they represent.

• *Appointment Proposal*: in this scene, using the interest landscape, the PRAs try to arrange an appointment between two attendees. First, PRAs negotiate a set of common topics for discussion (the meeting content). When they reach an agreement, PRAs negotiate about the appropriate meeting schedule. This scene will be detailed in the next section. Both PRAs participating in an appointment scene play the role of proposers.

• *Propagandist*: in this scene a PRA, adopting the role of advertiser, tries to attract to other PRAs, adopting the role of information filterer, to the conference event that represents (workshop, booth, or demonstration).

• *Competition for Attention* (CfA): in this scene PRAs, playing the role of proposers, try to convince PAs, playing the role of deliverers, about the importance of a given information. The CfA scene works like a sealed-bid auction. First at all the PA announces a new round and the time for accepting proposals. Then, the PRAs have to decide if are interested in sending a proposal and have to send it to the PA. When the time for proposals is exceeded, the PA informs about the decision and the scene begins again.

• *Context Scene*: in this scene a PRA, playing the role of information manager, can subscribe to a PA's context list for and track the physical context of a given attendee. When an attendee is physically near to another person, exhibition booth, or thematic session with similar interests,the PRA tries to inform the attendee by means of the CfA scene.

• *User Interaction* (UI): in this scene an attendee is informed by her PA about the information provided by the winner of the competition for attention scene. The attendee, can assess the information indicating a positive or negative feedback to the PA.

The performative structure holding all these previous scenes is given in Figure 3.13. Remark that PRAs can participate in all activities excepting the *UI* scene; PAs can participate in the *context*, *CfA*, and *UI* scene; and attendees only can participate in the *UI* scene.

Textual Representation

Finally, the resulting specification presented above can be translated into its XML-based textual counterpart in order to be manipulable by an agent, according to the *Document Type Definition* in Appendix A. Below we offer part of the specification corresponding to the conference centre environment:

```
<?xml version ="1.0" standalone = 'no'?>
<!DOCTYPE AMEI SYSTEM "http://www.iiia.csic.es/~jar/AMEI/dtd/amei.dtd">
<AMEI ID="comris-01">
      <DESCRIPTION>
      The Conference Centre Environment
      Agent-mediated Electronic Institution
      </DESCRIPTION>
      <ROLESET>
         <ROLEITEM>
            <ROLEID> pa </ROLEID>
            <DESCRIPTION> personal assistant </DESCRIPTION>
            <CARDINALITY> 100 </CARDINALITY>
         </ROLEITEM>
         <ROLEITEM>
            <ROLEID> pra </ROLEID>
            <DESCRIPTION> personal representative assistant </DESCRIPTION>
            <CARDINALITY> 1000 </CARDINALITY>
         </ROLEITEM>
         <ROLEITEM>
            <ROLEID> att </ROLEID>
            <DESCRIPTION> attender </DESCRIPTION>
            <CARDINALITY> 100 </CARDINALITY>
         </ROLEITEM>
      </ROLESET>
      <INHERITANCE>
         <INHITEM>
            <ROLEID> pa </ROLEID>
            <ROLEID> pra </ROLEID>
         </INHITEM>
      </INHERITANCE>
      <SSD>
         <ROLEPAIR>
            <ROLEID> pa  </ROLEID>
            <ROLEID> att </ROLEID>
         </ROLEPAIR>
         <ROLEPAIR>
```

```
            <ROLEID> pra  </ROLEID>
            <ROLEID> att </ROLEID>
         </ROLEPAIR>
      </SSD>
      <NPS>
      </NPS>
      <PS ID="ps-01" HREF="pslib.xml#ps-01"/>
</AMEI>
```

3.6 Summary

In agree with [Ferber and Gutknetch, 1998], although organisational design is widely admitted as a fundamental issue in multi-agent systems, social concepts have been introduced in a rather informal way. Hence the need for formally incorporating organisational terms and concepts into multi-agent systems.

In this chapter we have argued that open agent multi-agent systems can be effectively designed and constructed as electronic institutions. Then we have adopted a formal approach to specify electronic institutions. But why a formal technique? We believe that the following tip [Dav, 1993](page 215) answers the question:

> Use a formal technique when we cannot afford to have the requirements misunderstood.

And this is our case if we consider the high complexity of what we have identified as our main goal: the design and development of architecturally–neutral electronic institutions inhabited by heterogeneous (human and software) agents. In general, the presence of an underlying formal method underpins the use of structured design techniques and formal analysis, facilitating development, composition and reuse. Thus, we consider that the development of an electronic institution must be preceded by a precise specification that fully characterise the institutional normative environment (its rules). Some important advantages derive from the creation of specifications (models) of electronic institutions:

- An electronic institution model is a description of the modelled institution that can be used either as a specification or as a presentation. Such a model allows to investigate a new institution before being constructed. This possibility is particularly useful for institutions where design errors may jeopardise security or be expensive to correct.

- Graphical specifications are extremely easy to understand. They are similar to the informal diagrams employed by engineers and designers while designing, constructing and analysing a system.

- An electronic institution's specification offers an explicit description of both states and actions, in contrast to most description languages which describe either the states or the actions.

- The behaviour of an electronic institution model can be analysed, either by means of simulation or by means of more formal analysis methods.
- The process of creating the description and performing the analysis allows the modeller to gain a dramatically improved understanding of the modelled institution.

Chapter 4

Designing Agent-mediated Electronic Institutions

The main goal of this chapter is to present a computational model that arises from and fully captures the formalisation of electronic institution provided in Chapter 3. For this purpose, in Section 4.1 we start arguing on the capital importance of mediation for founding the realisation of infrastructures for agent-based systems in general and for electronic institutions in particular. Next, in Section 4.2 we introduce a particular type of facilitators, the so-called *interagents*, as autonomous software agents devoted to mediating the interaction among agents in an agent society in agent-based systems. In Section 4.3 we discuss how to extend this basic interagent model so that interagents can be also employed in the framework of an electronic institution. Furthermore, we also detail a computational model of *institutional agents*, i.e. the agents to which the institution delegates its services. Finally, we present how to fully realise an electronic institution based on interagents and institutional agents.

4.1 On the Need of Mediation

Agent-based systems provide a way of conceptualising sophisticated software applications that face problems involving multiple and (logically and often spatially) distributed sources of knowledge. In this way, they can be thought of as computational systems composed of several agents that interact with one another to solve complex tasks beyond the capabilities of an individual agent.

The development of agent-based systems can be regarded as a process composed of two well-defined, separate stages concerning respectively:

- *the agents' logics*: from the agents' inner behaviour (knowledge representation, reasoning, learning, etc.) to the agents' social behaviour responsible for high-level coordination tasks (the selection, ordering, and communication of the results of the agent activities so that an agent works effectively in a group setting [Lesser, 1998, Jennings, 1995]).

- *the agents' interactions* taking place at several levels: *content level*, concerned with the information content communicated among agents; *intentional level*, expressing the intentions of agents' utterances, usually as performatives of an agent communication language (ACL), e.g. KQML, FIPA ACL, etc.; *conversational level*, concerned with the conventions shared between agents when exchanging utterances; *transport level*, concerned with mechanisms for the transport of utterances; and *connection level*, contemplating network protocols (TCP/IP, HTTP, etc.).

Notice that nowadays a large amount of time during the development of agent-based systems is devoted to the management of the aforementioned time-consuming, complex agents' interactions. Since most of these appear to be domain-independent, it would be desirable to devise general solutions that can be subsequently reused for the deployment of other multi-agent systems. In this manner, and very importantly, agent engineers could exclusively concentrate on the domain-dependent agents' logics responsible of decision-making. And for this purpose, they can largely benefit from their previous experiences in the application of Artificial Intelligence techniques.

We defend that infrastructures for agent-based systems in general, and for electronic institutions in particular, can be successfully deployed making use of middle-agents, mediators, that take charge of the cumbersome interaction issues inherent to these types of systems. This observation motivates the introduction of a special type of facilitator, the so-called *interagent*, an autonomous software agent that mediates the interaction between each agent and the agent society wherein this is situated. Thus, interagents constitute the unique mean through which agents interact within a multi-agent scenario. And consequently, we regard interagents as the fundamental element for the construction of any agent-based system's infrastructure. Whereas agents' external behaviour is managed by interagents, their individual logics, knowledge, reasoning, learning and other capabilities are internal to agents.

As stated in Chapter 3, we consider electronic institutions as a particular case of agent-based systems. However, electronic institutions must be regarded as more restrictive environments. Thus, when realising electronic institutions we must not be exclusively concerned about guaranteeing the correct coordination among the participating agents when engaged in conversations (likewise any agent-based system), but also about guaranteeing that agents effectively abide by the institutional rules. From this follows that we must by some means enforce the institutional rules to the participants. For this purpose, we propose to add a new dimension to interagents and extend their basic functionality so that they can encapsulate and handle institutional rules. In this way, the interagent employed by each participant keeps track of the participant's current commitments and prohibitions, and furthermore either enables or disables paths in the performative structure depending on the participant's roles.

Notice that in fact we conceive interagents in electronic institutions as part of the institution itself. Therefore, interagents must be regarded as a kind of

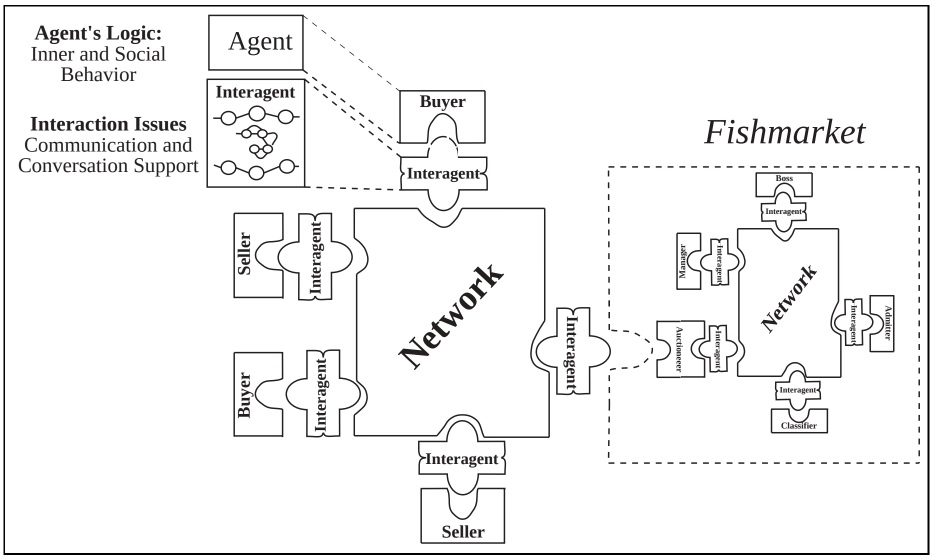


Figure 4.1: The Fish Market as an Agent-mediated Electronic Institution.

institutional agents. Interagents are customised by the institution before delivering them to participants, which are obliged to have their social behaviour mediated by interagents. As a major advantage of using interagents, we shall successfully manage to separate the rules (of the institution) from the players, an issue upheld as fundamental for institutional design [North, 1990].

Finally, there is the matter of realising the services provided by the institution. At this aim, we also propose to employ agents to which the institution delegates its services. We show that these institutional agents along with interagents will suffice to computationally realise an electronic institution. Observe the distinction that we draw between institutional agents in charge of services (service providers) and institutional agents in charge of mediation. Hence, by *agent-mediated electronic institution* (henceforth denoted as *AmEI* for shorter) we mean an electronic institution fully realised by means of agents.

In what follows firstly, in Section 4.2, we present a general-purpose model of interagents [Martín et al., 2000b] as the basis for building agent-based systems founded on the notion of *conversation protocols* [Martín et al., 2000a]. Secondly, in Section 4.3 we discuss the extensions required by an interagent in order to be effectively used for the development of *AmEIs*. Then we follow with the description of the other type of institutional agents, service providers, in order to finally detail how both types of institutional agents are effectively integrated to manage the operation of an electronic institution.

4.2 Agent-based Systems

4.2.1 Overviewing Interagents

Our proposal regarding the construction of infrastructures for agent-based systems relies upon two fundamental elements: *conversations* and *interagents*.

We view conversations as the means of representing the conventions adopted by agents when interacting through the exchange of utterances [Winograd and Flores, 1988, Barbuceanu and Fox, 1995] —"utterance suggests human speech or some analog to speech, in which the message between sender and addressee conveys information about the sender" [Parunak, 1996]. More precisely, such conventions define the *legal* sequence of utterances that can be exchanged among the agents engaged in conversation: what can be said, to whom and when. Therefore, *conversation protocols* are coordination patterns that constrain the sequencing of utterances during a conversation.

Interagents mediate the interaction between an agent and the agent society wherein this is situated. In fact, the management of conversation protocols must be considered as the *raison d'être* of interagents. Here we differentiate two roles for the agents interacting with an interagent: *customer*, played by the agent exploiting and benefiting from the services offered by the interagent; and *owner*, played by the agent endowed with the capability of dynamically establishing the policies that determine the interagent's behaviour. Needless to say that an agent can possibly play both roles at the same time. Moreover, several owners can even share an interagent's property (*collective ownership*), whereas several customers can make use of the same interagent (*collective leasing*).

In what follows we provide a detailed account of the full functionality of interagents —mostly from the point of view of customers— to illustrate how they undertake conversation management. An interagent is responsible for posting utterances of its customer to the corresponding addressee and for collecting the utterances that other agents address to its customer. This *utterance management* abstracts customers from the details concerning the agent communication language and the network protocol.

Each interagent owns a collection of relevant conversation protocols (CP) used for managing its customer conversations. When its customer applies for starting a new conversation with another agent the interagent instantiates the corresponding conversation protocol. Once the conversation starts, the interagent becomes responsible for ensuring that the exchange of utterances conforms to the CP specification.

Before setting up any conversation the interagent must perform a *CP negotiation* process with the interagent of the addressee agent. The goal of CP negotiation is to reach an agreement with respect to the conversation protocol to be used (see Section 4.2.2)[1]. Finally, an interagent allows its customer to hold

[1]Eventually the negotiation process might be extended to include a *CP verification* process as a subprocess. Such a process would check whether the CP to be used verifies the necessary conditions (liveness, termination, deadlock and race condition free) for guaranteeing the correct evolution of an interaction.

several conversations at the same time. This capability for *multiple conversations* is important because conversations with any number of participants can be built as a collection of simultaneous CP instances. In other words, the agent views a conversation as involving n participants while its interagent views such conversation as a collection of simultaneous dialogues represented as multiple CP instances.

For explanatory purposes, in this chapter we consider again the example of the auction scene, the main activity in the fish market, already introduced in Chapter 3. Recall that the auctioning of goods is governed by an auctioneer under the rules of the so-called *downward bidding protocol* (DBP). When the auctioneer opens a new *bidding round* to auction a good among a group of agents, he starts quoting offers downward from the chosen good's starting price. For each price called, three situations might arise during the open round: i) several buyers submit their bids at the current price. In this case, a collision comes about, the good is not sold to any buyer, and the auctioneer restarts the round at a higher price; ii) only one buyer submits a bid at the current price. The good is sold to this buyer whenever his credit can support his bid. Whenever there is an unsupported bid the round is restarted by the auctioneer at a higher price, the unsuccessful bidder is punished with a fine, and he is expelled out from the auction room unless such fine is paid off; and iii) no buyer submits a bid at the current price. If the reserve price has not been reached yet, the auctioneer quotes a new price which is obtained by decreasing the current price according to the price step. If the reserve price is reached, the auctioneer declares the good *withdrawn* and closes the round.

The conventions that buyer agents have to comply with when interacting with the auctioneer during a bidding round are represented by means of a conversation protocol managed by the so-called *trading interagents*, a special type of interagents owned by the institution, the auction house, but used by trading agents.

4.2.2 Conversation Protocols

A conversation protocol (CP) defines a set of legal sequences of utterances that can be exchanged between two agents holding a conversation. We model and implement a CP as a special type of Pushdown Transducer (PDT), which can be seen in turn as a combination of a Finite-State Transducer (FST) and a Pushdown Automaton (PDA):

- An FST is simply a Finite State Automaton (FSA) that deals with two tapes. To specify an FST, it suffices to augment the FSA notation so that labels on arcs can denote pairs of symbols [Roche and Schabes, 1997];

- A PDA is composed of an input stream and a control mechanism —like an FSA— along with a stack on which data can be stored for later recall [Aho and Ullman, 1972, Brookshear, 1989].

Therefore, a PDT is essentially a pushdown automaton that deals with two tapes. A PDA can be associated to a PDT by considering the pairs of symbols on the arcs as symbols of a PDA. The choice of PDTs as the mechanism for modelling CPs is motivated by several reasons. First, analogously to other finite-state devices a few fundamental theoretical basis make PDTs very flexible, powerful and efficient [Roche and Schabes, 1997]. They have been largely used in a variety of domains such as pattern matching, speech recognition, cryptographic techniques, data compression techniques, operating system verification, etc. They offer a straightforward mapping from specification to implementation. The use of pairs of symbols allows labelling arcs with (p/d) pairs (where p stands for a performative, and d stands for a predicate symbol[2]). This adds expressiveness to the representation of agent conversations. PDTs, unlike other finite state devices, allow us to store, and subsequently retrieve, the contextual information of ongoing conversations.

In the rest of this section we explain CPs in depth, whose management is the main task of interagents. We start introducing a conceptual model of CPs. Next, the formalism underpinning our model is presented in order to provide the foundations for a further analysis concerning the properties that CPs must exhibit. Finally, the way of negotiating, instantiating, and employing CPs is also explained.

Conceptual Model

Conceptually, we decompose a CP into the following elements (see Figure 4.3): a finite state control, an input list, a pushdown list, and a finite set of transitions.

First, the *finite state control* contains the set of states representing the communication state of the interagent's customer during an ongoing conversation. We shall distinguish several states based on the communicative actions that they allow:

- *send*, when only the utterance of performatives is permitted;
- *receive*, when these can be only received; and
- *mixed*, when both the utterance and reception are feasible.

The utterances heard by an interagent during each conversation are stored into an *input list*. In fact, this input list is logically divided into two sublists: one for keeping the utterances' performatives, and another one for storing their predicates. It is continuously traversed by the interagent in search of a (p/d) pair which can produce a transition in the finite state control. Notice that the continuous traversing of the input list differs from the type of traversing employed by classic FSAs whose read only input tapes are traversed from left to right (or the other way around).

Let us consider a particular CP related to an ongoing conversation. We say that an utterance is *admitted* when it is heard by the interagent and subsequently

[2]Henceforth simply predicate.

stored into the input list. Admitted utterances become *accepted* when they can cause a state transition of the finite state control. Then they are removed from the input list to be forwarded to the corresponding addressee, and thereupon the corresponding transition in the finite state control takes place. Notice that all utterances are firstly admitted and further on they become either accepted or not. Therefore, the input list of each CP keeps admitted utterances that have not been accepted for dispatching yet. From now on, these criteria will be taken as the message sending and receiving semantics used in Section4.2.2.

The context of each conversation can be stored and subsequently retrieved thanks to the use of a *pushdown list*. Such context refers to utterances previously sent or heard, which later can help, for example, to ensure that a certain utterance is the proper response to a previous one. For instance, an utterance in the input list —represented in a KQML-like syntax— will be processed only if its sender, receiver and the value of the keyword *:in-reply-to* match respectively the receiver, sender and value of the keyword *:reply-with* of the topmost message on the pushdown list.

Finally, each transition in the *finite set of transitions* of a CP indicates: i) what utterance can be either sent or received to produce a move in the finite state control (henceforth we shall employ the notion of *polarity* of an utterance to express whether it can be either sent or received); and ii) whether it is necessary to store (push) or retrieve (pop) the context using the pushdown list.

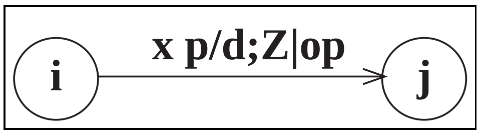


Figure 4.2: Transition

Each arc of the finite state control is labelled by one or more transition specifications. The structure of a transition specification is shown in Figure 4.2: a transition from state i to state j occurs whenever an utterance with polarity x, performative p, and predicate d is found in the input list and the state of the pushdown list is Z. In such a case, the chain of stack operations indicated by op is processed. In order to fully specify a transition, the following definitions and criteria must be observed:

- the polarity of an utterance u, denoted as $polarity(u)$, can take on one of two values: *+*, to express that the utterance is sent, or -, to indicate that the utterance is received. From this, for a given CP p we define its *symmetric view* $\overline{p}$ as the result of inverting the polarity of each transition;
- the special symbol *p/** represents utterances formed by performative p and any predicate, whereas the special symbol **/d* stands for utterances formed by any performative and predicate d;
- when several transitions $p_1/d_1;Z|op, \ldots ,p_n/d_n;Z|op$ label an arc, they can be grouped into a compound transition as $(p_1/d_1|\ldots|p_n/d_n);Z|op$;

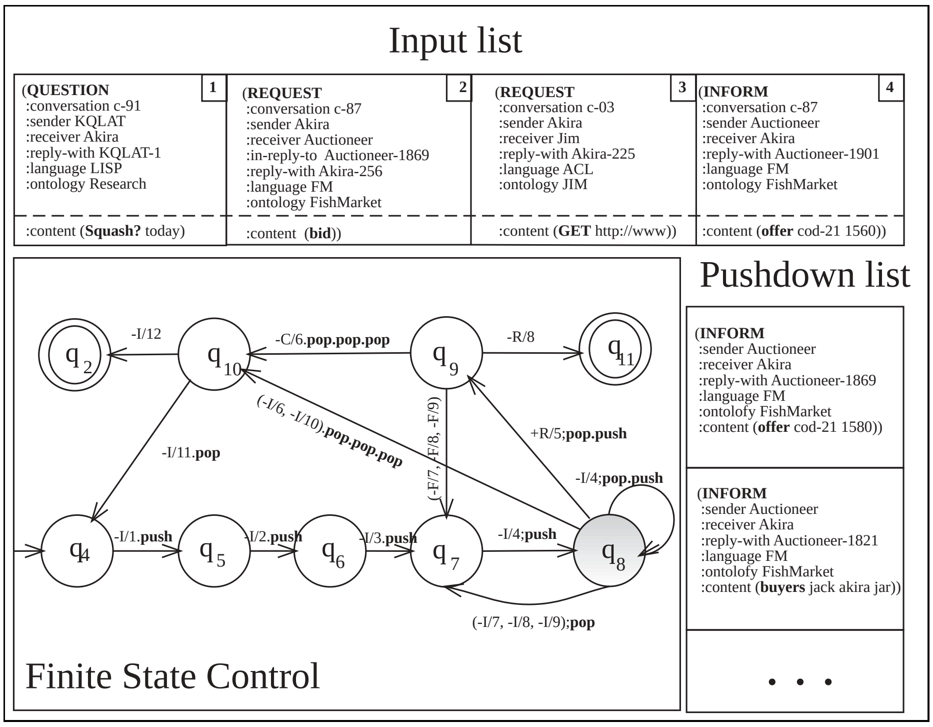
- our model considers the following stack operations:

  push pushes the utterance selected from the input list onto the stack;

  pop pops the stack by removing the topmost utterance; and

  nop leaves the stack unchanged. Usually this operation is omitted in specifying transitions;

- when the state of the pushdown list is not considered for a transition, it is omitted in the transition specification. In CPs, ϵ-moves, i.e. moves that only depend on the current state of finite state control and the state of the pushdown list, are represented using the transition specification Z|op.



```
(QUESTION
 :conversation c-91
 :sender KQLAT
 :receiver Akira
 :reply-with KQLAT-1
 :language LISP
 :ontology Research
 :content (Squash? today)
```

```
(REQUEST
 :conversation c-87
 :sender Akira
 :receiver Auctioneer
 :in-reply-to Auctioneer-1869
 :reply-with Akira-256
 :language FM
 :ontology FishMarket
 :content (bid))
```

```
(REQUEST
 :conversation c-03
 :sender Akira
 :receiver Jim
 :reply-with Akira-225
 :language ACL
 :ontology JIM
 :content (GET http://www))
```

```
(INFORM
 :conversation c-87
 :sender Auctioneer
 :receiver Akira
 :reply-with Auctioneer-1901
 :language FM
 :ontology FishMarket
 :content (offer cod-21 1560))
```

```
(INFORM
 :sender Auctioneer
 :receiver Akira
 :reply-with Auctioneer-1869
 :language FM
 :ontolofy FishMarket
 :content (offer cod-21 1580))
```

```
(INFORM
 :sender Auctioneer
 :receiver Akira
 :reply-with Auctioneer-1821
 :language FM
 :ontolofy FishMarket
 :content (buyers jack akira jar))
```

Figure 4.3: Partial view of the CP DBP used by trading interagents in the Fishmarket.

For instance, Figure 4.3 depicts the CP handled by a trading interagent to allow its customer (buyer agent *Akira*) to participate in a bidding round open by the *auctioneer* agent. Notice that the performatives of the utterances considered in the figure follow the syntax of Table 4.1, and the predicates within such utterances belong to the list in Table 4.2.

Several remarks apply to this simple example. First, notice that this specification respresents a buyer agent's view of the auction scene. This fact explains the apparent mismatch between this specification and the auction scene specification provided in Chapter 3. In addition to this, the example also intends to illustrate the use of the pushdown list: i) for saving the state of the bidding round (round number, good in auction, list of buyers, etc.); and ii) for ensuring that whenever a request for bidding is dispatched, the bid conveyed to the auctioneer will be built by recovering the last offer pushed by the trading interagent onto the pushdown list. From this follows that we also contemplate decision-making capabilities for interagents in the course of a mediated conversation. For the moment we put this consideration aside to be brought up back in Section 4.3.3, and subsequently exemplified in Chapter 5. Finally, though the specification in Chapter 3 was synchronous —the auctioneer asked the buyers in turns whether they accepted to bid at the current offer price—, the specification in Figure 4.3 is built considering a more realistic asynchronous version of the scene. In this way, each buyer continuously receives the offers called by the auctioneer till some event comes about.

ID	**Speech Act**	**Description**
Q	QUESTION	SOLICIT the addressee to INFORM the sender of some proposition
R	REQUEST	SOLICIT the addressee to COMMIT to the sender concerning some action
I	INFORM	ASSERT + attempt to get the addressee to believe the content
C	COMMIT	ASSERT that sender has adopted a persistent goal to achieve something relative to the addressee's desires
F	REFUSE	ASSERT that the sender has not adopted a persistent goal to achieve something relative to the addressees desires
A	ACKNOWLEDGE	SOLICIT the addressee to ACKNOWLEDGE the sender the reception of some performative

Table 4.1: Types of performatives following Cohen and Levesque [Cohen and Levesque, 1995] ([Parunak, 1996])

#Message	**Predicate**	**Parameters**
1	*open_round*	*round_number*
2	*good*	*good_id good_type starting_price resale_price*
3	*buyers*	$\{buyerlogin\}^*$
4	*offer*	*good_id price*
5	*bid*	
6	*sold*	*good_id buyerlogin price*
7	*sanction*	*buyerlogin fine*
8	*expulsion*	*buyerlogin*
9	*collision*	*price*
10	*withdrawn*	*good_id price*
11	*end_round*	*round_number*
12	*end_auction*	*auction_number*

Table 4.2: DBP Trading Interagent Predicates

Formal Definition

It is time to formally capture the conceptual model introduced above to be able to reason, later on, about the properties that we must demand from CPs. Therefore, formally we define a conversation protocol as an 8-tuple :

$$CP = \langle Q, \Sigma_1, \Sigma_2, \Gamma, \delta, q_0, Z_0, F \rangle$$

such that:

- Q is a finite set of state symbols that represent the states of the finite state control.
- Σ_1 is a finite alphabet formed by the identifiers of all performatives that can be uttered during the conversation.
- Σ_2 is a finite input list alphabet composed of the identifiers of all predicates recognized by the speakers.
- Γ is the finite pushdown list alphabet.
- δ is a mapping from $Q \times \Sigma_1 \times \Sigma_2 \times \Gamma^*$ to $Q \times \Gamma^*$ which indicates all possible transitions that can take place during a conversation.
- $q_0 \in Q$ is the initial state of a conversation.
- $Z_0 \in \Gamma$ is the start symbol of the pushdown list.
- $F \subseteq Q$ is the set of final states representing possible final states of a conversation.

CPs only contemplate a finite number of moves from each state that must belong to one of the following types:

moves using the input list these depend on the current state of the finite state control, the performative and predicate of a message into the input list and the state of the pushdown list. For instance, the move expressed by the following transition allows a trading interagent to convey to the auctioneer a request for bidding received from its customer (a buyer agent) whenever an offer, received from the auctioneer, has been previously pushed upon the pushdown list

$$\delta(q_8, +\mathsf{REQUEST}, \mathsf{bid}, \mathsf{offer}Z) = (q_9, \mathsf{bid(price)}Z);$$

ϵ-moves these depend exclusively on the current state of the finite state control and the state of the pushdown list. ϵ-moves are specifically employed to model and implement time-out conditions within CPs, so that interagents can handle expired messages and automatically recover from transmission errors

$$\delta(q_9, e, e, \mathsf{bid}Z) = \{(q_8, Z)\}.$$

It should be noted that at present we are only interested in deterministic CPs (DCP): a CP is said to be deterministic when for each state $q \in Q$, $p \in \Sigma_1$, $d \in \Sigma_2$ and $Z \in \Gamma^*$ there is at most one possible move, that is $|\delta(q, p, d, Z)| \leq 1$.

Instantiation

CPs can be defined declaratively and stored into conversation protocol repositories open to interagents. Each CP is identified by a unique name anonymously set by the agent society. When an interagent is requested by its customer to start a conversation with another agent it must retrieve the appropriate CP from a conversation repository, and next proceed to instantiate it. In fact, the CP must be instantiated by each one of the interagents used by the agents intending to talk.

We say that a CP becomes fully instantiated when the interagent creates a CP instance, i.e. after setting the values for the following attributes:

CP name CP class to which the CP instance belongs;

speakers identifiers of the agents to engage in conversation. Notice that we shall restrict a CP instance to consider exactly two speakers: the agent that proposes to start a conversation, the *originator*, and his speaker, the *helper*. In spite of this limitation, we are not prevented from defining multi-agent conversation as a composition of multiple CP instances;

conversation identifier a unique identifier created by the originator;

polarity this property indicates how to instantiate the polarity of each transition of the CP: if the instance polarity is positive, each transition is instantiated just as it is, whereas if it is negative each transition polarity is inverted. Notice that the helper must instantiate the symmetric view of the originator's CP in order to ensure protocol compatibility as shown in Section 4.2.2.

transport policies such as time-out or maximum time allowed in the input list. These can be altered during the course of a conversation whereas the rest of attributes of a CP instance remain fixed. The use of transport policies require to extend the CP being instantiated with ϵ-moves that enable to roll back to previous conversation states.

Then, and considering the definition above, from the point of view of interagents the conversations requested to be held by its customer can progress through several states:

pre-instantiated after retrieving the requested CP;

instantiated a CP becomes instantiated when the originator creates a CP instance, and subsequently asks the helper for starting a new conversation accepting the terms (attributes) of the interaction expressed by the CP instance;

initiated this state is reached when both speakers agree on the value of the attributes of a new conversation, as a result of the negotiation phase described in Section 4.2.2;

running state attained after the first utterance;

finished a conversation is over whenever either the final state of the CP is reached, the helper refuses to start it, or an unexpected error comes about.

Figure 4.4 shows the evolution of the states of a CP by means of a finite state diagram.

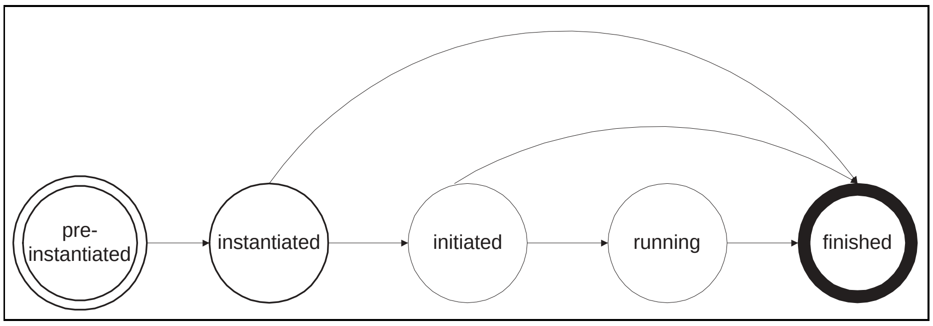


Figure 4.4: CP States.

Instantaneous Description

An instantaneous description describes the state of a CP instance at a particular time. An *instantaneous description* of a CP instance p is a 7-tuple:

$$\langle o, h, p, t, q, l, \alpha \rangle$$

such that:

o is the originator agent; h is the helper agent; p is the polarity of the CP instance; t is the current setting of transport policies; q is the current state of the finite state control; i represents all utterances currently kept by the input list; and α is the current state of the pushdown list.

Figure 4.3 depicts the instantaneous description for an instance of the CP DBP employed by a trading interagent to allow its customer (a buyer agent) to participate in a bidding round open by the auctioneer agent. We identify buyer *Akira* as the originator, the *auctioneer* as the helper, and the coloured node q_8 as the state of the finite state control.

A deterministic CP has at most —without taking into account possible ϵ-moves for dealing with expired utterances— one possible move from any instantaneous description. However, continuously traversing the input list in search of an utterance that causes a transition can lead to *race conditions*. For instance, in the CP instance of Figure 4.3 the second and fourth utterances can originate a race condition since both utterances can cause a move in the finite state control. Thus, it is necessary to define criteria for deciding which utterance must be accepted as we show in the next section.

Compatibility Semantics

In order to ensure the correct exchange of utterances during a conversation, there are important, desirable properties such as termination, liveness, and lack of deadlocks and race conditions that CPs must verify. In what follows we concentrate exclusively on the two last properties, since to guarantee both termination and liveness it suffices to assume that every CP whose set of final states is non-empty does not remain forever in the same state.

First we formulate our notion of CP compatibility, following the notion of protocol compatibility proposed by Yellin and Strom in [Yellin and Strom, 1997], whose work provides, in fact, the foundations of our analysis.

CPs can be assigned two different semantics: asynchronous or synchronous. Although asynchronous semantics may facilitate implementation, it makes generally harder reasoning about certain properties, such as *deadlock* which has proven undecidable under these semantics [Yellin and Strom, 1997]. On the contrary, under synchronous semantics, reasoning about such properties is easier, though an implementation technique must map these semantics to a particular implementation.

For our purposes, we have opted for a synchronous semantic for CPs and, consequently, for devising the adequate mechanisms for implementation. Such a type of semantic requires to assume that a speaker can send an utterance to the other speaker only if that is willing to receive the utterance. Therefore, we must assume that the finite state controls of both CP instances advance synchronously, and hence that sending and receiving an utterance are atomic actions. Upon this assumption, Yellin and Strom introduced the following notion of compatibility of protocols: "Protocols p_1 and p_2 are *compatible* when they have no *unspecified receptions*, and are *deadlock free*". On the one hand, in terms of CPs, the absence of unspecified receptions implies that whenever the finite state control of a CP instance corresponding to one of the speakers is in a state where an utterance can be sent, the finite state control of the CP instance of the other speaker must be in a state where such utterance can be received. On the other hand, deadlock free implies that the finite state control of both CP instances are either in final states or in states that allow the conversation to progress. Interestingly, the authors prove the existence of an algorithm for checking protocol compatibility. By applying such algorithm, it can be proved that under synchronous semantics a CP instance p and its symmetric view $\overline{p}$ are always compatible. From this follows that two CP instances are compatible if both belong to the same CP class, and both have the same speakers but different polarity. In this way, the complete agreement on the order of the utterances exchanged between the speakers is guaranteed.

Concerning the implementation, observe that the atomicity of sending and receiving utterances cannot be guaranteed. Nonetheless, when compatible CP instances lack mixed states, the unspecified receptions and deadlock free properties can still be guaranteed. But the presence of mixed states can lead to race conditions that prevent both speakers from agreeing on the order of the messages, and therefore unexpected receptions and deadlocks might occur. In order to avoid such situations, low-level synchronization mechanisms must be provided in accordance with the interpretation given to message sending and receiving in Section 4.2.2.

For this purpose, we consider that the speakers adopt different, conflicting roles —either *leader* or *follower*— when facing mixed states. The leader will decide what to do, whereas the follower will respect the leader's directions. By extending CP instances with a new attribute —which can take on the values *leader* or *follower*— we include the role to be played by each speaker in front of mixed states. Besides, we consider two special symbols —TRY and OK— that alter the interpretation of message sending. Thus, on the one hand when a message is sent under the TRY semantics, the receiver tries to directly accept the message without requiring previous admission. On the other hand, the OK symbol confirms that the TRY was successful. Then given a CP $\langle Q, \Sigma_1, \Sigma_2, \Gamma, \delta, q_0, Z_0, F \rangle$ for each mixed state $x \in Q$ the CP instance corresponding to the follower will be augmented in the following way:

$\forall p \in \Sigma_1, \forall d \in \Sigma_2, \forall Z \in \Gamma^*, \forall Z' \in \Gamma^*, \forall q \in Q$ such that $\exists \delta(x, +p, d, Z) = \{(q, Z')\}$ then a new state $n \notin Q$ will be added $Q = Q \cup \{n\}$ along with the following transitions:

(i) $\delta(x, +TRY(p), d, Z) = \{(n, Z)\}$

(ii) $\delta(n, -OK, e, Z) = \{(q, Z')\}$

(iii) $\forall p' \in \Sigma_1, \forall d' \in \Sigma_2, \forall Z'' \in \Gamma^*$ then $\delta(n, -p', d', Z'') = \delta(x, -p', d', Z'')$

Therefore, when the leader is in a mixed state and sends an utterance, the follower admits it and subsequently accepts it. Conversely, when the follower is in a mixed state and sends an utterance, the leader, upon reception, determines if it is admitted or not. Figure 4.5 illustrates how the CP instance on the left should be augmented to deal with the mixed state q_8. However, we will not unveil yet how the whole bidding process proceeds. We hold the explanation till Chapter 5 where we offer a thorough dissection of an agent-mediated electronic auction house where it is currently employed.

Notice that the conflict role played by each speaker must be fixed before the CP becomes completely instantiated. This and other properties of CP instances need be negotiated by the speakers as explained along the next section.

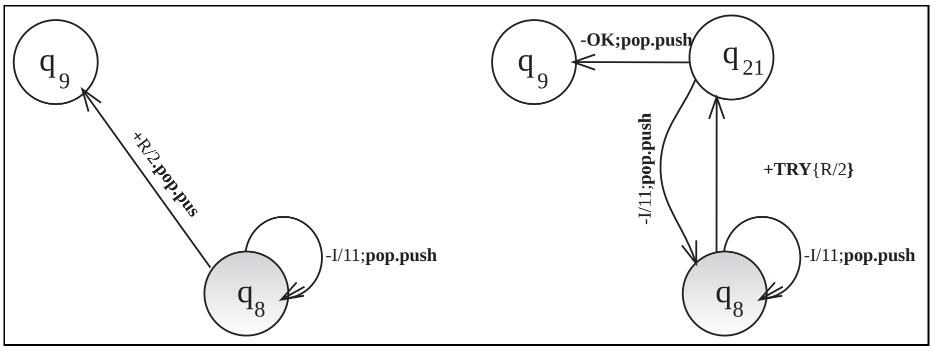


Figure 4.5: Augmented CP instance

CP Negotiation

In Section 4.2.2 we introduced the several attributes of a CP instance that have to be fixed before the conversation between the speakers becomes fully instantiated, and subsequently started. The value of each one of these attributes has to be mutually agreed by the speakers in order to guarantee conversation soundness. For this purpose, interagents have been provided with the capability of negotiating such values by means of the so-called *handshake* phase, following the directions of their customers. During this process, the initial connection between the originator and the helper is established, and next the originator conveys its multi-attribute proposal (the set of attributes' values) to the helper. Then, we distinguish two models of negotiation based on the helper's response: one-step and two-step negotiation.

In one-step negotiation the helper either automatically accepts or refuses the originator's proposal. The following sequence depicts a typical exchange for this model of negotiation, where q values indicate the degree of preference over each proposal.

```
originator:  START
helper:      OK
originator:  CP/DBP; id=21; polarity=+; leader=me; q=0.7
helper:      CP/DBP; id=21; polarity=-; leader=you; q=0.3
originator:  CP/DBP; id=21; polarity=+; leader=me
```

In this example the helper accepts the second proposal from the originator.

In two-step negotiation, instead of directly accepting or refusing the originator proposal, the helper can reply with a a list of counterproposals ranked according to its own preferences. Then, the originator can either accept one of these proposals or cancel the process.

```
originator:  START
```

```
             CP/DBP; id=22; polarity=+; leader=me; timeout=500; q=0.7
             CP/DBP; id=22; polarity=-; leader=you; timeout=1000; q=0.3
helper:      NOT
             CP/DBP; id=22; polarity=-; leader=me; timeout=200; q=0.4
             CP/DBP; id=22; polarity=+; leader=you; timeout=200; q= 0.6
originator:  OK
             CP/DBP; id=22; polarity=+; leader=you; timeout=200
```

In this example the helper refuses the proposals of the originator, who finally accepts the first helper's counterproposal.

It should be noted here that the concrete conversation protocol to be instantiated can be negotiated too. For this purpose, we have introduced the CP type (f.i. the CP/DBP for the downward bidding protocol), analogously to MIME content types (text/html, image/gif, etc.). On the other hand, it is nonsense to negotiate certain attributes for some CPs. For instance, the polarity of the CP to be employed by an auctioneer attempting to open a DBP round is unnegotiable since the auctioneer cannot play the role of a buyer and vice versa.

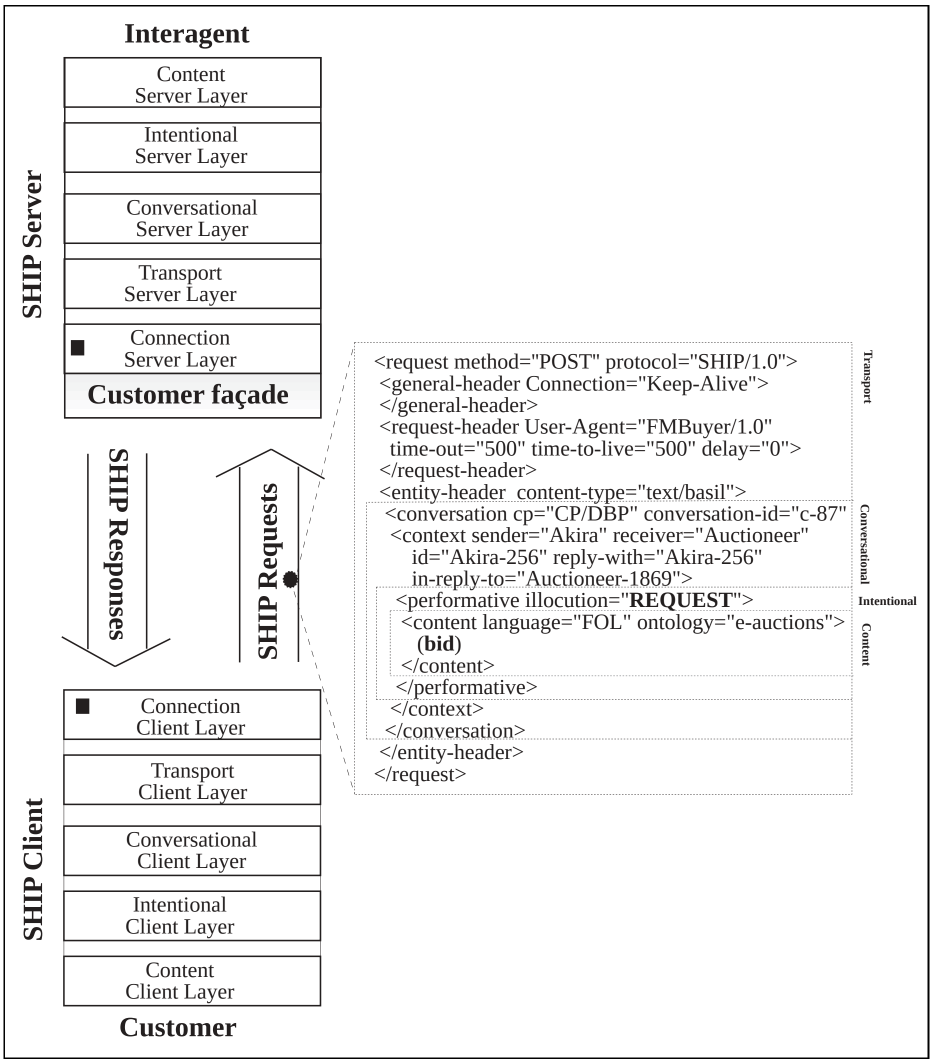
4.2.3 Implementing Interagents

In this section we introduce JIM [Martín et al., 1998], a general-purpose interagent enabled with the capability of managing CPs. We precede the presentation of JIM by the introduction of the SHIP protocol since this has profoundly shaped its architecture.

The extenSible and Hierarchical Interaction Protocol (SHIP) offers a layered approach to support all levels of agent interaction (a layer per level introduced in Section 4.1). Each layer groups a set of protocols with similar functionality together, and provides an abstract interface to higher layers that hides the details related to the particular protocol in use. For instance, two agents can exchange utterances without worrying about the underlying communication mechanism —message passing, tuple-space[3] or remote procedure call. In addition to this, the hierarchical structure of SHIP allows agents to choose their interaction level. This involves the use of the sub-hierarchy of SHIP represented by the chosen layer and the layers below, but not higher layers, e.g. an agent may decide to use the transport layer and the lower levels (the connection layer in this case) but not the higher layers of SHIP. Lastly, SHIP is extensible, in the sense that it permits the incorporation (plug-in) of new protocols into each layer.

SHIP is a request/response protocol which distinguishes two roles: client and server. An interagent may play both roles at the same time, e.g. it can act as a server for its customer, and as a client for another interagent. In a typical message flow a client sends a request to the server, and this sends a response back to the client. All messages exchanged between a client and a server are encapsulated as either requests or responses of the SHIP protocol as depicted in Figure 4.6. Observe that both requests and responses are represented as

[3]A tuple space is a name space shared between distributed clients.



```
<request method="POST" protocol="SHIP/1.0">
 <general-header Connection="Keep-Alive">
 </general-header>
 <request-header User-Agent="FMBuyer/1.0"
   time-out="500" time-to-live="500" delay="0">
 </request-header>
 <entity-header  content-type="text/basil">
  <conversation cp="CP/DBP" conversation-id="c-87"
   <context sender="Akira" receiver="Auctioneer"
      id="Akira-256" reply-with="Akira-256"
      in-reply-to="Auctioneer-1869">
    <performative illocution="REQUEST">
     <content language="FOL" ontology="e-auctions">
       (bid)
     </content>
    </performative>
   </context>
  </conversation>
 </entity-header>
</request>
```

Figure 4.6: All messages between an interagent and its customer are encapsulated as either requests or responses of the SHIP protocol.

XML [Connolly, 1997] messages that nest the information corresponding to each level of interaction. On the sender side, messages to be sent are wrapped at each layer and passed down till arriving to the communication layer responsible for the sending. Upon reception, on the receiver side, messages are unwrapped at each layer and passed up to higher layers till the message arrives at the layer chosen by the agent for interacting.

SHIP has been built upon the following hierarchy of layers (from top to bottom): content, agent communication, conversation,transport and connection. In what follows we describe the functionality of each one of these layers taken as example the SHIP request in Figure 4.6 corresponding to a request for bidding of a buyer agent.

The *content layer* is concerned with the language for representing the information exchanged between agents. Such information refers to terms of a specific vocabulary of an application domain (ontology [Gruber, 1993a]). For example, a piece of information at this level might be the predicate (*bid*) encoded in a first-order language using the ontology *e-auctions*.

The *intentional layer* codifies and de-codifies performatives in a given ACL. For example, a performative requesting for bidding at this level wraps the predicate (*bid*) received from the content layer with the illocution *REQUEST* of a basic ACL whose performatives are shown in Table 4.1.

The *conversational layer* is in charge of providing the necessary mechanisms for the negotiation, instantiation, and management of CPs. Following the example above, the performative is wrapped with the information related to the context in which it is uttered (conversation protocol in use, sender, receiver, etc.).

The *transport layer* is concerned with the transport of utterances. For this purpose, we have devised the Inter-Agent Protocol (IAP), an extensible application-level protocol based on the Hypertext Transfer Protocol (HTTP). IAP constrains the body of a message to be an utterance. Moreover IAP overrides the set of request methods provided by HTTP. For instance, on the one hand the POST method is used to indicate that the utterance enclosed in the message body must be addressed to the receiver specified in the context. On the other hand, the GET method is used to retrieve the utterance accepted by an interagent that matches the pattern of utterance enclosed in the message body. Furthermore, a new addressing scheme has been introduced (iap://) and a new default port (2112).

IAP allows messages to be tagged with different headers to specify their lifetime:

- the delay header indicates how long an utterance queued in an interagent is postponed before being processed. In this way, an agent can tell its interagent to deliver a given utterance after a certain period of time;
- the time-out header indicates the maximum period of time that an agent commits to wait for receiving the response to a given utterance;
- the time-to-live header indicates the lifetime of the utterance once admitted

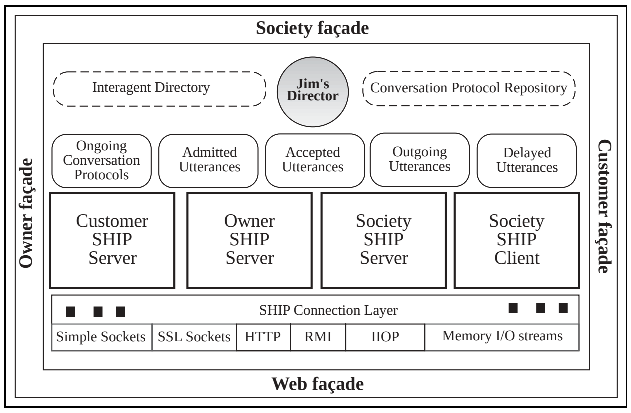


Figure 4.7: An Overview of JIM's Architecture.

by an interagent —thus, when this time expires, the utterance is thrown away by the receiving interagent and the sender receives (back) an error message.

We have introduced IAP due to the lack of an adequate transport protocol for agents' communicative acts. Currently, most ACLs such as FIPA ACL [FIPA, 1998] or KQML [Mayfield et al., 1996] only define minimal message transport mechanisms —in FIPA ACL terms "a textual form delivered over a simple byte stream" [FIPA, 1998]. As an alternative to simple TCP/IP sockets, other transport protocols such as HTTP, SMTP or even GSM have been proposed. However, on the one hand these protocols offer many unnecessary features for this task, and on the other hand they lack many desirable features. Hence the convenience of designing IAP as a specialized protocol for the transport of utterances.

The *connection layer* occupies the lowest level of the SHIP protocol. It is responsible for the creation of connections for SHIP requests. We consider a connection as a virtual circuit established between two agents for the purpose of communication. Different network protocols can be employed to establish such connections. Currently, JIM interagents support a wide variety of network protocols: memory I/O streams, (TCP/IP) stream sockets, and SSL sockets; others like RMI and IIOP will be supported in the near future.

As a final remark, notice that SHIP has required the introduction of several MIME (Multipurpose Internet Mail Extensions) types for declaring the types of conversation protocol (e.g. CP/DBP), of ACL (e.g. text/basil, text/kqml, text/fipaacl, etc.), etc.

Architecture

The architecture of JIM(outlined in Figure 4.7) is best explained in terms of its components:

Interagent Directory This component offers interagent naming services (white pages) that translate from unique identifiers of interagents into their logical addresses and vice versa. Each interagent can communicate with a subset of interagents of the agent society, named its *acquaintances*. A reflexive and symmetric, but not transitive, relation *acquaintance-of* can be established among interagents. An interagent's acquaintanceship is a set of unique identifiers and can vary dynamically, allowing that a collection of agents grow dynamically.

Director This component directs the behavior of the rest of components. It can receive specific directives from both its owner and its customer. Each CP defines a set of constraints that the director takes into account to sort the utterances stored in the different buffers.

CP repository This component stores all CP classes that can be instantiated by an interagent. Such repository can be updated in either *statically* or *dynamically*, e.g. in FM only the market intermediaries working for the institution, the auction house, can define, and store at run-time new CP classes into the CP repository of each interagent. Notice that the conversation protocol repository can either be owned by only one interagent or shared among several interagents.

Ongoing conversation protocols This component stores all *running* CPs (see Section 4.2.2). A CP instance is kept for each conversation in which the interagent's customer is involved. Once a CP is over (reaches the *finished* state) is thrown away (or alternatively kept for tracing purposes).

Buffer for admitted utterances This component is responsible for storing all utterances coming from other interagents in the agent society. An utterance is stored here until either it is accepted or it expires due to a time-out condition.

Buffer for accepted utterances This component stores utterances that have already been accepted by a CP but which have not been required by the customer yet. Once one of these utterances is required by the customer it is packaged up as a SHIP response and forwarded to it.

Buffer for outgoing utterances This component stores utterances that have been forwarded from a customer agent, and received by its interagent but that have not been accepted yet. Notice that the input list of each CP instance can be seen as a composition of the buffers for both incoming and ongoing utterances.

Buffer for delayed utterances This component stores the utterances that have been sent by the customer specifying a lapse of time (using the delay header of the IAP protocol) before they are actually transmitted. When such lapse of time expires, they are automatically transferred to the buffer for outgoing utterances.

Observe that the architecture of JIM presents up to four different façades: *owner, customer, society and web* façades. Each façade represents a different entry for each type of agent requesting the services provided by JIM. The services provided through each façade differ. For instance, only the owner façade allows an agent to upgrade the CP repository. The web façade simply provides access to World-Wide Web resources. For instance, this façade allows a human agent to interact with the JIM's customer through a standard WWW browser. Each façade is built on top of a different instance of the SHIP protocol. In JIM a chain of responsibility is established through the different protocol layers that conform the SHIP instance of each façade. That is to say, each layer is responsible for managing some of the services provided by JIM.

4.3 Agent-mediated Electronic Institutions

In the introductory section of this chapter we argued that an agent-mediated electronic institution can be realised by means of the articulation of institutional agents and interagents. In this section we propose a computational model of *AmEIs* based on this assumption. This computational model must fundamentally guarantee both the proper workings of the coordination mechanisms and the enforcement of the institutional norms contained in any *AmEI* specification.

In what follows we firstly present our conception of institutional agents. Next, in Section 4.3.2 we give the details of a general computational model intended to capture an *AmEI* specification. This explanation will serve also for motivating and introducing the extensions that the basic, general model of interagent presented in Section 4.2 —exclusively devoted to conversation management— need to be incorporated in order to permit an interagent to mediate the interaction of agents within an *AmEI*. Finally, in Section 4.3.3 the extended model of interagents for *AmEIs* is presented.

4.3.1 Institutional Agents

Functionality

Institutional roles are created for some particular purpose, in general for providing some service(s). An institution delegates part of its tasks to agents adopting institutional roles (f.i. recall from Chapter 1 that in the fish market the auctioneer is responsible for auctioning goods, the sellers' admitter for registering goods, and the accountant for the accounts' book-keeping). We refer to this type of agents as institutional agents. Although each institutional (human) agent in the fish market played a single institutional role, this is too hard a restriction that must not be generalised when considering *AmEIs*. Henceforth we assume that an institutional agent can possibly adopt multiple institutional roles.

We define institutional roles' functionalities by means of the responsibilities that they are assigned. Hence specifying a role's functionality is equivalent to making explicit its responsibilities. Furthermore, we could also think of specifying how an agent playing a given institutional role must handle such responsibilities. Summarising, in order to fully specify an institutional role we must specify its *life-cycle* within an institution in terms of its responsibilities along with the policy of responsibilities' management.

More concretely, we specify an institutional role's life-cycle as a regular expression built by combining the operators listed in Table 4.3. Each resulting expression is to contain the execution trajectories associated to the institutional role over the scenes composing a performative structure specification corresponding to the activities in which the role is involved.

$x.y$	x followed by y	$x\|y$	x or y occurs
x^*	x occurs 0 or more times	x^+	x occurs 1 or more times
$x\|\|y$	x and y interleaved	$[x]$	x is optional

Table 4.3: Operators for regular expressions (x and y stand for scene names)

The general form of an institutional role specification is:

$$roleName = expression$$

where *roleName* is the name of the role and *expression* is a regular expression defining its life-cycle whose atomic components are scene names.

Notice that our proposal for the specification of institutional roles resembles the conception of roles presented in [Wooldridge et al., 1999] and the way institutional agents are specified in [Padget and Bradford, 1998]. However, we do not consider the notions of safety and infinite repetition that they borrow from reactive systems [Pnueli, 1986].

For illustrative purposes, let us turn back to the specification of the fish market performative structure depicted in Figure 3.12 (Page 108). For example consider the institutional role *buyers' admitter*. An institutional agent playing this role must firstly enter at the *registry* scene with the *boss* of the market. Next, it is expected to meet other institutional agents (auctioneer, buyers' accountant, sellers' admitter and sellers' accountant) at the *opening* scene. Afterwards it can start processing buyers' requests for admission.

More precisely, the buyers' admitter role could be specified as follows:

$admission$	$= (buyer\_admission.output) \|\| admission$
ba	$= registry.((opening.admission) \|\| closing).output$

Table 4.4: Example of role specification via regular expressions.

where *ba* stands for an abbreviation of the role name (buyers' admitter), *output* stands for the output scene and *admission* is only an auxiliary regular expression.

After registering with the boss, the buyers' admitter makes in parallell for the opening scene and the closing scene. However typically it will not readily enter the closing scene because this must be firstly started by the boss when it decides to close the market (surely not right after opening it), and so it will wait, along with the rest of institutional agents, till the boss signals the creation of the closing scene. While waiting for the boss, the buyers' admitter progresses towards the opening scene, and after that it is ready to admit buyers. Notice that the specification of *admission* permits a buyers' admitter agent to process multiple requests for admission at the same time. It is enabled to process requests but also to keep on listening for new requests for admission from buyers.

From this example follows that an agent playing an institutional role might be required to comply with several responsibilities at the same time. In such a case, an institutional agent must know how to prioritise (schedule) simultaneous responsibilities. For this purpose, responsibilities are ranked according to their relevance. As an example, Table 4.5 lists the priorities assigned to the responsibilities of a buyers' admitter:

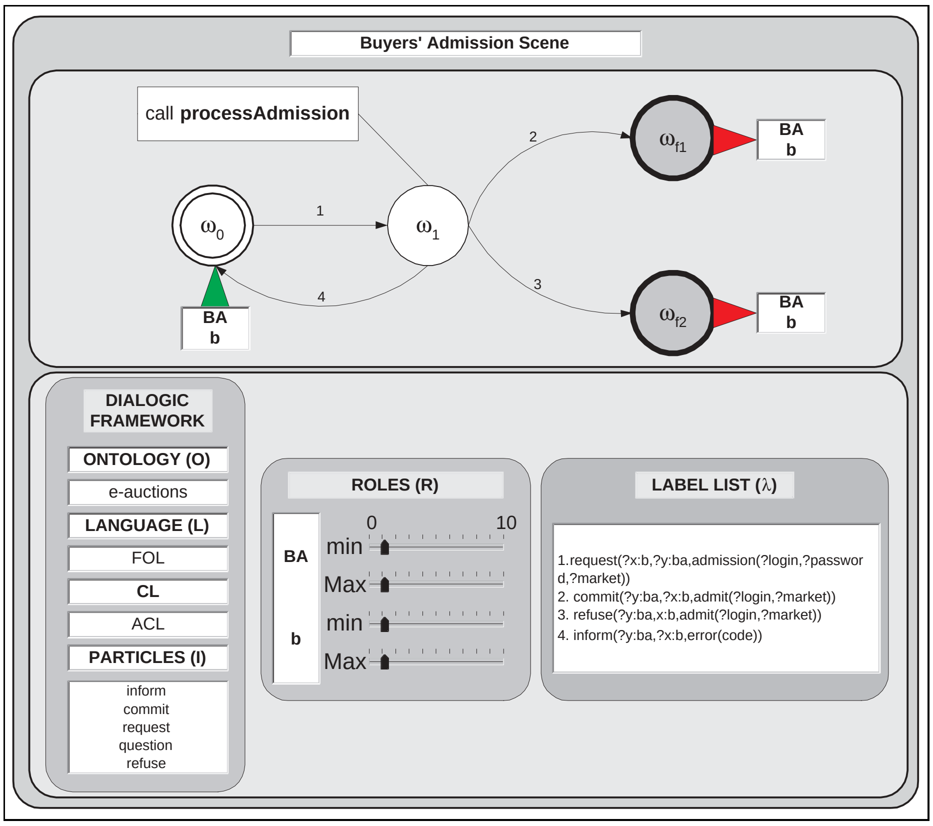
Responsibility	**Priority**
registry	high
opening	medium
buyer_admission	low
closing	high

Table 4.5: Example of priority assignment to an agent's responsibilities.

where *high*, *medium* and *low* denote different priority degrees. Thus, *registry* and *closing* are the highest prioritized tasks, whereas *buyer_admission* is the lowest prioritized task, and *opening* lies in the middle.

So far an institutional role specification provides an institutional agent adopting the role with a description of its responsibilities and even with a prescription of how to handle them. In this way, an institutional agent knows precisely in which scenes it is expected to take part and when. Therefore, the institutional agent knows how to move around within the institution.

And yet there remains the matter of deciding how to behave within each scene in which the agent will get involved. When participating in a scene, at some states an institutional agent will be expected to *act* by uttering an illocution as a result of an inner decision-making process. For instance, an auctioneer must know how to select the winner of a bidding round, a buyers' admitter must decide whether to admit a buyer or not, and a sellers' admitter must know how to tag the incoming goods to be put at auction. As a result, these inner activities should ideally yield illocutions to be uttered by the institutional agent. However, notice that not any illocution is valid, but an illocution that fires a transition whose source state is the scene's current state.



1.request(?x:b,?y:ba,admission(?login,?password,?market))
2. commit(?y:ba,?x:b,admit(?login,?market))
3. refuse(?y:ba,x:b,admit(?login,?market))
4. inform(?y:ba,?x:b,error(code))

Figure 4.8: Buyers' Admission Scene.

Intuitively notice that an institutional agent must only know which method[4] to fire at those scene states at which it is expected to act. Continuing with the example of the buyers' admitter, consider the example of a *buyers' admission* scene depicted in Figure 4.8. Observe that there is a single state, ω_1, requiring the intervention of the buyers' admitter. This is expected either to accept the buyer into the market session, or turning down his request, or perhaps give him another chance after informing him about an error in his request. Whatever the result, the buyers' admitter must decide what to say while at ω_1. At this aim, we suggest also to specify which method to fire at ω_1 (*processAdmission* in the example).

With all the ingredients introduced so far, next we show how to articulate an architecture for an institutional agent.

[4]Here the term method is used as defined in object-oriented programming. In object-oriented programming, a procedure that is executed when an object receives a message. A method is really the same as a procedure, function, or routine in procedural programming languages. The only difference is that in object-oriented programming, a method is always associated with a class.

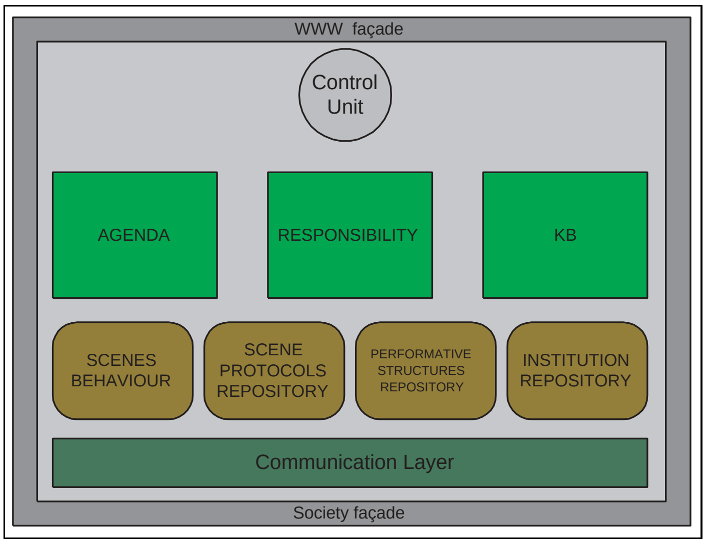


Figure 4.9: An Institutional Agent's Architecture.

Architecture

Figure 4.9 depicts the typical architecture of an institutional agent whose components are detailed below:

Control Unit. This component controls the behaviour of the rest of components.

Behaviour, Scene Protocol, Performative Structure and Institution Repositories. Local repositories that contain respectively specifications of behaviours (methods), scenes, performative structures and electronic institutions available to the institutional agent.

Responsibilities' Agenda. This component contains the specification of the responsibilities corresponding to each role to be played by the agent. In fact, it contains triples of the form $< role, reponsibilities, priority >$ where *role* is the name of the role to be played, *responsibilities* is a regular expression specifying the responsibilities attached to the role, and *priority* is a priority assignment to each responsibility composing the *responsibilities' expression*. Table 4.4 and Table 4.5 show examples of a role specification and a priority assignment respectively.

Agenda. This component contains the specification of the methods to be fired within each scene execution. It is the responsibility of the *Director* to fire the method corresponding to a given scene execution state. For instance, this component must keep annotated that the **processAdmission** method must be called by the buyers' admitter when reaching the ω_1 state.

It also keeps track of all the ongoing scenes in which the agent is involved as well as of the ongoing methods fired by the institutional agent within each scene execution.

Knowledge Base. While a scene is running, the methods fired by the institutional agent may annotate facts onto the knowledge base. Within the knowledge base, facts are grouped depending on the role and the instance of the role that stored them, and so we can consider that it is divided by the different roles adopted by the agent.

Communication Layer. Component in charge of shipping the performatives generated during the activity of the agent to its interagent.

Notice that the proposed architecture for institutional agents is clearly concurrent. This is motivated by the fact that mostly institutional agents will be involved in several scenes at the same time. Going back to our graphical specification of the fish market in Page 108, consider for instance the buyers' accountant. In general, he simultaneously services multiple transactions from buyers and requests from the auctioneer concerning the auction process. Similar situations apply to the rest of institutional agents.

Observe also that the architecture of the institutional agent presents, differently to interagents, only two façades: *society* and *WWW*. But likewise interagents, each façade represents a different entry. Thus, while the web façade allows for a human agent to interact with the institutional agent through a standard WWW browser, the society façade connects the agent to its interagent, and hence to the rest of the institution. However, it is not compulsory for an institutional agent to offer a graphical user interface for a human agent to interact. The human - institutional agent interaction may only occasionally occur. For instance, in the agent-mediated electronic auction house to be described in Chapter 5 most institutional agents offer a graphical interface that simply displays the agents' activities for them to be locally monitored by humans. But differently the *auctioneer* and the *boss* do permit human agents to act on them. Thus, for example, a human agent is allowed to stop a bidding round, or even order the closing of the market. Evidently, institutional agents allowing to have their behaviour altered by human agents lose part of (or perhaps all) their autonomy when doing so, and in that case they must be regarded as semi-autonomous agents.

4.3.2 Computational Model

Henceforth we assume that interagents constitute the sole and exclusive means through which agents interact with the rest of agents within the institution.

Interagents are all owned by the institution but used by both institutional and external agents.

Regarding the flow of agents from scene to scene within an institution, agents must request the institution for the type of scenes of the performative structure that they intend to start and for the active scenes that they aim at joining in.

Under these basic assumptions, picture the following situation: an *AmEI* is set up and then it becomes populated by a large amount of both institutional and exogeneous agents. Then two major questions arise:

- how can agents find other agents aiming at starting new scenes?; and
- once there are active scenes, how can agents find the active scenes that they are interested in participating?

In other words, one of the basic problems faced by an *AmEI* is the well-known connection problem [Davis and Smith, 1983]. Its solution will permit agents to jointly start and access scenes in order to interact.

We propose to devise a special type of institutional agent, the so-called *institution manager* (henceforth *i-manager* for shorter) endowed with the necessary capabilities for solving the connection problem. Below we list the tasks for which the i-manager will be responsible:

- **Scene generation authorisation.** The i-manager shall authorise a group of agents to start a new scene when they satisfy the conditions expressed by the scene's specification. For instance, no more scenes of a given type can be started if the maximum number of active scenes of that type has been reached.
- **Scene book-keeping.** It keeps track of the active scenes along with their participants.
- **Scene Yellow pages.** It employs the book-keeping information above to facilitate information to agents about the active scenes —ongoing activities— within the institution.
- **Request for scene (RFS) validation.** It validates the requests received by agents to either start or join in scenes. If valid, the requests are annotated for future dispatching.
- **Agent name service (ANS).** The i-manager keeps track of all the agents taking part in the institution, along with their roles and the logical addresses of their interagents.

Let us briefly overview how an i-manager behaves. First, recall that an active scene was introduced in Chapter 3 as a multi-agent conversation (jointly) started by some agent(s) whose set of participants may vary dynamically: some participating agents may leave and others join in. In order for the i-manager to keep track of active scenes, it must know when a new scene starts, when some agents leave an active scene, when some agents enter an active scene, and when an active scene is over.

But prior to having any active scene, agents must have been granted access to the institution. Any agent intending to enter the institution must firstly contact the i-manager and apply for the roles to be played. The i-manager will authorise the requested roles to a given agent according to the policy of static separation of duties (SSD) and the cardinality of each requested role, both established when specifying the electronic institution. Once admitted an agent, the i-manager registers the identity of the agent, along with its authorised roles and its interagent's logical address.

Typically the i-manager receives requests from agents to either start or enter active scenes. Upon reception, the i-manager must validate these requests for scenes (RFSs) before accepting them and annotating them as pending requests. This validation process is basically based on the existence of the requested target scenes as active scenes. As a result, agents having issued invalid RFSs have them refused by the i-manager. Otherwise, the i-manager keeps them to check whether they can be indeed serviced. While being on wait, an agent has the right to withdraw its request.

Considering the admitted RFSs, the i-manager checks them in order to detect whether a given group of agents can be authorised to start a scene or some agents waiting to enter active scenes can be authorised so. Say that a group of agents are authorised to jointly start a scene. In such a case, the i-manager accepts the request of the group by sending them a unique scene identifier and the list of participants. Furthermore, and very importantly, the i-manager chooses an interagent to play the role of *scene leader* and broadcasts to all interagents the parameters of the newly created scene along with the identity of the scene leader. Notice that there is no *handshake* between interagents as it happened when negotiating conversation protocols. Here the parameters of the scene protocol to employ are fixed by the i-manager.

We do not claim that our proposal for starting scenes is the most appropriate, but we regard it as a natural extension of our proposal for maintaining the soundness of conversation protocols presented in Section 4.2. Thus, we regard the scene leader as the responsible for guaranteeing the correct operation of a scene execution in a transparent way to its participants.

Scene leaders synchronise with the i-manager for allowing agents to incorporate to active scenes. Thus, a scene's access states establish synchronisation points between the scene leader and the the i-manager at which the first asks the second for agents requesting for joining the scene. If so, agents on wait are allowed to enter.

Exit and final states must also be regarded as synchronisation points between scene leaders and the i-manager. When reaching either an exit or a final state the scene leader conveys to the i-manager the list of agents leaving the scene.

Following these intuitions we shall attempt at illustrating the dynamics of the i-manager by example. First, how does the i-manager handles requests for new scenes and active scenes? It will be keeping the requests for starting or entering scenes issued by agents and the description of the active scenes in two separate blackboards. For instance, Tables 4.6 and 4.7 illustrate respectively the possible states of the blackboards containing the agents' requests and the active scenes in the fish market.

Agent	**Transition**	**Scene Types**	**Target Scenes**	**Target Roles**
pere	and#06	[buyer_admission]	[[ ]]	[buyer_admitter]
martin	and#06	[buyer_admission]	[[ ]]	[buyer]
marc	or#03	[Dutch_auction]	[auction#01]	[buyer]

Table 4.6: Example of an i-manager's Blackboard of Pending Requests

Scene Type	**Active Scenes**	**Participants**	**Leader**
Dutch_auction	auction#01	(peyman,b) (carles,b) (KQLAT,a)	(KQLAT,a)
buyer_admission	buyer_admission#01	(pere,ba) (lluis,b)	(pere,ba)

Table 4.7: Example of an i-manager's Blackboard of Active Scenes

In Table 4.6 agent *pere* and agent *martin* request both for starting a *buyer_admission* scene (denoted as [ ] in the column labelled *Target Scene*) playing respectively the *buyer_admitter* and *buyer* roles.

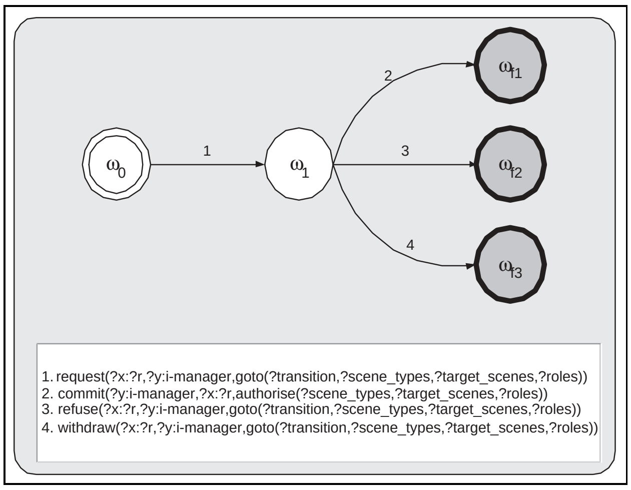
In Table 4.7 there appear two active scenes: one *Dutch* auction scene identified as *auction#01* and one *buyer_admission* scene identified as *buyer_admission#01*. Two buyers (b), *peyman* and *carles*, and an auctioneer (a), *KQLAT*, participate in *auction*#01; while a buyers' admitter (ba), *pere*, and a buyer, *lluis*, participate in *buyer_admission*#01. Looking back at Table 4.6, we see that agent *marc* has requested for joining *auction*#01. Moreover, we distinguish agents *KQLAT* and *pere* as the *leaders* of the current active scenes. Observe that *pere* is taking part in a *buyer_admission* scene and, at the same time, keeps a request for starting a new *buyer_admission* scene. This is consistent with the specification of the *buyer_admitter* role provided above when presenting institutional agents.

Consider now that a new agent, *lucas*, which has already registered as a buyer requests for entering a new buyers' admission scene[5]. For this purpose, it issues the following illocution to its interagent:

[5]It may sound confusing that a buyer requests for admission. Here we understand that agent *lucas* was granted the access to the institution as a buyer. But in order to be registered for participating in auctions, it must go through a separate process.

$$request(lucas : b, [\,], goto(and\#06, [buyer\_admission], [[\,]], [b]))$$

Notice that this illocution contains no recipient indicating that the message is addressed to the institution[6], and then it is collected and managed by the interagent transparently to the agent. Agent *lucas* is requesting for starting a new *buyer_admission* scene ([[ ]] stands for the list of target scenes) playing the *buyer* role ([b] stands for the list of requested roles). In general, RFSs are pre-validated by interagents, which locally filter them before finally posting them to the i-manager, as we explain later on in Section 4.3.3. If valid the received RFS, the interagent autonomously starts a scene with the i-manager whose specification is depicted in Figure 4.10, where $?x$ and r are an agent variable and a role variable respectively to be instantiated with the values of the agent identifier and role of the agent submitting the RFS.



1. request(?x:?r,?y:i-manager,goto(?transition,?scene_types,?target_scenes,?roles))
2. commit(?y:i-manager,?x:?r,authorise(?scene_types,?target_scenes,?roles))
3. refuse(?x:?r,?y:i-manager,goto(?transition,?scene_types,?target_scenes,?roles))
4. withdraw(?x:?r,?y:i-manager,goto(?transition,?scene_types,?target_scenes,?roles))

Figure 4.10: Scene for the management of RFSs (Request for Scenes)

Upon reception of the request, the i-manager writes it down on the blackboard of pending RFSs as depicted in Table 4.8.

At this point, consider for instance that the i-manager authorises agent *pere* and agent *martin* to start a new execution of a *buyer_admission* scene. Next, and according to the specification of the *buyer_admitter* role, *pere* will also request for starting a new *buyer_admission* scene. And then consider that *pere* and *lucas* are also authorised to start a new *buyer_admission* scene. The resulting situation is illustrated by Tables 4.9 and 4.10.

Once admitted a buyer, he is permitted to participate in the auctioning of goods. But first it must know whether there any ongoing auctions, and if so which one to apply for taking part. For this purpose, it asks the institution

[6]We keep this syntax for historical reasons since it was the syntax employed when developing the *AmEI* presented in the next chapter.

Agent	**Transition**	**Scene Types**	**Target Scenes**	**Target Roles**
pere	and#06	[buyer_admission]	[[ ]]	[buyer_admitter]
martin	and#06	[buyer_admission]	[[ ]]	[buyer]
marc	or#03	[Dutch_auction]	[auction#01]	[buyer]
martin	and#06	[buyer_admission]	[[ ]]	[buyer]

Table 4.8: i-manager's Blackboard of Pending Requests (I)

Agent	**Transition**	**Scene Types**	**Target Scenes**	**Target Roles**
pere	and#06	[buyer_admission]	[[ ]]	[buyer_admitter]
marc	or#03	[Dutch_auction]	[auction#01]	[buyer]

Table 4.9: i-manager's Blackboard of Pending Requests (II)

Scene Type	**Active Scenes**	**Participants**	**Leader**
Dutch_auction	auction#01	(peyman,b) (carles,b) (KQLAT,a)	(KQLAT,a)
buyer_admission	buyer_admission#01	(pere,ba) (lluis,b)	(pere,ba)
buyer_admission	buyer_admission#02	(pere,ba) (martin,b)	(pere,ba)
buyer_admission	buyer_admission#03	(pere,ba) (lucas,b)	(pere,ba)

Table 4.10: i-manager's Blackboard of Active Scenes

about the current running auctions by conveying the following illocution to his interagent:

$$question(?x : b, [\,], active\_scenes(Dutch\_auction))$$

This question is collected and handled by an interagent by automatically starting with the i-manager the scene depicted in Figure 4.11. As a result, the agent is informed about the ongoing Dutch auctions.

Notice that requesting about active scenes is only an example of the type of information that the i-manager can provide to agents.

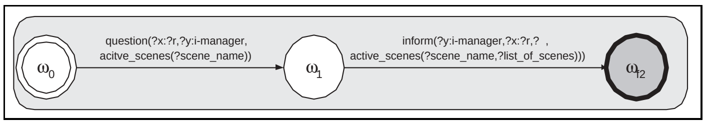

question(?x:?r,?y:i-manager,
acitve_scenes(?scene_name))
inform(?y:i-manager,?x:?r,? ,
active_scenes(?scene_name,?list_of_scenes)))
ω0
ω1
ωf2

Figure 4.11: Querying Active Scenes.

As an example consider that after being informed, *lucas* requests for entering *auction#01*. Its RFS is annotated as a pending request. When do buyers get admitted in *auction#01*? Here it comes the importance of counting on a scene leader. When *auction#01* reaches an access state, its scene leader asks the i-manager for agents that have applied for entering. If any, the scene leader receives a list of new participants which is forwarded to the rest of interagents in the scene so that they can inform their agents. The incorporation of new agents is complete when the i-manager confirms the agents on wait that their RFSs have been accepted, and so finishing the scenes corresponding to their RFSs. For example, Table 4.11 depicts the new situation after *lucas* and *marc* are accepted.

Scene Type	**Active Scenes**	**Participants**	**Leader**
Dutch_auction	auction#01	(peyman,b)	(KQLAT,a)
		(carles,b)	
		(KQLAT,a)	
		(lucas,b)	
		(marc,b)	
buyer_admission	buyer_admission#01	(pere,ba)	(pere,ba)
		(lluis,b)	
buyer_admission	buyer_admission#02	(pere,ba)	(pere,ba)
		(martin,b)	

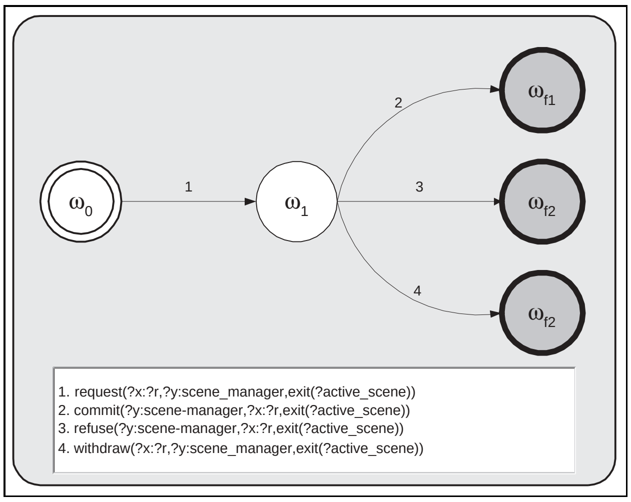
Table 4.11: i-manager's Blackboard of Active Scenes

Some scenes do allow for agents to leave through exit states. For instance, the Dutch auction scene allows for buyers to leave through the ω_1 exit state.

Thus ω_1 will allow participating buyers to leave and new buyers to join in before a new bidding round starts. Say that *marc* wants to leave *auction#01.* At this aim he applies for it by sending the following illocution to its interagent:

$$request(marc : b, [\ ], exit(auction\#01))$$

This time the interagent does not need to contact the i-manager, but directly the scene leader. The type of interaction started between the agent and the scene leader is depicted in Figure 4.12. Several situations may arise. First, if an exit state for buyers is reached, the scene leader lets *marc* leave and updating information is conveniently sent to the i-manager for scene book-keeping and to the rest of interagents in the scene in order to keep their agents informed.



1. request(?x:?r,?y:scene_manager,exit(?active_scene))
2. commit(?y:scene-manager,?x:?r,exit(?active_scene))
3. refuse(?y:scene-manager,?x:?r,exit(?active_scene))
4. withdraw(?x:?r,?y:scene_manager,exit(?active_scene))

Figure 4.12: Request for Exit Scene.

Consider now that in the meantime buyers *lluis* and *martin* have successfully completed their admission processes and have subsequently requested for entering *auction#01.* Then for instace if buyer *lluis* gets tired of waiting he can cancel his RFS by issuing the following illocution to its interagent:

$$request(lluis : b, [\ ], withdraw([Dutch\_auction], [[auction\#01]], [b]))$$

which is then forwarded to the i-manager. Later on, *auction#01* continues till reaching a final state. At that moment, the scene leader contacts the i-manager to inform him about the closing of the scene which implies its removal from the blackboard of active scenes. Consequently, the i-manager must refuse the RFS submitted by buyer *martin* and remove it also from the blackboard of pending requests. And so buyer *martin* receives:

$$refuse([\ ], martin : b, goto(or\#03, [Dutch\_auction], [[auction\#01]], [b]))$$

finishing the scene depicted in Figure 4.10 corresponding to his RFS.

But the closing of an active scene is not the only cause for the i-manager to refuse an RFS. It might be also the case that the i-manager cannot find a proper "binding" for the requesting agent[7]. If this is the case, the RFS is readily refused.

Summarising, notice that in order to move around within the institution agents employ *goto* commands and *exit* commands. An agent's interagent and the i-manager are in charge of validating the agent's RFS according to the performative structure specification and the current set of active scenes within the *AmEI*. Furthermore, they issue questions to the i-manager about the ongoing scenes within the institution in order to decide which activities to apply for either starting or joining in. In all cases, notice that we do not consider the scenes spawned by these commands as part of the specification of an *AmEI*.

Notice also that the management of the agent flow within the electronic institution is realised by means of coordinating the i-manager with the interagents playing the role of scene leaders.

So far we have been referring to accessing or leaving a single scene. Nonetheless, the most intricate situations arise when some agents aim at entering several scenes concurrently. These situations appear when agents traverse transitions of type *and* on the output side (concretely *sync/parallel* and *choice/parallel*) and are in general very demanding because they do require the coordinated activity of all scene leaders of the active scenes involved with the i-manager. In order to illustrate the difficulty inherent to the institutional management of these types of transitions , we take the sample of performative structure depicted in Figure 4.13.

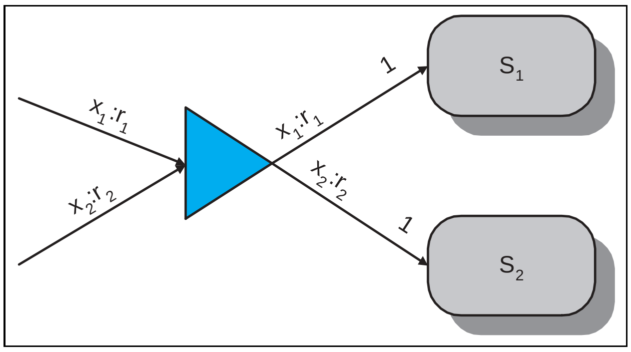


Figure 4.13: A Fragment of a Performative Structure's Specification.

For this example we consider two agents a_1, a_2 playing respectively the roles $r1$ and $r2$ and arriving at the transition shown in Figure 4.13. Let $s_1\#01$, $s_2\#01$ stand for scene executions of s_1 and s_2 respectively. Say that a_1 issues $request(a_1 : r_1, [\,], goto(and\#01, [s_1], [[s_1\#01]], [r_1]))$ and a_2 issues $request(a_2 : r_2, [\,], goto(and\#01, [s_2], [[s2\#01]], [r_2]))$. Provided this setting, when the scene

[7]Here we use the term binding in the sense proposed in Chapter 3.

leader of either s_1#01 or s_2#01 reaches an access state, he asks the i-manager for new agents, and then the i-manager waits (till a time-out expires) for being asked by the other scene leader. If not, he sends a negative answer so that the requesting scene leader can let his scene progress. Otherwise, the i-manager sends to both scene leaders the lists of new agents. Therefore, it might be the case that agents never reach their requested destinations when trying to traverse *sync/parallel* and *choice/parallel* transitions. But that is something that we can only observe dynamically, either via simulation or at run-time, during the insitution's life-cycle.

4.3.3 Interagents for Electronic Institutions

In Section 4.2 we presented a basic model of interagent devoted to the management of conversation protocols. Though enough to be employed for putting through agents in agent-based systems, it is apparent that that basic model is required to have its capabilities extended in order to mediate the interaction of agents within an e-institution. Fundamentally, extending the functionality of interagents must permit them to provide support for addressing the following issues:

- the management of scenes;
- the management of performative structures;
- the management of normative rules; and
- the accountability of an agent's activities.

Therefore, the resulting interagents must not be only capable of enforcing a particular conversation protocol, but also capable of guaranteeing that every agent behaves according to the rules of the institution.

The purpose of the rest of this section is to answer the fundamental question of how to evolve interagents to agents' mediators within electronic institutions. Firstly we will be focusing on the management of scenes and secondly on the rest of the issues above.

Scene Management

Given the definition of scene provided in Chapter 3 we shall distinguish the following scene types:

- **One-to-one.** Scenes involving solely two agents.
- **One-to-many.** Scenes involving multiple agents in which a single agent talks to a group of agents and these in turn can also talk to the agent but cannot talk among themselves.
- **Many-to-many.** Scenes involving multiple agents such that any agent can talk to any agent(s).

Conversation protocols such as introduced in Section 4.2.2 suffice to model one-to-one scenes and even one-to-many scenes. In this second case, the scene can be effectively realised by having the agent talking to a group of agents holding multiple CPs in parallel. In particular, this is the case of the auctioneer in the auction scene wherein he is considered to be holding multiple conversations at the same time with the several buyers taking part in the bidding round. In addition to this, some basic extensions need to be incorporated in order to permit the treatment of access and exit states.

In practice, one-to-one scenes and one-to-many scenes proved to be enough in the practical realisation of the *AmEI* presented in Chapter 5 (the computational counterpart of the fish market), as well as in the development of the *AmEI* described in [Plaza et al., 1999]. Nonetheless the design of other *AmEIs* might eventually need the inclusion of the most general type of scenes, i.e. many-to-many scenes, whose realisation cannot be exclusively achieved by means of CPs. Thus, in what follows we analyse what is required for interagents to be capable of managing scenes.

Fundamentally, the management of many-to-many scenes demands the coordination of interagents so that the maintainance of the global state of the scene is guaranteed. At this point, we must distinguish between the interagents' view and the agents' view of the scene. While all interagents are expected to see and keep a global scene state, their agents have a partial view of the scene in which they are taking part. Thus, depending on their roles, agents only see those states at which either they are recipients of some illocution or they are allowed to send an illocution.

Mainly, we identify two synchronisation problems arising in the management of scenes. Firstly, notice that at some scene states there might be multiple agents intending to utter an illocution that can make the scene state evolve. This is the case of states at which any agent playing a given role is allowed to send some type of illocution. The question is how to handle this type of states. Secondly, we must consider what happens when the scene evolves transparently to some participants, but then it reaches a state that allows them to talk. It must be the responsibility of an interagent to signal its customer, its agent, when its turn arrives.

As to the first issue, it somewhat resembles the case of *mixed* states in CPs. There we proposed to prevent eventual race conditions motivated by mixed states by making the speakers adopt the conflicting roles of *leader* and *follower* and by augmenting CP instances (see Figure 4.5). The leader was said to determine whether to accept or not the follower's messages, while the follower was said to always accept the leader's messages. Considering scenes, we adopt a similar strategy. First, at the outset, when a scene starts, one of the participants' interagents is chosen to play the role of *manager* or *leader* while the rest of interagents play the role of *follower*. One of the tasks of the leader is to accept the illocution that makes the scene state evolve and broadcast it to all interagents in the scene so that they all can maintain the global state[8]. Say that the scene reaches a state at which various agents playing the very same role utter an illocution, but only one of them can make the scene evolve. All the illocutions are sent under the TRY semantics introduced for mixed states, i.e. the interagents receiving illocutions from their agents forward them to the leader and wait for the leader's acceptance or refusal. Upon reception of the illocutions, the leader chooses one of them as the winning illocution and forwards it to all the interagents in the scene. Then the sender of the winning illocution is signaled its acceptance by its interagent while the rest of agents that submitted an illocution have it refused by their interagents. Importantly, the recipient of the winning illocution receives it via its interagent. Notice also that the winning illocution transparently travels from the sender to the addressee via the scene leader, instead of directly going from the sender to the addressee. As a result of the whole process, all the interagents can keep the global state of the scene.

Likewise institutional agents, scene specifications can be also grouped together into a scene repository. Moreover, we can also think of incorporating behaviour into intergents. Instead of being exclusively devoted to guaranteeing the soundness of the interactions taking place within a scene, the institution might need that interagents act at some scene states. We can find a good example in a Dutch auction scene. A buyer receiving the prices called by the auctioneer can only send a request for bidding to its interagent that does not contain the price. Upon reception of the request, the interagent composes the bid to be submitted to the auctioneer with the price of the last offer received by the agent. In this way, buyers cannot cheat by manipulating their bids. Notice that the method to be fired at a given scene state will depend on the role played by the interagent's agent. Observe also the analogy with the specification of an institutional agent's behaviour within a scene (see Figure 4.8).

Consequently, interagents must be also endowed with a scene behaviour repository containing the methods to be fired at the different states of each scene contained in the scene repository.

Requests for leaving scenes and for moving into other scenes, along with questions and requests for withdrawal are handled by the interagent as commands. The starting of the scenes depicted in Figures 4.10, 4.11 and 4.12 with the i-manager are the consequences of these commands.

Institution Management

Recall from Chapter 3 that an agent may possibly adopt various roles at the same time. Moreover, an agent may simultaneously participate with the very same role in different active scenes of an electronic institution. However, an agent can only play a single role within an active scene.

Here arises the question of how to manage an agent's roles. Consider the following example. In a conference centre environment an agent playing the *personal representative agent*(PRA) role can be negotiating several appointments

[8]Alternatively a token-ring-based approach would be also a good way of achieving the same purpose.

at the same time with other PRAs. While in an auction house, an agent playing the *buyer* role can be participating in multiple auctions simultaneously. From this follows that an agent can have several *threads* corresponding to the very same role. For each role, an interagent differentiates between a role's threads involved in some active scenes or *running threads* and a role's threads that are not taking part in any active scene or *idle threads*. For instance, Table 4.12 lists an example of running and idle threads of the buyer role in an auction house. Along the first column we find the scenes in which the buyer took part, while along the second column we find the active scenes in which he is currently involved.

Idle Role Threads	**Running Role Threads**
(Dutch_auction,auction#03)	(First_price_sealed_bid_auction,auction#01)
(English_auction,auction#04)	(Vickrey_auction,auction#02)

Table 4.12: An Example of Running and Idle Role Threads for an agent with the buyer role.

A role thread is switched from running to idle when either the agent leaves the active scene wherein the running thread participates or the scene itself reaches a final state forcing its participants to leave.

In this simple way, an interagent can keep track of all the scenes where the agent either is or has been involved for each adopted role. An interagent manages each role's threads separately, as if each role corresponded to a different agent. Notice that in fact the running and idle role threads represent the current state of the agent within the institution.

At this point we can start considering how an agent can navigate within an electronic institution. For this purpose we require an interagent to contain not only a repository of scene specifications, but also repositories with performative structures' and electronic institutions' specifications likewise institutional agents.

Only idle role threads are considered by an interagent when its agent (its customer) requests for either starting or joining an active scene by issuing a request for scene. When receiving this request, the interagent itself carries out a pre-validation process prior to the starting of the scene shown in Figure 4.10 with the i-manager.

First of all, the interagent checks whether the agent requests for entering an active scene wherein there is a running thread of the same agent. If so, the RFS is readily refused without needing to disturb the i-manager. Otherwise, the interagent continues the checking of the pre-validity of the RFS by testing whether the request is consistent with the performative structure's specification. For this purpose, it suffices for the interagent to certify that the source scene, represented by an idle role thread, and the requested target scenes are indeed connected by means of a transition indicated in the RFS. Furthermore, the request must be consistent with the semantics of the transition. Thus, for instance, for an *choice/choice* transition the agent must only choose one of the output active scenes.

Finally, once guaranteed that the request is structurally valid, the pre-validation process finishes checking that the conditions over the arcs to be followed are satisfied considering the facts referring to the role stored onto the knowledge base.

Next we provide some examples that illustrate the pre-validation process.

First we consider the agent *martin* playing the *buyer* role finishes a *buyer admission* scene (and so has an idle role thread situated in the buyer admission scene) and intends to enter the active *Dutch auction* scene *auction*#01. It sends the following request to its interagent:

$$request(martin : b, [\ ], goto(or\#03, [Dutch\_auction], [[auction\#01]], [b]))$$

The interagent finds that the *choice/choice* transition labelled as *OR03* in Figure 3.12 (Page 108) connects the source scene (*buyer admission*) and the target scene (*Dutch_auction*). The RFS is pre-validated and consequently the scene depicted in Figure 4.10 is started with the i-manager. If not, the interagent itself refuses the RFS.

Consider now the example of an agent playing the buyer role and simultaneously participating in several active auctions of different types. Besides, during their activity the different running threads have succeeded in winning several bidding rounds, i.e. they have made some purchases. If some of these running threads leaves the active auction scene in which he is taking part and requests for going to the output scene (i.e. for leaving the institution), his interagent will refuse the request because the agent has pending obligations: the payment of the acquired goods (according to the *C2* condition in Figure 3.12).

Notice that here we are referring to the pending obligations of agents. Recall that in Chapter 3 we argued that agents' actions within each active scene may introduce future commitments interpreted as obligations in a certain direction that lead to the limitation or enlargement of an agent's acting possibilities, depending on whether an agent fulfills or not its acquired commitments.

Normative rules were introduced as general institutional rules expressing the establishment of obligations and prohibitions within an electronic institution. Evidently these rules are evaluated on an individual basis because they affect the actions made by each agent. This fact suggests the local management of normative rules or, in other words, its interagent-based management. Thus an interagent having the institutional normative rules and the facts resulting from the participation of its customer (its agent) in active scenes is capable of determining which obligations and prohibitions to trigger. Both normative rules and facts are grouped together into the very same knowledge base handled by the interagent.

We address the reader to Section 3.3.5 for some examples of normative rules employed in the fish market.

4.4 Summary

In this chapter we have explored the interrelated issues of designing agent-based systems and designing *AmEIs*.

First, we have started introducing conversation protocols CPs as the methodological way to conceptualise, model, and implement conversations in agent-based systems. CPs allow to impose a set of constraints on the communicative acts uttered by the agents holding a conversation. We have also introduced interagents as autonomous software agents that mediate the interaction between each agent and the agent society wherein this is situated. Interagents ease the development of agent-based systems by taking charge of the complex, time-consuming interaction issues inherent to the construction of this type of systems. In this way, the overhead related to the management of the interaction tasks needed by an agent is shifted to its interagent. Interagents employ CPs for mediating conversations among agents, and so the management of CPs is identified as their main task. Two major benefits, from the point of view of the agent developer, are achieved by employing interagents to articulate the infrastructure of agent-based systems: on the one hand, their agents can reason about communication at higher levels of abstraction, and on the other hand they are released from dealing with interaction issues, and so they can concentrate on the design of the agents' logics. We have also introduced JIM, a general-purpose interagent.

However, the basic model of interagent exclusively devoted to conversation management presented in Section 4.2 does not suffice when facing the development of *AmEIs*. In order to be useful for being employed in this type of systems, we have showed how to extend the functionality of interagents so that the following issues can be successfully addressed:

- the management of scenes;
- the management of performative structures;
- the management of normative rules; and
- the accountability of an agent's activities.

Complementarily, we have also introduced a general model of institutional agent whose roles are specified as responsibility plans. We have also identified a special type of institutional agent, the so-called *i-manager*, as the agent responsible for solving the connection problem between both the institutional and exogenous agents participating in an *AmEI*.

Finally, we have presented an integrated computational model of *AmEIs* founded on the coordinated activity of interagents and institutional agents guided by an electronic institution's specification.

Chapter 5

Realising Agent-mediated Electronic Institutions

It is time to prove the usefulness of the computational model proposed in Chapter 4 by making use of it in the development of a practical agent-mediated electronic institution. In this chapter we present the computational counterpart of the fish market institution presented as a case study in Chapter 1. Thus, we detail the design and implementation of an agent-mediated electronic auction house inspired by the age old institution of the fish market, where heterogeneous (software and human) agents may trade.

In Section 5.1 we motivate the solution adopted to cope with the issue posed by the development of an electronic auction house. Next, in Section 5.2 we argue on the scientific and economic importance of auction-based e-commerce as well as the implications of agent technology in this sparkling area. Sections 5.3, 5.4 and 5.5 show how the computational model presented in Chapter 4 is practically deployed, whereas Section 5.6 describes the resulting architecture. Section 5.7 argues on the capital need of counting on accountability processes that can be exploited by the institution itself to verify and validate its own activity and to found monitoring mechanisms that help gain human agents' trust.

5.1 Motivation

Internet is spawning many new markets. One that is particularly attractive for agent technologies is network-based trading. But if that market is to become an effective actual market various non-trivial issues need to be addressed. Three issues appear to be particularly significant:

- *diversity* of goods, trading conventions, participants, interests;
- *dispersion* of consumers and producers, and also of resources and opportunities; and

- *safety and security* of agent and network-mediated transactions.

Thus it is not surprising that they have been the object of concern and positive attention both by the commercially interested parties as well as the academic community.

We propose to address those issues through a *mimetic* strategy, i.e. by adapting to the new context created by the Information Highway those traditional *institutions* that have proven effective in dealing with those same issues, as upheld in Chapter 3.

Traditional trading institutions such as auction houses –and the fish market in particular– have successfully dealt with the issues of diversity and dispersal. For instance, by defining strict trading conventions where goods of specified kinds (e.g. fish of certain quality) are traded under explicit time/location restrictions (e.g. twice a day at fixed times at the fish market building) under strict negotiation protocols (e.g. downward bidding protocol). Participating agents are subject to terms and conditions –involving identity, credit and payment, guarantees, etc.– whereby the soundness of transactions becomes a responsibility of the institution itself, who in turn enforces those terms and conditions on its own behalf. In practice, the auction house upholds the fairness of the negotiation process and the accountability of transactions by defining and enforcing stable conditions on:

- the eligibility requirements for participating buyers and sellers;
- the availability, presentation and delivery of goods;
- acceptable behaviour of participants within the site;
- the satisfaction of public commitments made by participants.

In this chapter we show how this mimetic strategy can lead to an actual electronic auction house. We present a *proof of concept*-level release, FM96.5 [Rodríguez-Aguilar et al., 1997], of an electronic auction house that is a rather complete implementation of the trading conventions of the fish market. It includes, among other features, a sound and fair implementation of a real-time downward bidding protocol and the capability of allowing for the fair and secure participation of heterogeneous human and software agents[1] thanks to the effective deployment of interagents. Interagent-based mediation guarantees that participants are subject to the explicit –and enforceable– behavioural conventions of the institution.

Originally, FM96.5 only allowed for the auctioning of goods under the rules of the downward bidding protocol, likewise the fish market. At a further stage, the auction house was evolved in order to accommodate other classic auction protocols, namely *English*, *Vickrey*, and *First-price Sealed-bid*. Two major benefits stem from the evolution of FM96.5. On the one hand, an electronic auction house capable of auctioning goods under various auction protocols (the *classic*

[1]By heterogeneous we mean agents of arbitrary complexity, with different architectures, and written in different languages.

ones) was obtained. On the other hand, and more importantly, a more flexible implementation derived ready to incorporate other price-fixing mechanisms if required in a smoothly, non-expensive way.

Lastly, an important aspect to consider is that the implementation presented in this Chapter is founded on the specification depicted in Figure 3.12 (see Chapter 3).

5.2 Auction-based e-Commerce

5.2.1 Why Auctions?

Auctions have gained enormous popularity in the Internet. The proliferation of on-line auctions —such as Auctionline [AuctionLine, www], Onsale [Onsale,], InterAUCTION [InterAUCTION, www], eBay [eBay, www], more recently amazon [Amazon, www] and many others— has established auctioning as a main-stream form of electronic commerce. Typically negotiation on the Internet amounted to the seller offering his products at fixed prices. Instead, auctions appeared offering a more general, dynamic mechanism for price determination, benefitting from the existence of a vast range of auction protocols. Auctions have succeeded as a convenient mechanism for automated trading, due mainly to the simplicity of their conventions for interaction when multi-party negotiations are involved, but also to the fact that on-line auctions may successfully reduce storage, delivery or clearing house costs in many markets. Moreover, the digital world does not require that parties be geographically co-located (only virtually co-located). Interestingly, and according to [Economist, 1999], auction-based commerce is predicted to continue growing:

> Bid-ask auction markets will proliferate, enabling commodity goods in over-supply to be sold efficiently at market-clearing prices instead of either clogging channels or being dumped. Perishable or time-sensitive goods, such as unused space in lorries or unsold media advertising, can be sold economically, allowing intelligent yield management across industries, even in highly fragmented markets. Indeed, Forrester predicts that yield-management pricing will become the norm in automated purchasing.

Furthermore, the success of auctions has led to the rapid creation of sub-industries such as newsletters [AuctionLand, www], auction software providers [Opensite, www], and specialised search engines [Bidfind, www, mysimon, www].

From the AI point of view, this popularity has spurred AI research and development in auction houses [Wurman et al., 1998, Rodríguez-Aguilar et al., 1997, Collins et al., 1998][2] as well as in auction strategies for agents [Preist, 2000, Boutilier et al., 1999, Garcia et al., 1998a]. Moreover, we must not forget

[2]In fact, the work presented in [Collins et al., 1998] is a test-bed for multi-agent negotiation, but we include in this group because it also includes auction mechanisms.

that automated auctions are not only employed in web-based trading, but also as one of the most prevalent coordination mechanisms for market-based resource allocation problems (f.i. energy management [Ygge and Akkermans, 1997, Ygge and Akkermans, 1996], climate control [Huberman and Clearwater, 1995], flow problems [Wellman, 1993]).

5.2.2 On the Role of Auctions in e-Commerce

Generally speaking, software agents can help automate a wide range of tasks including trading over the Internet. Thus, software agents (or multi-agent systems in a more general sense) can function as mediators in electronic commerce over the Internet. To understand the roles that agents or multi-agent systems are called to play as e-commerce mediators, we agree with [Guttman et al., 1998] in the need to explore such roles in the framework of a common model.

In [Guttman et al., 1998] a model is presented inspired on traditional marketing *Consumer Buying Behaviour*(CBB) research useful for analysing these roles in the context of a common model. This model considers the actions and decisions involved in buying and using goods and services. According to the model there are six fundamental stages guiding consumer buying behaviour:

- **Identification.** At this stage the consumer becomes aware of some need, and so he becomes receptive to product information.
- **Product Brokering.** Retrieval of product information based on the consumer criteria leading to the *consideration set*, the set of product alternatives, that helps determine what to buy.
- **Merchant Brokering.** This stage combines the consideration set with merchant-based information so as to determine who to buy from.
- **Negotiation.** This stage is concerned with fixing the terms (price and others) of the transaction that will be highly varying in duration and complexity depending on the type of market (retail market, stock market, etc.). The main benefit of dynamically negotiating a price is that the merchant is relieved from pre-determining the value of the good, and therefore it is the market itself which does the job for the merchant. Ideally, the good is to be allocated to those who value them most.
- **Purchase and Delivery.** This stage either signals the termination of the negotiation stage or occurs some time afterwards.
- **Product Service and Evaluation.** This stage highly depends on the degree of satisfaction achieved by the buyer and the quality of product and costumer service offered by the seller.

Considering the nature of agents, they are most suited for the product brokering, merchant brokering and negotiation stages of the model presented above.

We identify several commercial products devoted to the *product brokering* stage of the CBB model. For example *PersonaLogic* [PersonaLogic,] is a system endowed with a constraint satisfaction engine that helps the consumer filter out products based on his preferences specified as constraints. Or *Firefly* [Firefly, www], which acts as a recommendation system using the opinions of like-minded people. The recommendation mechanism employed by Firefly, the so-called *automated collaborative filtering* (ACF) [Shardanand and Maes, 1995], is based on the comparison of product ratings of shoppers with similar taste.

There are also systems assisting consumers at the *merchant brokering* stage: *BargainFinder* [Bargainfinder,], and its successor Jango [Jango, www] too, compare merchant alternatives on the sole basis of their on-line prices, while Kasbah [Chavez et al., 1997, Chavez and Maes, 1996, Kasbah,] allows to create agents that pro-actively look for potential buyers and sellers to negotiate with them on the consumers' behalf.

As to the negotiation stage, auctions come into play when considering automated negotiation. At a web-based auction house, analogously to traditional auctions, goods are sold using a choice of auction protocols. The fixed rules defining each auction protocol highly constrain the negotiation process, and so reduce its complexity. Within an auction setting, the auction house mediates the interaction and transaction between potential buyers and sellers. Buyers' and sellers' interactions are in most cases quite natural and simple:

- Goods –which may be inscribed directly by external sellers, or otherwise obtained by the auction house– are catalogued and even sometimes displayed –electronically and/or physically– before and during an auction.
- After registering in a given auction –usually a simple e-mail inscription– a buyer can submit his bids either by e-mail, by fax, by submitting a web-form or interacting with a web-based interface or even by post.
- Payments are usually through credit cards, and sales are definite up to actual payment, but most of the time the physical transactions (actual payments) are explicitly relinquished by the auction houses. Some sales are defeasible if protested –and properly supported– within a period of time.
- Most auction servers include several auction formats, and yet in general the most popular auction format is a sealed-bid *English* protocol —the item goes to the highest bidder.
- The evolution of each bidding round is displayed on a browser or sent by e-mail to participating buyers. Special mention deserve the live auctions conducted at [Amazon, www], where traders can even watch and listen live the auction event.
- In most cases, single bidding rounds are open for an extended period of time —though there are some exceptions such as [Amazon, www, Auction-Line, www] that host live bidding rounds— and terminate on a previously

announced closing date, though sometimes the auctioneer determines when a bidding round closes.

In general, buyers must mainly concern about managing their own trading strategies. And so far they do not receive much support in this respect. However, it has become fashionable for some auction houses (f.i. eBay [eBay, www], amazon [Amazon, www], Adauction [Adauction, www]) to offer a bidding agent, the so-called *proxy* bidder, to save the buyers' time by automating their bidding: since some auctions may last for several hours or even days, a buyer can tell his proxy bidder the maximum price that he is prepared to pay for a specific good without letting other bidders know, and let the proxy bidder do.

In order to somewhat support them in such a task, some auction houses allow for the enablement of proxy bids. Nonetheless, this capability is fundamentally based on the definition of a reserve price, disregarding the specification of bidding strategies.

On the other hand, and alternatively to these specialised auction houses, there are already examples of agent-based systems that deal with non-automated negotiation transactions. For instance, Kasbah is a multi-agent system where users create buying and selling agents to help transact products. Unfortunately, negotiation in Kasbah has been largely simplified, because, after matching buyers and sellers, buyers can only offer a bid to sellers without restrictions as to time or price, and sellers can only accept it or refuse it. Negotiation across multiple terms is not conducted by Kasbah's agents. This drawback does not appear in Tête-a-Tête [Guttman and Maes, 1998, Tete-a-Tete,], an agent-based system that profits from the results obtained during the first stages of the CBB model to make the agents cooperatively negotiate across multiple terms of a transaction based on [Parsons et al., 1998].

5.3 Market Blueprint

Following the description and analysis presented in Chapter 1 the actual fish market can be described as a place where several *scenes* take place simultaneously, at different places, but with some causal continuity. Each scene involves various agents who at that moment perform well-defined functions. These agents are subject to the accepted market conventions, but they also have to adapt to whatever has happened and is happening at the auction house at that time. The principal scene is the auction itself, in which buyers bid for boxes of fish that are presented by an auctioneer who calls prices in descending order —the downward bidding protocol. However, before those boxes of fish may be sold, fishermen have to deliver the fish to the fish market (in the *sellers' admission scene*) and buyers need to register for the market (at the *buyers' admission scene*). Likewise, once a box of fish is sold, the buyer should take it away by passing through a *buyers' settlements scene*, while sellers may collect their payments at the *sellers' settlements scene* once their lot has been sold.

One important aspect of the actual fish market —which can be transferred directly to the computational version— is the presence of market intermediaries playing different roles: the auctioneer, the market boss, a receptionist, a credit officer. These intermediaries interact with buyers and sellers on behalf of the institution, and therefore have authority to request, acknowledge, dismiss or accept all the actions that sellers and buyers need to perform within the fish market. Furthermore, all those interactions between the market intermediaries and external agents (buyers and sellers) can in fact be associated with standardised speech acts, some of which are probably tacit in the actual fish market, but explicitable nonetheless in the computational model. In fact, we have tried to mirror all actual fish market illocutions, tacit or explicit, to produce a purely dialogical computational version. FM96.5 has been designed to show the full complexity of those interactions while keeping as strong as possible a similarity with the ontological elements of the actual fish market.

For instance, we have tried to identify computational agents in FM96.5 with either buyers or sellers or actual market intermediaries —we identify agents not with functions of intermediation, but with actual persons. We have also tried to mirror all actual fish market illocutions, tacit or explicit, with agent illocutions that are always explicit. Illocutions can be regarded as the basic unit of analysis in the actual-world fish market. These illocutions are performed by humans with some intention in mind and eventually change the state of the world in a way analogous to the way physical actions do [Searle, 1969].

In this version we have chosen to build agents that correspond to actual fish market intermediaries. A market boss, in FM96.5, acts as auction supervisor and ultimate authority in the auction house. An auctioneer takes care of the bidding process. Two buyer intermediaries are deployed in this version: a buyer admitter who handles identity and acceptability conditions of buyer candidates, and a buyer manager who takes care of the financial dealings and physical location of buyers. There are also two seller intermediaries, a seller admitter who registers sellers and goods, and a seller manager responsible for the sellers' accounts.

As to trading (buyer and seller) agents, they participate in the FM96.5 always and exclusively through interagents which must be regarded as software incarnations of the *electronic mineing devices* to which we referred when introducing the actual fish market in Chapter 1. Interagents receive all the (significant) market illocutions, and transmit to the market only those illocutions that trading agents are allowed to express; always in a standardised form and only in scenes and moments when these illocutions are acceptable. Notice that human traders will be treated differently by interagents, since their interaction is handled through a graphical user interface which collects the users' actions to be subsequently translated by interagents into their corresponding illocutions.

Conceptually we conceive the computational marketplace composed of several *virtual places*, locations, corresponding to the physical places composing the actual fish market. Then we associate each scene to a single virtual location and this in turn may have associated multiple scenes or, in other words, scenes are logically grouped into virtual places. Typically, each virtual place is composed of a vast amount of agents that might be physically running at different sites but virtually situated in the same place, and possibly involved in different scenes within the place. Under these assumptions, one should distinguish an agent flow corresponding to buyers and sellers moving from virtual location to virtual location and from scene to scene (i.e. from activity to activity), and a communication flow caused by illocutions exchanged between agents. We regard virtual locations as particularly useful for assisting human agents' navigation through the market's activities.

Next, in Section 5.4 we present the computational realisations of the different types of agents participating in the marketplace, while in Section 5.5 we describe how these agents interact in the framework of the several activities (scenes) composing the market's activity.

5.4 Market Agents

We shall distinguish three types of agents:

(i) *institutional* agents;

(ii) *interagents*; and

(iii) *trading* agents,

whose features are presented next.

5.4.1 Institutional Agents

Though in Chapter 4 we presented a very general model of institutional agent some simplifications apply to the actual realisation of institutional agents in FM96.5. In fact, the general model presented in the preceding chapter arose as a generalisation of the institutional agents developed for FM96.5.

First, as noted above, we employ an institutional agent for each institutional role identified in the fish market or, in other words, each institutional agent adopts a single institutional role. Therefore there will be an agent for each one of the following roles: *boss*, *buyers' admitter*, *sellers' admitter*, *auctioneer*, *sellers' admitter* and *sellers' manager*. Henceforth we will be referring to each institutional agent by the role that it plays. Note that institutional agents must be regarded as the institution's service providers (trading agents' admission, credit management, auctioning, etc.) from which trading agents profit. Henceforth we will be also referring to institutional agents as *market intermediaries*.

In order to specify the responsibilities assigned to each institutional agent, we make use of regular expressions composed of scene names as introduced in Section 4.3.1 in Chapter 4. The specifications in Table 5.1 have been elaborated according to the final specification of the fish market performative structure shown in Figure 3.12 (see Page 108).

When introducing our general model of institutional agent, we referred to the assignment of priorities as a means of handling the responsibilities ascribed to a role. Thus, in FM96.5 institutional agents have their responsibilities ranked according to their importance. In general, scenes involving the boss are ranked the

$boss$	$= registry.opening.closing$
ba	$= registry.((opening.(buyer\_admission)^*)\|\|closing)$
sa	$= registry.((opening.(seller\_admission.good\_delivery)^*)\|\|closing)$
$auctioneer$	$= registry.((opening.(RFG.(Dutch\|\|credit\_line))^*)\|\|closing)$
bac	$= registry.((opening.((credit\_line.good\_adjudication)^*\|\|$ $buyer\_settlements^*))\|\|closing)$
sac	$= registry.(opening.(good\_adjudication^*\|\|seller\_settlements^*)\|\|closing)$

Table 5.1: Institutional agents' responsibilities specification.

highest, next follows those scenes involving exclusively institutional agents, and lastly those scenes wherein trading agents, buyers and sellers, participate. From this follows that inner coordination activities among the institutional agents themselves are considered more important than servicing trading agents. Tables 5.2 and 5.3 list the assignment of priorities for each institutional agent:

boss	ba	sa
(registry,H) (opening,H) (closing,H)	(registry,H) (opening,H) (closing,H) (buyer_admission,L)	(registry,H) (opening,H) (closing,H) (seller_admission,L) (good_delivery,L)

Table 5.2: Assignment of responsibilities' priorities for the boss, the buyers' admitter and the sellers' admitter.

where each pair *(scene,priority)* stands for a scene name and the priority degree (H = high, M = medium, L = low).

Though highly similar to the general model of institutional agent introduced in Chapter 4, FM96.5 institutional agents differ in the way they communicate both among themselves and with trading agents. Thus, institutional agents in FM96.5 do not employ interagents to communicate with the rest of institutional agents in the electronic institution. Instead their communication layer was turned into a built-in conversation layer. However, we could costlinessly do away with this conversation layer and substitute it by an interagent containing

auctioneer	bac	sac
(registry,H)	(registry,H)	(registry,H)
(opening,H)	(opening,H)	(opening,H)
(closing,H)	(closing,H)	(closing,H)
(RFG,M)	(credit_line,M)	(good_adjudication,M)
(Dutch,L)	(good_adjudication,M)	(seller_settlements,L)
(credit_line,M)	(buyer_settlements,L)	

Table 5.3: Assignment of responsibilities' priorities for the auctioneer, the buyers' accountant and the sellers' accountant.

the protocols needed by each institutional agent.

As to the communication of institutional agents with human agents, only the auctioneer and the boss of the market offer GUIs that allow humans to eventually alter their behaviour. Table 5.4 depicts the simple GUIs corresponding to the boss and the auctioneer respectively. While the boss' GUI monitors the boss operations and allows for the automatic launching of the market and its eventual interruption, the auctioneer's GUI allows a human supervisor to decide whether an ongoing auction must be suspended. Notice that the rest of institutional agents operate in a completely autonomous way.

A distinguishing feature of the institutional agent playing the *boss* role is that it also acts as institution manager. Recalling the general functions of an institution manager (i-manager) identified in Chapter 4, the i-manager in FM96.5 implements some of them (but applying some simplifications as detailed in Section 5.5):

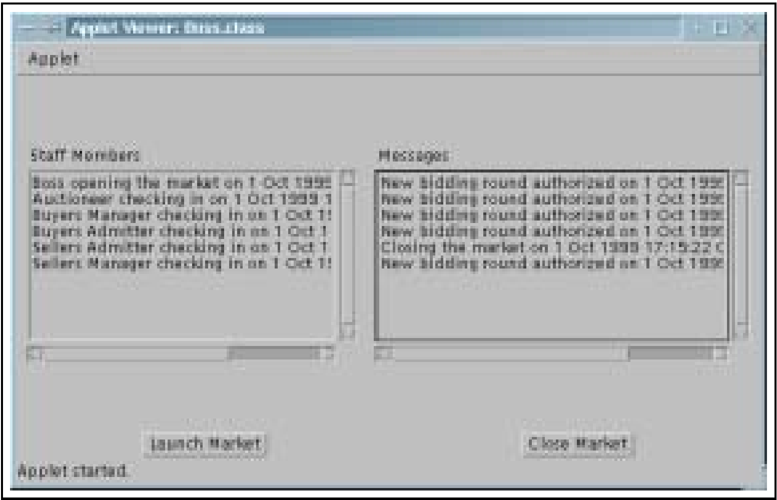
- scene generation authorisation;
- scene book-keeping;
- request for scene (RFS) validation; and
- agent name service (ANS).

Furthermore, the i-manager is also in charge of controlling the static separation of duties of the agents requesting for participating in the electronic marketplace. The policy of separation of duties in FM96.5 is rather simple:

- institutional agents are only permitted to play a single role; and
- trading agents are permitted to play both the buyer and seller role.

5.4.2 Interagents

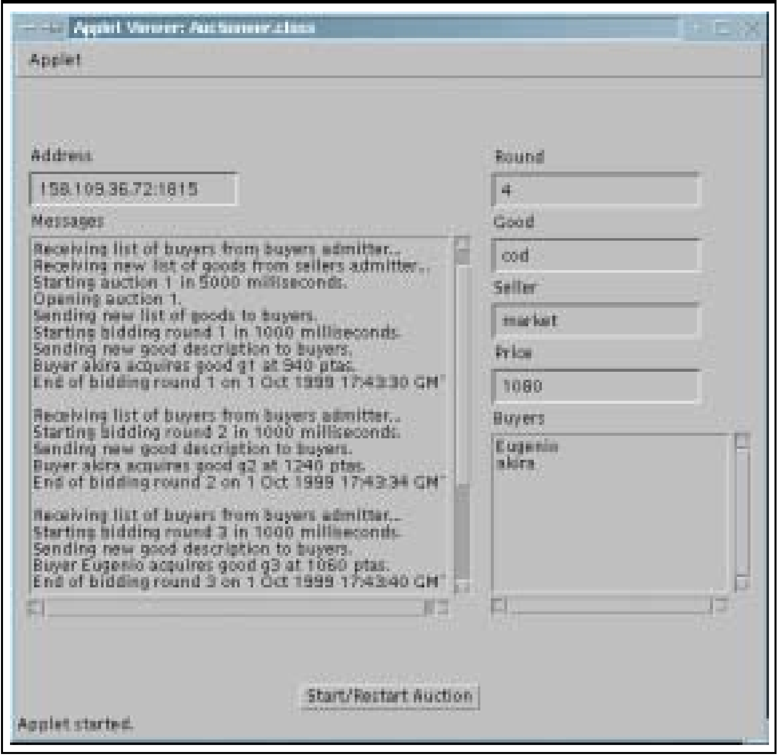
Recalling our conception of *AmEI* presented in Chapter 4, we argued that an *AmEIs* infrastructure is articulated based on institutional agents and interagents, being interagents a particular type of institutional agents responsible for



```
Boss opening the market on 1 Oct 199
Auctioneer checking in on 1 Oct 1999 1
Buyers Manager checking in on 1 Oct 1
Buyers Admitter checking in on 1 Oct 1
Sellers Admitter checking in on 1 Oct 1
Sellers Manager checking in on 1 Oct 1
```

```
New bidding round authorized on 1 Oct 199
New bidding round authorized on 1 Oct 199
New bidding round authorized on 1 Oct 199
New bidding round authorized on 1 Oct 199
Closing the market on 1 Oct 1999 17:15:22 C
New bidding round authorized on 1 Oct 199
```

Applet started.



Address: 158.109.36.72:1815

Round: 4

Good: cod

Seller: market

Price: 1080

Buyers:

```
Eugenio
akira
```

Messages:

```
Receiving list of buyers from buyers admitter...
Receiving new list of goods from sellers admitter...
Starting auction 1 in 5000 milliseconds.
Opening auction 1.
Sending new list of goods to buyers.
Starting bidding round 1 in 1000 milliseconds.
Sending new good description to buyers.
Buyer akira acquires good g1 at 940 ptas.
End of bidding round 1 on 1 Oct 1999 17:43:30 GM

Receiving list of buyers from buyers admitter...
Starting bidding round 2 in 1000 milliseconds.
Sending new good description to buyers.
Buyer akira acquires good g2 at 1240 ptas.
End of bidding round 2 on 1 Oct 1999 17:43:34 GM

Receiving list of buyers from buyers admitter...
Starting bidding round 3 in 1000 milliseconds.
Sending new good description to buyers.
Buyer Eugenio acquires good g3 at 1060 ptas.
End of bidding round 3 on 1 Oct 1999 17:43:40 GM
```

Applet started.

Table 5.4: (a) Boss Graphical User Interface (GUI); (b) Auctioneer GUI.

mediating agent interactions. In the most general model, interagents were said to mediate interactions for both institutional and non-institutional (external) agents.

But the general model of interagents presented in Chapter 4 has its origins in the interagent model implemented in FM96.5. In order to achieve the most realistic implementation of the auction house activity, we decided to standardise as much as possible all conceivable external agent interactions with the market. We took advantage of the highly structured negotiation convention of auctions, and of the fact that in actual fish markets all bidding round interactions can be mediated through an electronic mineing device.

Thus, we mimeticly built interagents as the counterparts of the electronic mineing devices employed in the actual fish market to be used as universal interfaces by buyer and seller agents. Interagents are expected to be installed in the external agent's computer and become the only channel through which messages can pass between external agents and institutional agents. Since interactions are all linked to illocutions in FM96.5, this interface is all that is needed, in principle, to participate effectively in the electronic auction house. But in fact, interagents comply with other necessary duties as well: they sustain the identity of participants, validate illocution emission and reception, and, generally speaking, enforce the auction-house rules –including the bidding protocol. Through institution-owned interagents buyer and seller agents –developed and owned elsewhere or even human buyers or sellers– can participate in electronic auctions.

The versatility of interagents in FM96.5 falls along several directions:

- First, they allow the user to determine the scene where it wants to be active (external agents can only act —or more properly, engage in *dialogue* with institutional agents— at one scene —and so at one place—at a time).
- Secondly, depending on the specific scene, and the prevalent market conditions, it either displays or conveys market information and habilitates the transmission of the pertinent standardized illocutions.
- Finally, since interagents are market-owned, some accounting, liveness and fault-tolerance functions are performed in the background.

Consequently interagents are capable of managing the navigation of trading agents within the performative structure depicted in Figure 3.12, enforcing the institutional rules among and within scenes.

Here we solely concentrate on describing how interagents operate for managing trading agents' inter-scene behaviour, putting off the discussion about intra-scene behaviour for Section 5.5. Therefore, the fundamental issue is the representation of the performative structure and the institutional norms. For this purpose, looking back at the specification we firstly obtain the projection of the performative structure for buyers and sellers, i.e. the sub-performative structure whose scenes involve buyers and sellers. From the result we observe that trading agents, either buyers or sellers, do only participate in a single scene at any time. Therefore we represent the performative structure corresponding to be handled by interagents as a directed acyclic graph whose states are scenes and whose labels are expressions representing arcs' constraints.

Figures 5.1 and 5.2 depict the representations to be handled by a buyer's interagent and a seller's interagent respectively.

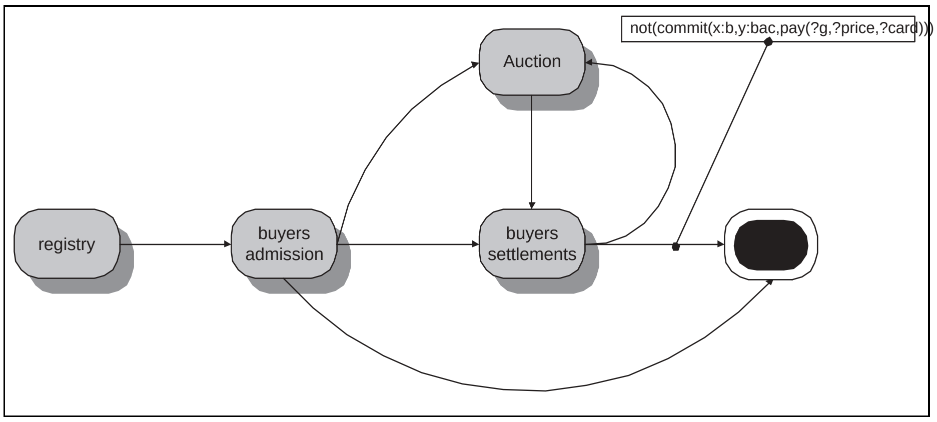


Figure 5.1: Performative structure projection for buyer agents.

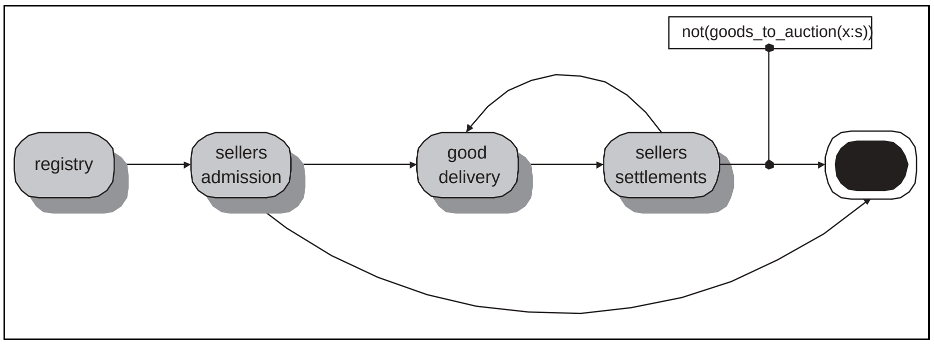


Figure 5.2: Performative structure projection for seller agents.

From Figure 5.1 we observe that a buyer's interagent will allow for his buyer to move to either the auction scene —to take part in the ongoing auction— or the settlements scene —to either update his credit or pay for the acquired goods— after being admitted into the market session. Thereafter, the buyer can move between the *Dutch* auction scene and the *buyers' settlements* scene till deciding to leave the market. A buyer will not be allowed to leave the market if there are pending payments deriving from his purchases in the auction scene (when acquiring a good by auction the buyer makes the commitment to subsequently pay for it in the settlements scene). This is captured by a constraint over the arc connecting *buyers' settlements* and the output scene.

In Figure 5.2 a seller admitted in the market session at the *sellers' admission* scene is permitted to deliver his goods at the *good delivery* scene to be subsequently put in auction. Once completed the delivery, the seller can make it for the *sellers' settlements* scene where he can monitor the sale of his goods. From there a seller can step back to the *good delivery* scene if required to submit more goods. In the end, once a seller's goods have been auctioned —and so have been either sold or withdrawn—, he can collect his earnings and leave the market; otherwise —some of his goods remain to be auctioned— he must wait for the completion of his goods' auctioning. This is captured by the constraint over the arc connecting *sellers' settlements* and the output scene.

Notice that in both cases we solely refer to norms affecting the inter-scene behaviour of trading agents, and so we postpone those affecting their intra-scene behaviour to Section 5.5 where a full account of the activities occurring in the market place is provided.

A trading agent's interagent will keep track of his agent when moving from scene to scene and will guarantee that his agent behaves according to the specification.

5.4.3 Trading Agents

Interagents are the only interface for (human and software) trading agents to the auction house. Note that interagents are coupled with a graphic interface when dealing with human agents, while when interacting with external trading agents they transmit and receive message-shaped illocutions between these external and the internal institutional agents. Figure 5.3 shows the web interface offered to human buyers for remotely participating in the auction house.

Regarding the connection between a trading agent and an interagent, we must choose one of the following models based on their physical locations:

- **Local-local:** Trading agent and interagent running on the trading agent's host.

- **Local-remote:** Trading agent running on his local host machine and interagent running remotely on the auction house's host.

- **Remote-remote:** Trading agent running along with the interagent in the auction house's host.

In FM96.5 we have opted for the local(trading agent)-local(interagent) model, discarding trading agents' mobility. Thus, interagents are shipped over to be hosted in trading agents' host machines, instead of having each trading agent sent over to the auction house. Although we do see mobile agents opening new possibilities for e-commerce —particularly for both data-intensive applications and disconnected computing—, we think that at the same time they raise new security threats. The main problem with mobile agents from the host view is that one can never know what mobile agents are going to do in the host computer. Securing the host from malicious agents is a quite demanding task, because we can only confine where a mobile agent can operate, but we might not recognise if an agent does something ilegal within his constraint operation domain. Furthermore too much restrictions may seriously limit an honest agent's mission.

As to interagents, we prefer to have them sent over to their trading agents' hosts motivated by several reasons:

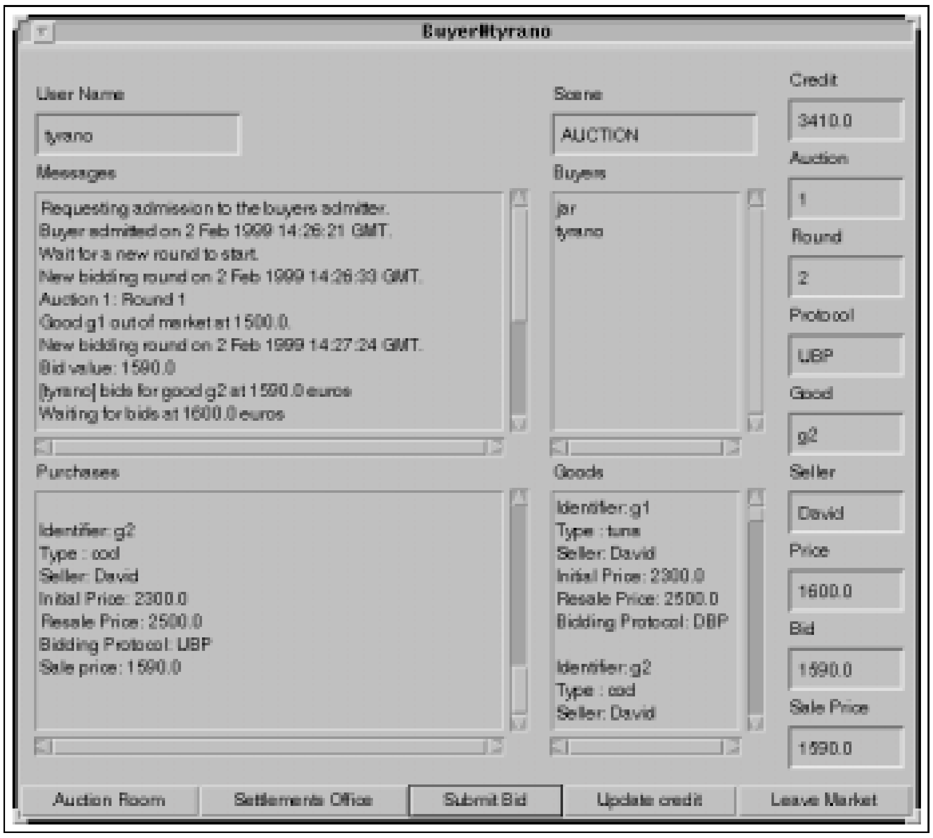
- they can filter out and discard unnecessary messaging that consumes auction house's resources—eventually avoiding denial of service attacks;
- they guarantee that any communication issued from trading agents' host is conveniently encrypted and secured;
- they are small, trustworthy pieces of code (since they belong and are distributed by the institution itself).

Now consider software trading agents. An interagent works as a Java process which uses its standard input and standard output to communicate with trading agents via pipes. In adopting such a simple convention, software agents written in any programming language can interact with the auction house via interagents. Thus, a trading agent must firstly spawn the interagent received from the auction house as a child process and subsequently plug to it. Thereafter trading agent and interagent communicate in a rather straightforward way by exchanging string–based illocutions according to the protocol of the scene wherein the trading agent is situated. Tables 5.5 and 5.6 list respectively the possible contents of the illocutions employed in an interagent-mediated market-buyer exchange, while Table 5.7 lists their intended meanings. Messages 0 to 3 are employed by a buyer to move from scene to scene within the market; messages 3 to 7 stand for the buyers' allowed actions; and messages 8 to 32 correspond to the messages that buyers may receive from the market during a market session.

Interagents allow agents to access all of the features of the auction house present in the Web interface. Thus software agents can interact with the auction house as they would do through the web interface by simply sending to their interagents string-based messages encoding commands.

Notice also that interagents convey to buyers and sellers other information that human buyers and sellers in the fish market would have available *in situ*. For instance, buyers receive the list of participating buyers (which would be seen by a human buyer taking part in the auction), the list of auctionable goods (which are scattered over the floor in the auction room), details of the next good to be auctioned, his own current credit and the list of purchases, etc.

To the best of our knowledge, FM96.5 and AuctionBot [Wurman et al., 1998] are the only auction houses with explicit support for user-written software agents though based on different approaches. AuctionBot defines a message-based API, a communication specification, that agent developers must follow to implement agents that communicate with AuctionBot via TCP. The specification defines



User Name: tyrano

Scene: AUCTION

Credit: 3410.0

Messages

```
Requesting admission to the buyers admitter.
Buyer admitted on 2 Feb 1999 14:26:21 GMT.
Wait for a new round to start.
New bidding round on 2 Feb 1999 14:26:33 GMT.
Auction 1: Round 1
Good g1 out of market at 1500.0.
New bidding round on 2 Feb 1999 14:27:24 GMT.
Bid value: 1590.0
[tyrano] bids for good g2 at 1590.0 euros
Waiting for bids at 1600.0 euros
```

Buyers

```
jar
tyrano
```

Auction: 1

Round: 2

Protocol: UBP

Good: g2

Purchases

```
Identifier: g2
Type : cod
Seller: David
Initial Price: 2300.0
Resale Price: 2500.0
Bidding Protocol: UBP
Sale price: 1590.0
```

Goods

```
Identifier: g1
Type : tuna
Seller: David
Initial Price: 2300.0
Resale Price: 2500.0
Bidding Protocol: DBP

Identifier: g2
Type : cod
Seller: David
```

Seller: David

Price: 1600.0

Bid: 1590.0

Sale Price: 1590.0

Figure 5.3: Human Buyer Applet

message formats for API operation. The API was developed for agent developers to provide a method of interaction with the AuctionBot server through software calls. Notice that trading agents are allowed to directly interact with AuctionBot. Differently, in FM96.5 trading agents are obliged to have their interactions with the auction house mediated by interagents.

However, in both cases the API is complete enough that developers can use it to build their own front end to the auction house. This flexibility is particularly useful for researchers who wish to use the functionality of the auction houses but would rather provide an alternative interface.

In order to ease the implementation of trading agents, FM96.5 provides skeleton programs for buyer agents written in Java, C, and Common Lisp[3]. By providing such a set of templates, we attempted to isolate agent builders from low-level details so that they could concentrate on the design of trading strategies. The Java piece of code below shows the main part of a Java agent template. An agent developer must simply concern about implementing the *to_bid_or_not_to_bid* function corresponding to the agent's bidding strategy.

```
public void run(){
  streams = catchInteragent();
  DataInputStream pannel = (DataInputStream)streams.elementAt(0);
  PrintStream remote = (PrintStream)streams.elementAt(1);
  new ChannelIn(pannel,queueIn,Thread.MAX_PRIORITY).start();
  new ChannelOut(remote,queueOut,Thread.MAX_PRIORITY).start();
  goIntoAuctionRoom();
  while (auctionOn){
    String message = receive();
    if (message != null){
      processMessage(message);
      to_bid_or_not_to_bid(remote);
    }
  }
  goOutOfMarket();
}
```

Notice that so far interagents have been identified to be useful for coping with *agent heterogeneity* —allowing for the participation of agents (possibly) written in different languages with different architectures— and with *societal change* — thanks to their capability of managing performative structures' specifications. Furthermore, interagents importantly contribute to the *robustness* of the auction house by handling trading agent failures that may jeopardise the overall operation of the institution. In general, when an interagent's trading agent crashes, the interagent pro-actively leaves the auction house after informing the buyers' accountant about the failure so that pending payments are postponed. A typical dysfunctional behaviour of trading agents observed in the experiments presented in Chapter 6 comes about during the bidding rounds. Typically some agents

[3]Analogously to the *Standard Template Library* provided by AuctionBot.

#Message	Predicate	Parameters
0	goto	admission $\mid$ auction $\mid$ settlements
1	leave	auction
2	exit	
3	admission	buyerlogin password
4	bid	[price]
5	update_credit	card_code credit
6	statement_state	buyer_login
7	pay	purchase_value

Table 5.5: Messages that (software) buyer agents can utter during a market session

#Message	Predicate	Parameters
8	deny	deny_code
9	accept	admission $\mid$ settlements $\mid$ auction auction_state
10	open_auction	auction_number
11	open_round	round_number
12	good	good_id good_type starting_price resale_price auction_protocol
13	buyers	{buyerlogin}*
14	goods	{good_id good_type starting_price resale_price}*protocol
15	offer	good_id price
16	sold	good_id buyerlogin price
17	sanction	buyerlogin fine
18	expulsion	buyerlogin
19	collision	price
20	withdrawn	good_id price
21	bidders	{#bidders $\mid$ buyerlogin*} + price
22	going	{1, 2}
23	gone	
24	tie_break	buyerlogin
25	start_bidding	bidding_time
26	bidding_time	step
27	end_bidding	
28	statement	[round{sale $\mid$ fine}good_type sellerstarting_price{sale_price $\mid$ fine}]* purchase_value current_credit
29	end_round	round_number
30	end_auction	auction_number
31	closed_market	closing_code

Table 5.6: Messages that (software) buyer agents can receive during a market session.

Predicate	**Semantics**
goto	Goto scene.
leave	Leave scene.
exit	Leave the marketplace.
admission	Request for admission.
bid	Bid at the current or specified price.
update_credit	Update my current credit.
statement_state	Request current state of a trader's statement.
pay	Pay for the acquired goods.
deny	Refuse requested action.
accept	Accept access to scene.
open_auction	The auctioneer opens a new auction.
open_round	The auctioneer opens a new bidding round.
good	Features of the good in auction.
buyers	List of participating buyers.
goods	Lot of goods to be auctioned.
offer	Current offer called by the auctioneer.
sold	The good in auction has been sold.
sanction	Sanction imposed on a given buyer.
expulsion	Buyer expelled out of the market.
collision	Multiple bids at the same price (DBP).
withdrawn	Reserve price reached. Good withdrawn.
bidders	Bidders accepting the called price (UBP).
going	Going.
gone	Gone.
tie_break	Draw among bidders undone.
start_bidding	Authorises to start bidding.
bidding_time	Bidding time counter.
end_bidding	End of bidding time.
statement	Description of the statement of acquired goods.
end_round	Bidding round over.
end_auction	Auction over.
closed_market	End of market session..

Table 5.7: Semantics of the messages exchanged between a buyer and the auction house.

collapse when engaging in lengthy deliberative processes for elaborating their trading strategies and so they may either crash or simply fail to consume the offers called by the auctioneer. In such a case the interagent takes the initiative and leaves on behalf of the trading agent.

5.5 Scenes

Generally speaking, the market activity sequentially progress through three stages: the *opening* of the market, the *auctioning* of goods and the *closing* of the market. Each stage involves multiple scenes. In order to understand and explain how scenes are started, joined and left by agents, i.e. how the institution dynamically evolves, we resort to the computational model introduced in Chapter 4.

First, we dissect the opening of the market, its general dynamics and the closing of the market. Next we treat the auctioning of goods separately due to its higher complexity.

In FM96.5 once the market boss agent starts, the rest of market intermediaries are forced to check in at the *registry* scene by identifying themselves. In fact, no auction can take place until all the market intermediaries check in. The activation of the market is started by the market boss agent who *opens* the market place at the *opening* scene once all market intermediaries have checked in. Thereafter, upon registration at the *registry* scene, traders (buyers and sellers) may start entering the scenes where they are to conduct business, but always subject to the market rules and illocutory constraints.

In FM96.5 the market boss agent plays also the role of institution manager[4]. We must recall that one of the functionalities of the institution manager is to act as the ANS of the institution. Therefore, the market boss will be keeping the list of participating agents along with their roles and logical addresses. Note that the participants' addresses will be exclusively handled by market intermediaries and trading agents' interagents. Thus trading agents know the agents participating in the institution by their logical names.

According to the role specifications of the market intermediaries (institutional agents) provided above, the i-manager will be permanently keeping requests from the market intermediaries to start new scenes whenever the market is not closed. Thus the i-manager will keep the request of the boss for starting a new registry scene, of the buyers' admitter and the sellers' admitter for respectively starting a new admission scene and registering goods, and from the buyers' and sellers' accountants. Furthermore, notice that all these market intermediaries are ready to handle multiple scenes at the same time.

In order to start new scenes trading agents are required. According to our computational model, traders must explicitly request which scene they want to either start or join. However some simplifications apply in FM96.5. Thus, traders do not need to specify either which transition to follow or which active

[4]Henceforth we will be referring to the boss to mean the agent playing both roles: boss and i-manager.

scene is chosen as target since all scenes involving traders must be created, except from the auction scene. But as there is a single, central auction scene in FM96.5, specifying which auction scene to enter is not required either.

For explanatory purposes we will only refer to buyer agents. In general, a software buyer requests for starting or entering new scenes through *goto* messages (see Table 5.5 above), and requests for leaving scenes with *exit* messages. In Figure 5.3 a human buyer can also choose where to go by simply clicking the button corresponding to the target scene. When succeeding, software buyers receive *accept* messages, while human buyers have their change of scene displayed.

Consider the case of a software buyer aiming at starting an admission process. For this purpose, it issues a *goto admission* request to its interagent that validates it, taking into account whether the agent satisfies the preconditions to apply for the scene[5], and subsequently forwards it to the i-manager. Then the i-manager authorises (or perhaps refuses) the buyer's interagent to contact the buyers' admitter to start a new *buyers' admission* scene. The buyer receives the response to its request through its interagent as either an *accept* or a *deny* message. Observe that the process described corresponds to the *Request for Scene* scene introduced in Chapter 4.

When starting a new scene with any external trading agent, market intermediaries are always the leaders of the scene. Notice also that the scenes between market intermediaries (except from the auctioneer) and external trading agents are 2-agent scenes and so can be readily realised by means of conversation protocols.

An important difference with respect to the general computational model presented in Chapter 4 is that the i-manager in FM96.5 does not keep track of active scenes, except from the auction scene. This is motivated by the fact that the auction scene is the only scene whose participants may vary —new buyers may enter and others leave. Furthermore, market intermediaries do not inform either the i-manager about the end of an active scene.

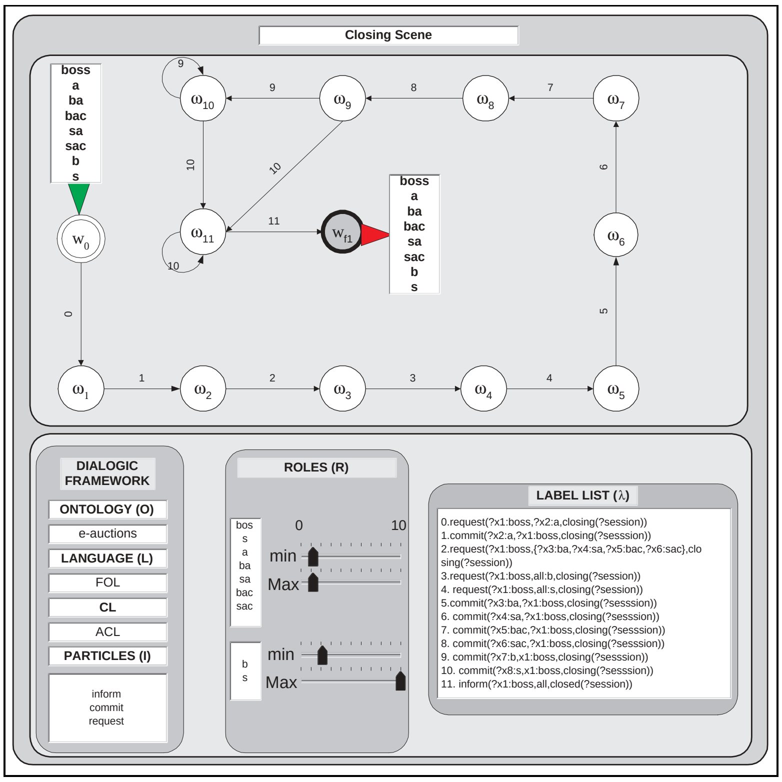
Regarding the auction scene, when reaching an access state the auctioneer, which plays also the role of scene leader, requests the i-manager for buyers having solicited to join the scene —buyer agents by sending a *goto auction* request and human buyers by pressing the *Auction room* button. Then, the auctioneer directly contacts the buyers on wait and accepts them into the auction scene. Thus a buyer agent receives an *accept* message from its interagent, while a human buyer has the change of scene displayed. Importantly, whatever the case buyers also receive information about the current state of the auctioning (goods left and good in auction).

On the other hand, when reaching an exit state the auctioneer considers the requests for leaving the auction scene, if any, submitted by buyers —buyer agents do it by posting a *leave auction* request, while human buyers by requesting either for jumping into the settlements office or for leaving the marketplace. If any buyer leaves the auction "room", the i-manager is informed about the new

[5]For instance a buyer participating in the auction scene cannot apply for starting an admission scene since it has been already admitted.

composition of the scene.

So far we observe that indeed the i-manager in FM96.5 is in charge of most of the functions for an i-manager identified in our general computational model — namely *scene generation authorisation*, *scene book-keeping*, *RFS validation*, and *ANS*. Nonetheless we feel obliged to highlight that they are largely simplified in FM96.5. Although in Chapter 4 we employed the fish market for explanatory purposes to introduce a general computational model of agent-mediated electronic institutions, we shall see that the actual computational realisation represented by FM96.5 does not faithfully comply with the general model, containing some simplifications. These obey to two main factors: the need for easing the implementation and the fact that the general model arises in part from the implementation of FM96.5



0.request(?x1:boss,?x2:a,closing(?session))
1.commit(?x2:a,?x1:boss,closing(?sesssion))
2.request(?x1:boss,{?x3:ba,?x4:sa,?x5:bac,?x6:sac},closing(?session))
3.request(?x1:boss,all:b,closing(?session))
4. request(?x1:boss,all:s,closing(?session))
5.commit(?x3:ba,?x1:boss,closing(?sesssion))
6. commit(?x4:sa,?x1:boss,closing(?sesssion))
7. commit(?x5:bac,?x1:boss,closing(?sesssion))
8. commit(?x6:sac,?x1:boss,closing(?sesssion))
9. commit(?x7:b,x1:boss,closing(?sesssion))
10. commit(?x8:s,x1:boss,closing(?sesssion))
11. inform(?x1:boss,all,closed(?session))

Figure 5.4: Closing Scene

Market closing involves some artificiality in FM96.5. The market boss may stop an auction through a *closing declaration* —whose triggering conditions are explicit, albeit varied— but actual closing requires that all pending activities of market intermediaries be properly terminated. Figure 5.4 depicts the specification of the closing scene, which involves the co-ordination of all market intermediaries as well as the interagents employed by trading agents.

We could briefly summarise the termination of the market in the following steps:

(i) the boss sends a closing signal to the auctioneer (the boss has the right to close the market at any time);

(ii) the auctioneer acknowledges this closing signal as soon as the current bidding round is over;

(iii) the boss multicasts a closing signal to the rest of market intermediaries and to all interagents employed by trading agents to indicate that they have to finish their activities;

(iv) the buyers' admitter and sellers' admitter close the doors of the market to buyers and sellers intending to enter the market session by rejecting their requests;

(v) trading agents in the transient to other scenes have their requests for scenes refused;

(vi) trading agents not participating in any scene are also notified the closing of the market;

(vii) market intermediaries and interagents acknowledge the closing signal once the scenes in which they are involved are over and once they have conveniently annotated the pending obligations of trading agents;

(viii) the boss declares the definitive closing of the market.

Ideally, the institution should not finish the activity till every single agent has finished off his pending obligations. Nonetheless in our current implementation we opted for simply storing each agent's individual commitments in his agenda for future market sessions.

5.5.1 Auction

FM96.5 originally exclusively included the Dutch auction protocol since it is the protocol employed in the real-world fish market. However, later on the auctioneer was redesigned in order to make it flexible enough to ease the task of adding other auction protocols. At present, FM96.5 is also capable of auctioning goods under the rules of the English, Vickrey and First-price Sealed-bid auction protocols.

It is worth recalling that the auctioning of goods requires the coordinated activity of the auctioneer with several market intermediaries.

First, the auctioneer obtains from the sellers' admitter the lot of goods to be sold at the *RFG* (request for goods) scene. Afterwards the auctioneer can start the auction scene in order to sell the received lot. While auctioning goods, the auctioneer will hold an additional scene, the *credit line* scene, with the buyers' accountant with the aim of determining whether a buyer's bids can be supported. In turn, the buyers' accountant will hold the *good adjudication* scene with the sellers' accountant to keep him informed about eventual sales and withdrawals.

Next we exclusively concentrate on the activity taking place in the auction scene. Thus we describe how our original specification of auction scene introduced in Chapter 3 must be enlarged in order to host other bidding protocols.

Downward Bidding Protocol (DBP)

We can identify several situations that may arise during each bidding round in the actual fish market:

- *Proper sale.* When a single buyer submits a bid that his credit can support, it is turned into a sale.
- *Unsupported bid.* When a buyer submits a bid that his credit cannot guarantee. The buyers' manager fines this bidder and the round is restarted by the auctioneer who calculates the new starting price by increasing 25% the price within the bid.
- *Collision.* When two or more buyers simultaneously submit the same bid. The auctioneer declares a collision and restarts the round. Again, the new starting price is calculated by increasing 25% the collision price.

FM96.5 implements faithfully all theses cases, plus two more:

- *Expulsion.* When a buyer is overdrawn and cannot back up a fine, he is sent off the market and the round is restarted as usual
- *Minimum price.* Each good is assigned a minimum price when passing through the sellers' admitter office. If minimum prices are reached, the round is restarted as usual.

Our main concern when implementing the downward bidding protocol has been to ensure fairness while preserving realistic response time[6]. In FM96.5 we achieve it –without supposing common fixed delay intervals– through a clever alternative to common clocks.

In FM 96.5 we regard the termination of a bidding round as the synchronisation point of the round participants. All buyers receive syncopated price sequences. If a buyer is going to submit a bid, he will signal this as soon as the price quotation reaches his target bid. The signal sent back from the interagent to the auctioneer includes the price at which the buyer signalled his intention and the time stamp. As soon as the auctioneer receives a bid, he multicasts to the buyers' interagents the information that a bid is in, which these interagents must acknowledge. Since we can assume a reliable network, the order in which

[6]In the fish market this corresponds to time delays between prices that are short enough to be imperceptible to human buyers but long enough to allow for collisions (i.e. one or two seconds between successive prices).

messages are transmitted is never altered, thus the auctioneer must receive any delayed bids before we receive the corresponding acknowledgements from these bidders. Hence we have the standard two cases:

- *Proper sale.* One bidder.
- *Collision.* Multiple bidders at the same price.

and a new case:

- *Multiple bidders at different prices.* In this case, the highest price bid wins if there is just one, or we restart as usual.

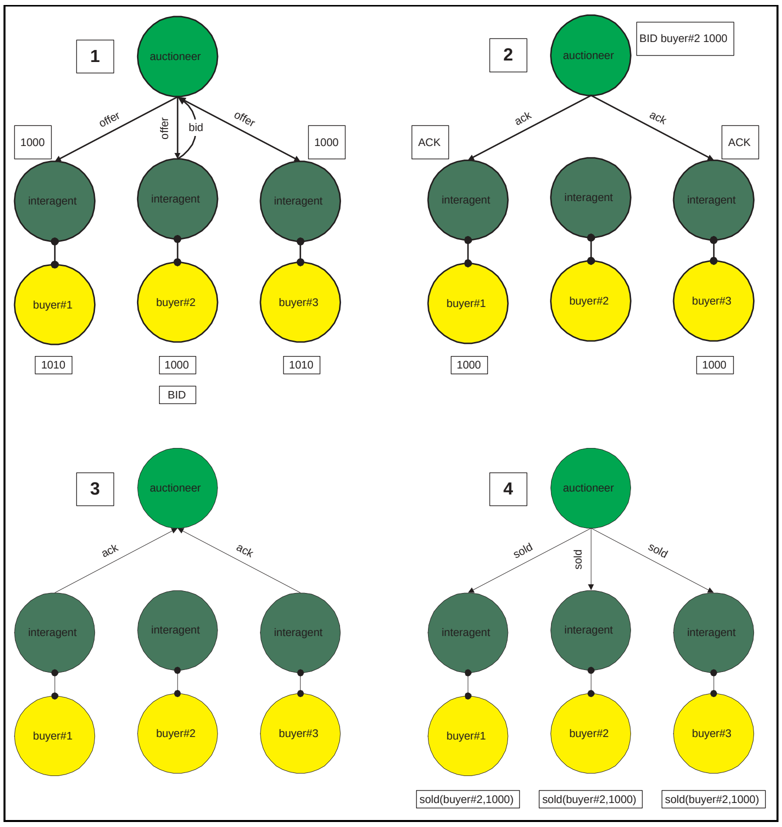


Figure 5.5: DBP Case Study

Figure 5.5 illustrates the synchronisation between the auctioneer and the buyer agents' interagents when a proper sale occurs. We see that *buyer#2* submits a bid when the offer called by its interagents reaches 1000. Next its interagent posts the agent's bid to the auctioneer who requests for acknowledging the received bid to the rest of buyers' interagents. The rest of buyers are given the chance to see the same offers that *buyer#2* received, but if they do not submit any bid their interagents acknowledge *buyer#2*'s bid so that the auctioneer closes the round by publicly declaring the sale.

Recall that in Chapter 3 we already provided a specification of the auction scene considering that all goods were auctioned following a Dutch auction protocol. Next in Chapter 4 we anticipated that the realisation of the scene is achieved by means of conversation protocols held between the auctioneer and the agents participating in the auctioning. Thus each buyer has the view of the auction scene represented by the conversation protocol in Figure 4.3 (see Chapter 4), while the auctioneer holds a protocol compatible to that one for each buyer, i.e. holds multiple conversation protocols in parallel. When there is some bid(s), that means that some conversations have reached the q_9 state (bidding) while others are still at the q_8 state (no bid). The auctioneer stops calling prices and requests for bid acknowledgement to the interagents of the buyers that have not bid yet (their conversations are at the q_8 state). The firing of the auctioneer's method *callACK* starts the synchronisation between the auctioneer and the non-bidding buyers' interagents. In fact, the synchronisation between auctioneer and interagents must be regarded as an interleaved, inner protocol transparent to buyers. And then the process proceeds as described above.

In this version of the downward bidding protocol, synchronism is achieved not within each price quote —as in the actual fish market room— but within the sequence of price quotations that are needed to sell one good[7]. By doing so, and thanks to the fact that a reliable network is assumed, fairness conditions are preserved. Thus, premature bids, foot-dragging, and spoofing are adequately avoided directly by the protocol implementation, while malicious impersonation and snooping are dealt with through interagents.

Upward Bidding Protocol (UBP)

The English auction is perhaps the most popular protocol. It is also known as the open-outcry auction or the ascending-price auction.

We take the definition of the English auction provided by Paul Milgrom [Milgrom and Webber, 1982]:

> Here the auctioneer begins with the lowest acceptable price —the reserve price— and proceeds to solicit successively higher bids from the customers until no one will increase the bid. The item is 'knocked down' (sold) to the highest bidder.

Notice that not all goods at an English auction are actually sold. Eventually, when the reserve price is not reached, the item is declared unsold. Therefore, the auctioneer must state at the conclusion of bidding whether the item has been sold or not.

In America typically the auctioneer calls out the amount he has in hand and the amount he is seeking as well. This is precisely how the auctioneer in FM96.5

[7]Here that (complete) sequence is called a *bidding round*.

acts.[8] Furthermore, an English bidding round in FM96.5 does not start from the reserve price, but from some price above the reserve price. Then the auctioneer starts quoting prices in descending order till some buyer(s) stops the sequence by submitting a bid before the reserve price is reached. When reaching the reserve price, the good is withdrawn, else the auctioneer informs the participating buyers about the bid that he is in hand and starts calling out prices in ascending order.

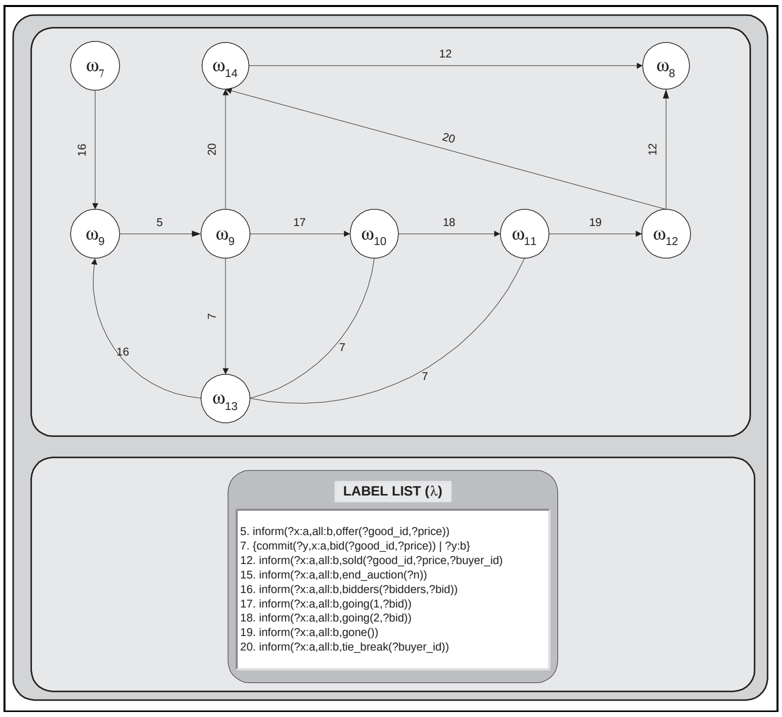
Thus the auctioneer employs the downward bidding protocol to fix the starting price of an English bidding round. Thereafter the auction takes place as explained above. This is why we treat the specification of an English bidding round as an extension of the specification of the auction scene in Figure 3.3 (see Chapter 3). The former specification of auction scene exclusively considered a Dutch protocol as the only auction format. Figure 5.6 depicts the specification of an English bidding round departing from the w_7 state in the auction scene specification. The w_7 is the state at which some buyer(s) stops the descending sequence being quoted by the auctioneer. Henceforth, we shall refer to our computational version of the English auction protocol as *upward-bidding protocol (UBP)*.

The auctioneer proceeds as follows. When having bids in hand, the auctioneer announces it by multicasting to all buyers the value of the current bid along with either the list of buyers or the number of buyers involved[9] (state w_{12}). Next the auctioneer calls another price corresponding to a new offer (state w_{13}). If any bids are submitted at the called offer, the auctioneer announces again the existence of bidders; otherwise, he starts multicasting to all buyers the *going, going, gone* sequence (states w_{14}, w_{15}, w_{16}). If the sequence is not interrupted by any bidder accepting the auctioneer's offer, the good is "knocked down" and sold. Notice that it might be the case that the end of the bidding round is reached with multiple buyers bidding at the same price. In such a case the auctioneer must undo the tie by choosing out a single bidder from the set of bidders.

Similarly to the implementation of the downward-bidding protocol, the implementation of the upward-bidding protocol also employs the synchronisation of the auctioneer with the buyers' interagents. There are three synchronisation points corresponding to the states representing the going, going, gone sequence. Thus, when receiving a *going* message, a buyer's interagent must either post to the auctioneer its buyer's bid or, otherwise, acknowledge it. Before closing the bidding round with a *gone* message, the auctioneer asks all buyers' interagents whether their buyers have any bid to be submitted. If so, the auctioneer restarts the round announcing a higher bid; otherwise the auctioneer sends the *gone* message and the bidding round is closed by announcing the winner.

[8]Differently, in England the auctioneer waits to be told what a bidder's offer.

[9]FM96.5 allows for the configuration of the degree of information revelation required. Thus depending on the chosen configuration the buyers' and sellers' identity might be hidden from the rest of traders.



```
5. inform(?x:a,all:b,offer(?good_id,?price))
7. {commit(?y,x:a,bid(?good_id,?price)) | ?y:b}
12. inform(?x:a,all:b,sold(?good_id,?price,?buyer_id)
15. inform(?x:a,all:b,end_auction(?n))
16. inform(?x:a,all:b,bidders(?bidders,?bid))
17. inform(?x:a,all:b,going(1,?bid))
18. inform(?x:a,all:b,going(2,?bid))
19. inform(?x:a,all:b,gone())
20. inform(?x:a,all:b,tie_break(?buyer_id))
```

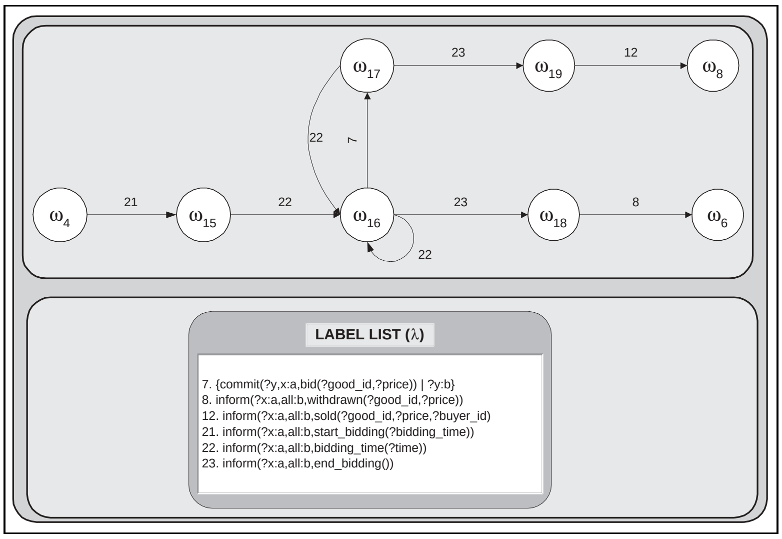
Figure 5.6: Extension of the Auction Scene to handle English bidding rounds.

First-Price Sealed Bid and Vickrey

We group together the first-price sealed bid and Vickrey auction[10] protocols because they are almost identical, only differing in the price payed by the winner. This analogy also appears at the design and implementation levels.

Both auction protocols share the distinguishing feature of being sealed —differently to the open-outcry format of the Dutch and English auction protocols—, and consequently hidden from other bidders, i.e. each bidder is ignorant of other bids. As to the first-price sealed bid, the item is awarded to the highest bidder, who pays exactly the amount that he bid. However, though in a Vickrey auction the item is also awarded to the highest bidder, he must pay an amount equal to the second highest bid (i.e. the highest unsuccessful bid).

Again we specify the first-price sealed bid and Vickrey auction protocols as an extension of the auction scene in Figure 3.3 (see Chapter 3). Figure 5.7 depicts such extensions. Notice that we employ the same specification for both protocols. They only differ in the behaviour of the auctioneer at state w_{18} when determining the sale price to be payed by the winning bidder.



7. {commit(?y,x:a,bid(?good_id,?price)) | ?y:b}
8. inform(?x:a,all:b,withdrawn(?good_id,?price))
12. inform(?x:a,all:b,sold(?good_id,?price,?buyer_id)
21. inform(?x:a,all:b,start_bidding(?bidding_time))
22. inform(?x:a,all:b,bidding_time(?time))
23. inform(?x:a,all:b,end_bidding())

Figure 5.7: Extension of the Auction Scene to handle First-price sealed-bid and Vickrey bidding rounds.

In the current implementation built in FM96.5 the auctioneer starts by informing all buyers that they can start submitting their bids (state w_4). From that moment on, the auctioneer starts the count-down of the bidding time and buyers commence to send their bids. till the bidding time expires. This time we establish a single synchronisation point between the auctioneer and the buyers'

[10]Known also as the uniform second-price auction.

interagents. In this case, once the bidding time is over, the auctioneer asks all buyers' interagents whether their buyers have any unsent bids.

Auctioneer Design

Figure 5.8 depicts the Coad diagram [Dav, 1993] representing the auctioneer agent's behaviour in the auction scene. All rectangular boxes are objects except from the box named *Protocol*, which is a generic interface. Each box is divided into three parts. The upper area contains the name of the object, the middle area its attributes, and the lower area the object's services. The *Auctioneer* object contains general services, while *DBPProtocol*, *UBPProtocol*, *FirstProtocol* and *VickreyProtocol* contain specific service for each bidding protocol required in the auction scene. For instance, the *BestBid* service in *FirstProtocol* and *VickreyProtocol* whose output is in both cases the winning bidder and bid produce different results since the highest bidder in a Vickrey auction pays an amount equal to the second highest bid, instead of the amount that he bids as occurs in a First-price sealed bid auction.

5.6 Market Architecture

Figure 5.9 shows our conceptual view of the architecture of FM96.5 as a multi-layered *AmEI*:

- **Market kernel.** Composed by agents playing the roles of market intermediaries. Each market intermediary is implemented as a Java agent that interacts with the rest of market intermediaries via TCP streams. All the events generated by each market session is recorded onto the market database using the JDBC[11] [JDBC, www] database access API, which also contains a library of agent templates (currently in C, Lisp and Java) for agent developers to construct their own agents. Monitoring agents also post monitoring information to the *auditor*(audit) via *Remote Method Invocation*(RMI) [RMI, www] calls[12] that permit the access to the repositories containing the monitoring information handled by the auditor.

- **Middle-ware layer.** Consisting of the interagents employed by trading agents. They are also completely implemented in Java. Interagents interact with the market intermediaries via TCP and also post monitoring information to the auditor via RMI.

- **External agents layer.** On the one hand, human agents employ a GUI to interact with interagents situated in the middle-ware layer, and hence with the auction house. On the other hand, software agents communicate with interagents via string-based messages exchanged through pipe channels.

[11]Java Database Connection.

[12]The discussion about trust and accountability follow in Section 5.7.

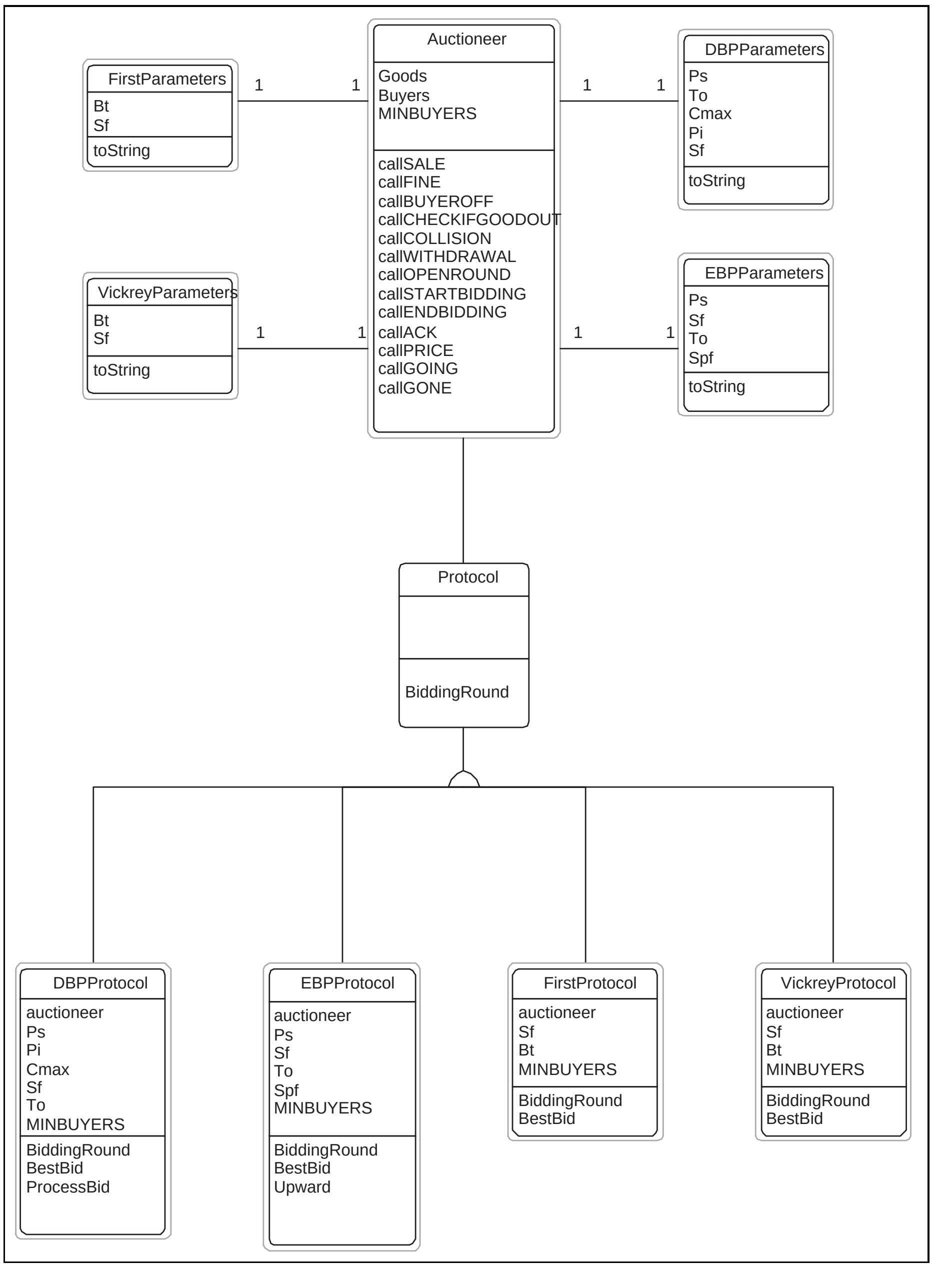


Figure 5.8: Coad diagram representing the design of the auctioneer agent's behaviour in the auction scene.

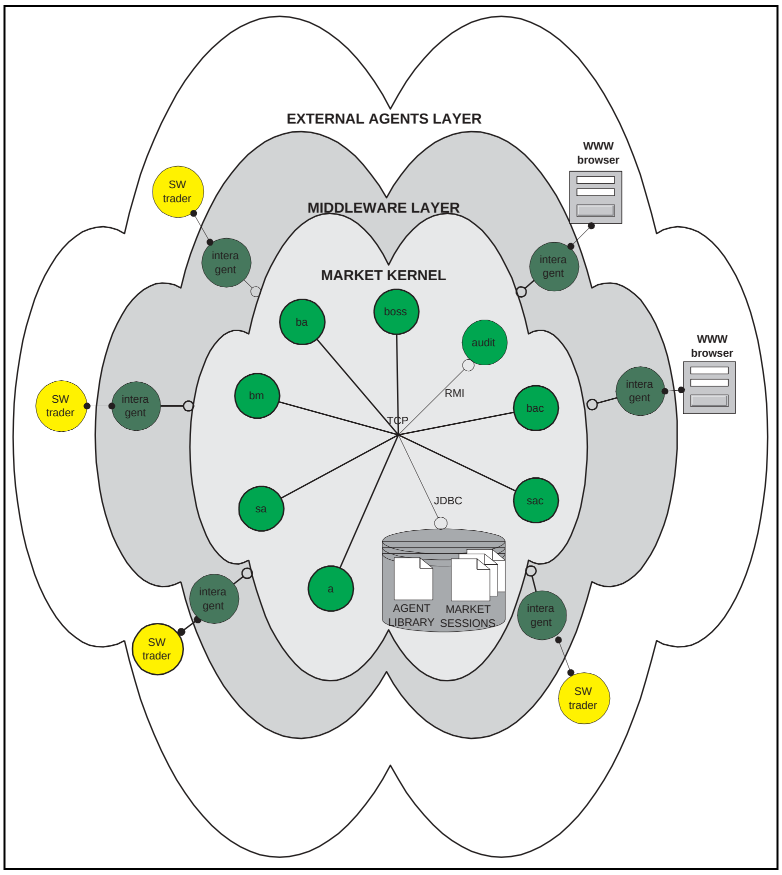


Figure 5.9: Market Architecture

Notice that in fact the institution is composed of the two inner layers. Notice also that all agents depicted above are assumed to be possibly running at different sites distributed over the Internet.

We believe that this multi-layered view of the architecture of FM96.5 can be readily generalised as the general architecture of an *AmEI*.

In our choice of development languages, we profited from our previous experiences too. Having already developed prototypes using C and Eu-Lisp [Padget et al., 1993] with PVM [Napoli et al., 1996, Gei, 1994] and MPI [Forum, 1994] (see Table 5.8), Java suggested relevant advantages [Gosling, 1996] that were worth testing in the implementation of FM96.5:

- it provides the advantages of object-oriented languages (reusability, modularity, encapsulation, etc.) and claims to be designed for maximum portability;
- its ease of programming and safety features help produce debugged code quickly shortening the development stage;
- it is highly powerful for constructing distributed applications;
- there is available an increasingly large collection of specialised packages which allow the programmer to count on powerful tools to successfully handle distributed computing [Sun, www, RMI, www], database connectivity [JDBC, www], security [SSL, www], graphics, etc.

Besides, given the recent industrial commitment and investment as well as generalised commercial activity around Java, it seems to us there is strong indication that Java may become a *de-facto* standard, therefore having permanence and complementary developments that would facilitate taking FM96.5 to a product-level stage.

5.7 Accountability

In Chapter 3 we identified heterogeneity, exception handling, societal change and trust and accountability as the major issues arising in the practical development of robust open agent organisations. So far fundamentally the use of interagents has allowed us to provide an answer to the first three issues. Nothing has been said yet about the not less important issue concerning trust and accountability within an agent-mediated electronic institution.

Trust and accountability must be regarded as a fundamental issue when practically deploying electronic institutions.

From an institution's point of view, monitoring is essential for detecting trading agents' deviating behaviours, and so subsequently gain protection against those that corrupt the trading activity. Moreover, monitoring the market activity can also help an institution to detect unexpected behaviours exhibited by its own agents, and act consequently in order to fix the defective institutional agents or interagents.

Trust builds upon the transparency that institutions demonstrate by facilitating accountability information to the trading agents involved in the market transactions. In general, any institution must specially care about guaranteeing the soundness of the services that it provides so that the participants trust it. But in the particular case of electronic institutions, gaining the confidence of humans appears as a delicate issue, specially if we consider that humans regard the events taking place in the institution as unobserved and they wonder whether they indeed ever occurred. Furthermore, we must not forget the fact that some humans may be participating in the market through some personal trading assistant. Evidently, trading assistants' owners will be extremely interested in checking whether they behaved as expected. And this is also a relevant matter from the point of view of the trading assistants' developers.

Technically speaking, monitoring the behaviour of multi-agent applications (and of agent-mediated computational institutions in particular) with distributed data, control and process is extremely difficult. However, the increasing number of multi-agent applications has made agent researchers face the matter of monitoring multi-agent systems (see f.i. [Ndumu et al., 1999, Kaminka and Tambe, 1999] for a sample of two very interesting approaches). Typically, agents are geographically distributed over a network, each one having its local view of the organisation. The challenge then is to integrate into a coherent whole the vast amounts of information provided by individual agents in order to:

- picture, understand and analyse the behaviour of the system as a whole;
- detect violations and eventual dysfunctional behaviours; and
- debug the system (even though the individual agents behave correctly, the system might not behave as expected).

In this section we describe a new institutional agent, the so-called *monitoring agent* or auditor[13], incorporated to FM96.5 so as to provide support for the monitoring of market sessions. One of the main tasks of the monitoring agent will be to collect and collate the information resulting from the interactions coming about in the market place in order to present it to the user in an understandable, appropriate manner.

Next, we succinctly summarise the main features of the monitoring agent:

- *on-line* (monitoring of *live* interactions in a market session) and *off-line* (video-like replay of saved market sessions) monitoring modes;
- monitoring mechanism based on the use of logical clocks;
- **Multi-level representation.** In order to visualise the operation of FM96.5, the monitoring agent defines a virtual market place divided into a set of virtual locations corresponding to their counterparts in the actual fish market. The virtual market place must be regarded as a blackboard employed

[13]In what follows we employ both terms indistinctly.

by the *monitoring agent* to depict in a centralised manner the distributed market activity. Within this setting, it offers two representation levels:

- **Societal (macro).** Visualisation of the *agent flow* (i.e., the flow of agents from virtual location to virtual location) and the main events taking place during an auction.
- **Agent-centered (micro).** Messages sent and received by each agent (agent-centered communication flow).

- **Customisable and Scalable**. Its design and implementation have succeeded in producing a flexible framework capable of adapting to future structural changes of the market-place.

The purpose of this section is to discuss in more detail the features of the *monitoring agent.* First, in Section 5.7.1 we start with some introductory material concerning logical time and logical clocks so that subsequently we can describe how these concepts have currently applied to the monitoring of market sessions. Next Section 5.7.2 argues on how to produce a flexible monitoring policy which allows to adapt the monitoring agent to changes in the structure and ontology of the market place. Section 5.7.3 provides a thorough description of the architecture and functionality of the actual implementation of the monitoring agent.

5.7.1 Logical Time and Logical Clocks

When considering any single process, events are solely ordered by times shown on the local clock. Nonetheless, Lamport pointed out that in general we cannot use physical time to find out the order of any arbitrary pairs of events occurring within a distributed system because we cannot synchronize clocks across the system [Lamport, 1978].

To order some of the events occurring at different processes, a scheme similar to physical causality, applied to distributed systems, can be utilized. The resulting ordering is based on two extremely intuitive assumptions:

- **Assumption #1.**Two events that occurred in the same process happened in the order in which this process observes them.
- **Assumption #2.**The event of sending a message occurs before the event of receiving it when the message is sent between processes.

The ordering resulting from the generalisation of these two relationships is the so-called *happened-before* relation, or also *causal ordering* or *potential causal ordering.*

Formally speaking, we will write $x \overrightarrow{p} y$ if two events x e y occurred in a single process p and besides x happened before y. From this, we define the $h$appened-before, denoted as $\rightarrow$, as follows:

(i) If $\exists p : x \overrightarrow{p} y \Rightarrow x \rightarrow y$.

(ii) For any message m, *send(m)* $\rightarrow$ *receive(m)*, where *send(m)* stands for the event of sending the message, while *receive(m)* for the event of receiving it.

(iii) *Transitivity.* If x, y and z are events such that $x \rightarrow y$ and $y \rightarrow z$, then $x \rightarrow z$.

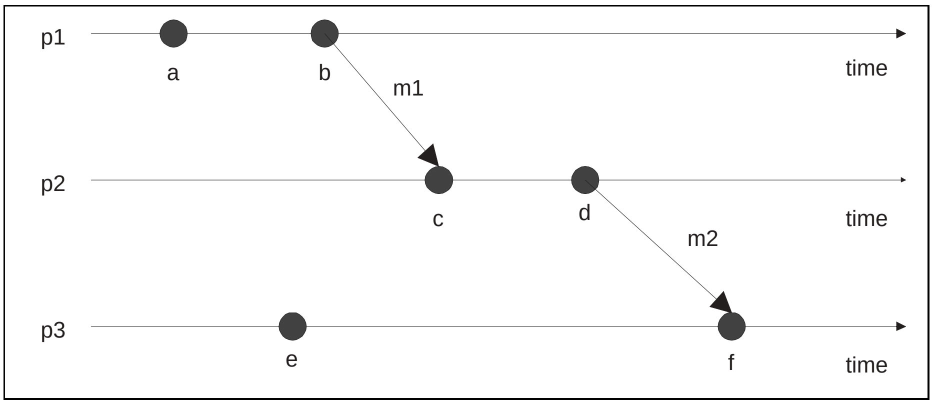


Figure 5.10: Events occurring in three processes.

Figure 5.10 illustrates the relation $rightarrow$ for three processes p_1, p_2 and p_3. We observe that $a \rightarrow b$, since the events follow this order in process p_1, and similarly $c \rightarrow d$; $b \rightarrow c$, corresponding to the sending and the reception of message m_1, and finally $d \rightarrow f$. The combination of these relations lead us to new conclusions. For instance, we can also assert that $a \rightarrow f$.

We can also infer from Figure 5.10 that not all events are related by $rightarrow$. For example, $a \rightarrow e$ and $e \rightarrow a$, because these two events occur in different processes and there is no chain of messages intervening between them. We will call *concurrent* those events not ordered by $\rightarrow$, and they will be denoted as $a \parallel e$.

Notice that:

- The relation $\rightarrow$ captures a flow of data intervening between two events.
- If the *happened-before* relation holds between two events, then the first might or might not have caused the second.

Logical Clocks

Lamport devised a simple mechanism for numerically capturing the *happened-before* ordering called *logical clock.* This is a monotonically increasing software counter generally unrelated to any physical clock. Each process keeps its own logical clock, C_p, to time-stamp events. Let us denote the time-stamping of event a in p as $C_p(a)$, and as $C(b)$ the time-stamping of event b in any process.

Processes update their logical clocks and transmit their values as follows so as to capture the *happened-before* relation:

LC_1: C_p incremented before each event is issued in process p: $C_p := C_p + 1$.

LC_2: (i) When a process p sends a message m, it includes in m the value $t = C_p$.

(ii) When a process q receives (m, t), it calculates $C_q = \max(C_q,\mathrm{t})$ and subsequently applies LC_1 before time-stamping *rcv(m)*.

It can be proved by induction on the length of any sequence of events relating two events a and b that $a \to b \Rightarrow C(a) < C(b)$. But more importantly observe that the converse is untrue. In other words, although $C(a) < C(b)$, we cannot infer that $a \to b$.

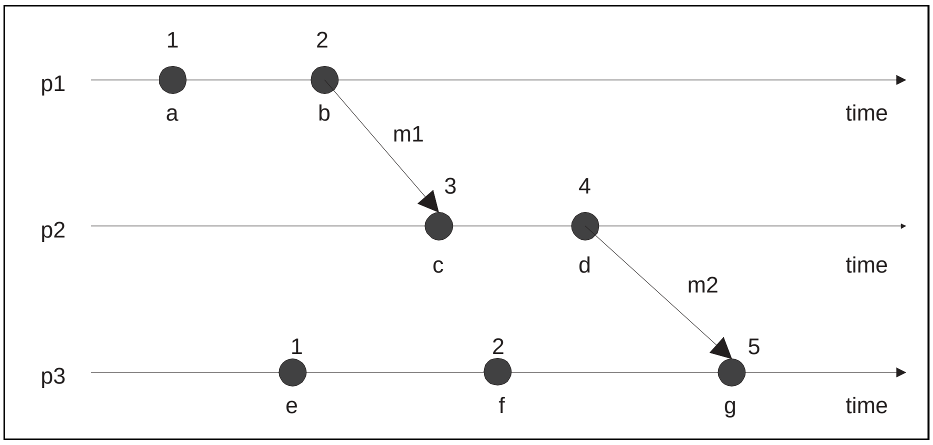


Figure 5.11: Logical times for the events in Figure 5.10

We exemplify the use of logical clocks in Figure 5.10. There, all processes' logical clocks are initially set to 0. The evolving clock values are represented immediately after the event to which they are adjacent.

Totally Ordered Logical Clocks

Observe that logical clocks impose a partial order on the set of all events. In order to obtain a total ordering, we simply consider the identifiers of the processes in which events occur. If a occurs at p_a at local time T_a and b occurs at p_b at local time T_b, we define the global logical time for these events to be *(T_a,p_a)* and *(T_b,p_b)* respectively. Thus $(T_a, p_a) < (T_b, p_b)$ iff either $T_a < T_b$, or $T_a = T_b$ and $p_a < p_b$.

5.7.2 Flexible Monitoring of Market Sessions

FM96.5 is a distributed system. Therefore, and based on the arguments introduced in the sections above, time-stamping each illocution uttered in the market will not help us much to obtain the ordering of the market events — agents will be running in different computers that contain their own physical clocks. Alternatively we propose to use logical time, instead of physical time, to time-stamp market illocutions.

In practice, the use of logical clocks by agents for time-stamping market illocutions is straightforward. When starting a new market session, both market intermediaries and interagents have their own local, logical clocks set to 0. During the market session, Lamport's algorithm is locally applied by them to tag both incoming and outgoing illocutions with the logical time of their logical clocks. In this manner, a partial ordering of the communication flow is built. Then the monitoring agent can employ the logical time accompanying each illocution to monitor a market session. In fact, monitoring market sessions fundamentally reduces to running a logical clock and depicting the illocutions occurring at the logical time displayed by the logical clock.

It is worth highlighting the capital role of interagents in the monitoring process. Apart from posting monitoring information to the auditor (as discussed later on in Section 5.7.3), they will also help detect trading agents' deviating behaviour. An interagent keeps account of all illocutions uttered by an agent, both valid and non-valid. While valid illocutions are conveniently sent to some institutional agent, non-valid illocutions are filtered out and rejected by the interagent itself. However, non-valid illocutions can be pictured during the monitoring process to show agents' dysfunctional behaviours.

Notice that the use of logical clocks does not (in general) help us order concurrent events[14], i.e. illocutions uttered in different scenes. However, considering that the boss, when acting as institution manager, must authorise agents to start or access scenes, the following orderings are guaranteed:

- the ordering of the events within every market scene;
- the order in which scenes are created and accessed; and
- the flow of trading agents from scene to scene.

Therefore, when employing logical time we cannot tell whether a buyer is bidding in the auction scene before or after another buyer updates his credit with the buyers' accountant. Nonetheless we can tell whether a scene started by a buyer to update his credit occurs prior to another scene started by a seller for having his goods registered. Furthermore, we can also tell whether a buyer paying for his goods did previously acquire it in the auction scene. And these are precisely the kind of orderings that we are interested in preserving for monitoring purposes.

Figure 5.12 illustrates how logical clocks are used in practice by the market intermediaries and trading agents' interagents. The diagram only depicts a

[14]We employ the term *concurrent* such as defined in Section 5.7.1.

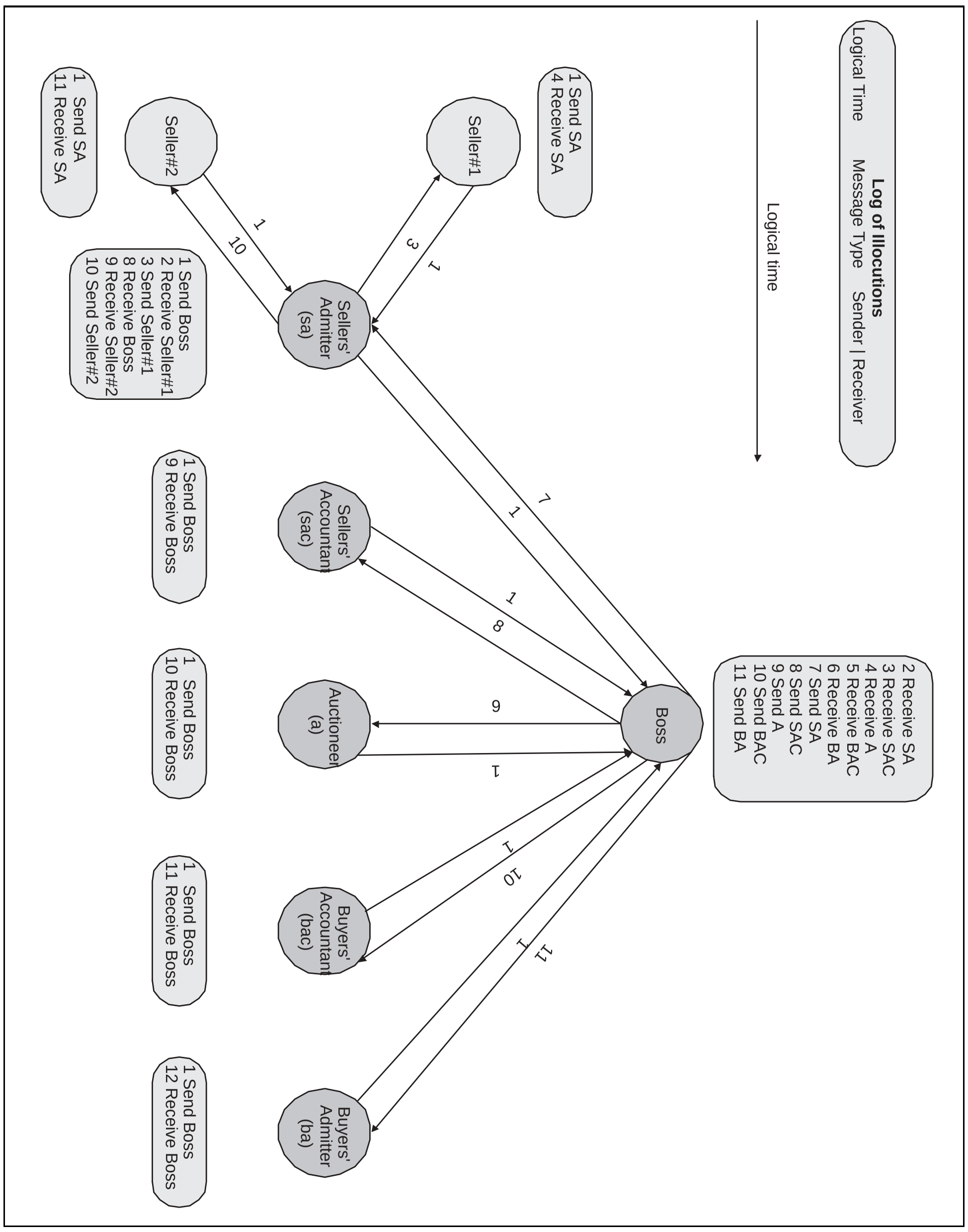


Figure 5.12: A sample of the use of logical times for ordering the exchange of illocutions among agents in the market place.

sample of the interaction among the market intermediaries and two sellers represented by their interagents. In the figure, agents are represented by circles while each rounded rectangle contains the history of illocutions sent and received by each agent. In fact, for the sake of simplicity, the histories do not contain the actual illocutions, but the logical time corresponding to either the sending or the reception as well as the sender or addressee.

So far we know how the events, illocutions, distributedly generated during a market session are partially ordered by employing logical times. Nevertheless still remains the matter of transforming this information into a centralised, understandable visual representation that pictures and helps us understand and analyse the behaviour of the system as a whole. Thus recall that we have demanded that the resulting representation captures both the agent flow (from virtual location to virtual location) and the communication flow (the exchange of illocutions among agents) within each scene composing the market activity. Furthermore, we also intend that the monitoring of the market activity help us detect defective behaviours, particularly those appearing on the external agents' side.

Therefore we regard the construction of a visual picture of the market activity as the main function of the monitoring agent. For this purpose, it will be employing a virtual market place composed of virtual locations (the counterparts of the places composing the actual fish market). The virtual market place must be regarded as a blackboard on which the distributed market activity is depicted in a centralised manner. Figure 5.13 depicts the GUI representing the virtual market place used by the monitoring agent divided into the following virtual locations: *out of market*, *staff room*, *buyers' admission room*, *sellers' admission room*, *buyers' settlements room*, *sellers' settlements room*, and *auction room*. Observe that agents are represented by different icons, depending on the role that each one plays.

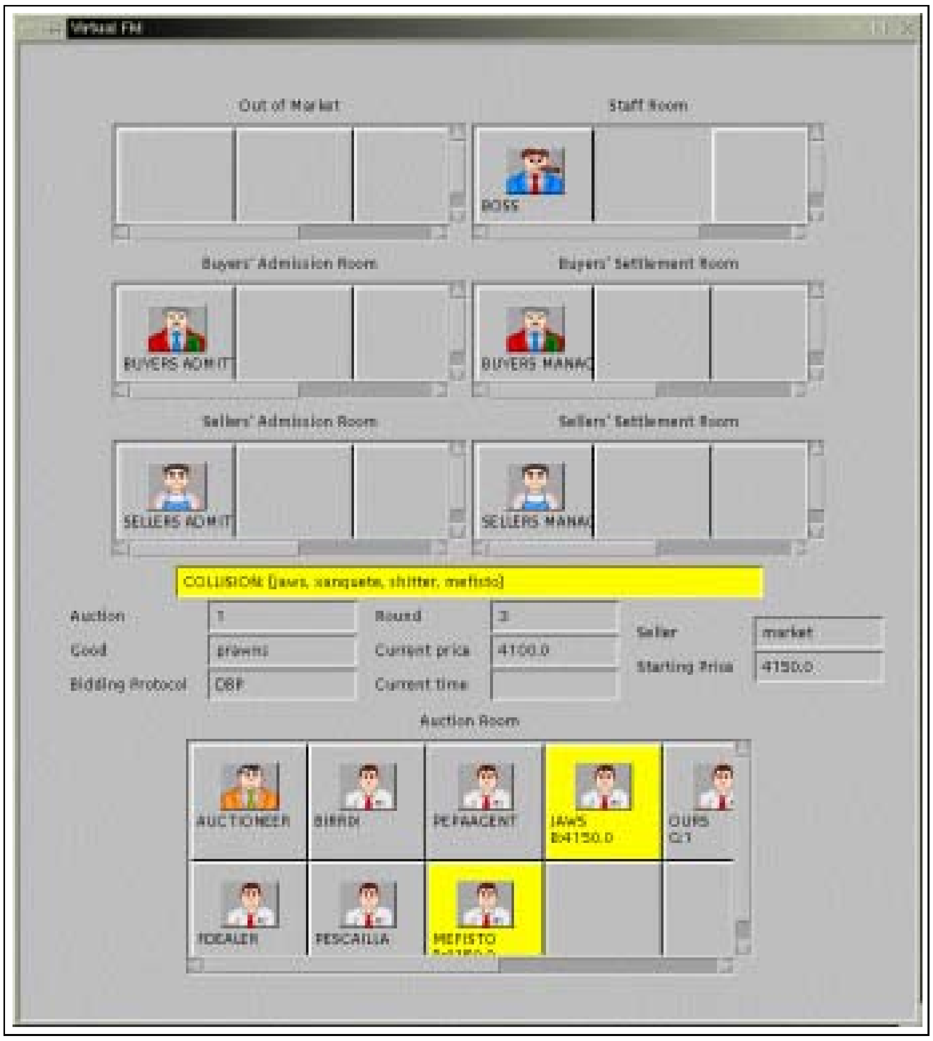
Apart from displaying how agents move from virtual location to virtual location, the monitoring agent also must show the illocutions sent and received by each agent within the virtual market place as shown in Figure 5.14. Each illocution is either sent or received in the context of a particular scene occurring in the virtual location wherein the sender/receiver is situated.

The way the monitoring agent represents the market activity depends on several factors:

- the virtual locations composing the virtual market place;
- how agents move from a virtual location to another; and
- the scenes considered to be taking place within each virtual location.

Notice that in fact scenes are logically grouped into virtual locations. Therefore the transition of an agent from a virtual location to another is equivalent to going from one scene to another belonging to a different virtual location.

It is apparent that the monitoring agent must count on a specification similar to the performative structure that contains the virtual locations composing the



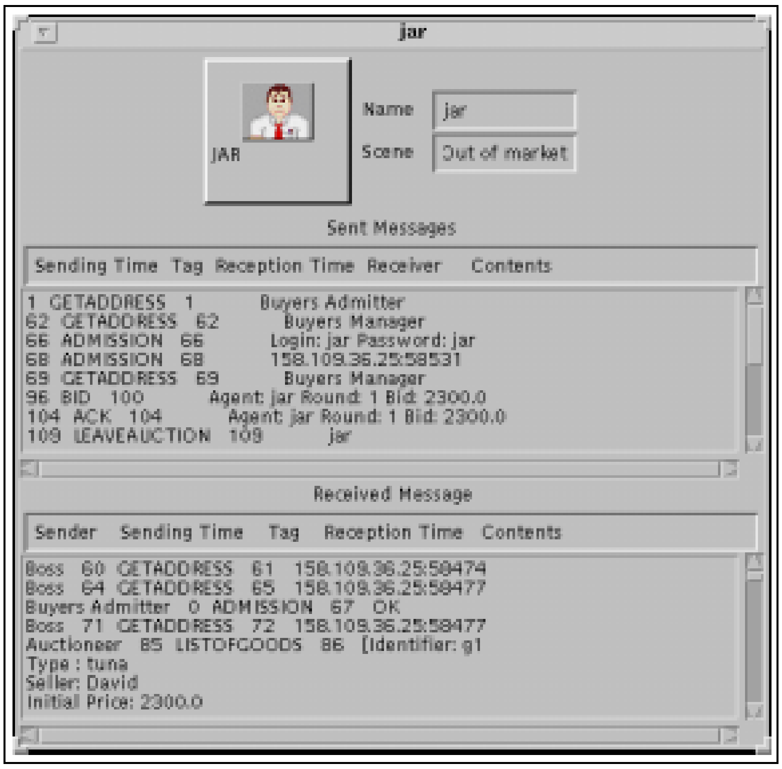
COLLISION: [jaws, xanquete, shitter, mefisto]

Auction	1	Round	3	Seller	market
Good	prawns	Current price	4100.0	Starting Price	4150.0
Bidding Protocol	DBP	Current time			

JAWS B:4150.0

OURS Q:1

Figure 5.13: Virtual locations composing the virtual market place.



Name jar

Scene Out of market

Sent Messages

```
Sending Time  Tag  Reception Time  Receiver    Contents
1  GETADDRESS  1        Buyers Admitter
62  GETADDRESS  62        Buyers Manager
66  ADMISSION  66        Login: jar Password: jar
68  ADMISSION  68        158.109.36.25:58531
69  GETADDRESS  69        Buyers Manager
96  BID   100          Agent: jar Round: 1 Bid: 2300.0
104  ACK   104          Agent: jar Round: 1 Bid: 2300.0
109  LEAVEAUCTION  109          jar
```

Received Message

```
Sender   Sending Time   Tag   Reception Time  Contents
Boss   60  GETADDRESS   61   158.109.36.25:58474
Boss   64  GETADDRESS   65   158.109.36.25:58477
Buyers Admitter   0  ADMISSION   67   OK
Boss   71  GETADDRESS   72   158.109.36.25:58477
Auctioneer   85  LISTOFGOODS   86   [Identifier: g1
Type : tuna
Seller: David
Initial Price: 2300.0
```

Figure 5.14: Local communication flow and virtual location of a trading agent.

market place, the scenes contained within each location, and the connections among scenes belonging to different locations. In this manner the monitoring is not sensitive to future changes in the market structure.

Although this is the most general approach, the current version of the monitoring agent built into FM96.5 departs from some simplifications. However, the description that follows abut the monitoring process in FM96.5 must not be understood as an *ad-hoc* solution. On the contrary, in fact the lessons learned in deploying the monitoring agent in FM96.5 have lead us to the observations in Section 5.7.4. There we discuss the drawbacks of the monitoring such as realised for FM96.5 and propose how to solve them in order to monitor *AmEIs* in general.

In practice, monitoring the behaviour of market intermediaries is fairly simple. Thus, upon registration market intermediaries make for the virtual locations containing scenes in which they are involved, and they remain there till the market closes. However, things get more complicated when considering trading agents. They may move from scene to scene (and eventually from virtual location to virtual location) back and forth, and even abandon an ongoing market session to join it back later on.

Currently the monitoring agent in FM96.5 solely handles specifications of communication behaviours corresponding to trading agents. In fact, the monitoring agent counts on the specification of the global, communication behaviour of trading agents' interagents. This is motivated by the need of capturing both the trading agent—interagent communication flow and the trading agent—institutional agents communication flows. In this manner, for each scene state an interagent can tell whether a given trading agent attempted to produce a non-legal illocution or even if it went down. Therefore, monitoring the activity of market intermediaries is somewhat hard-wired while the monitoring of the activity of trading agents is flexible and interpreted.

Since trading agents can only be present in a single scene, and so in a virtual place, we have opted for an FSM model to encode the specification of a trading agents' interagent communication behaviour. Concretely we employ an extension of a Mealy-like automaton model, a deterministic automaton model with output. Thus the communication behaviour of trading agents' interagents will be specified as a tuple $\langle Q, \Sigma, \Omega, \delta, \lambda, \rangle$ such that:

- Q is the set of internal states. It stands for the possible communication states of a trading interagent during a market session.

- Σ is the input alphabet, containing patterns of illocutions, i.e. illocutions' schema, represented according to the following syntax:;

  $$(< particle >, < predicate >, < argument\_type >, [values(: key < value >^{*})])$$

  where *particle* stands for an illocutionary particle, *predicate* stands for a predicate name, *argument_type* stands for the type of the argument, and $[values(: key < value >^{*})]$ is an optional element denoting a list of values.

- Ω is the output alphabet, the set containing all virtual locations of the market place in which the trading agent using the interagent can participate.
- δ is the transition function: $\delta : Q \times \Sigma \rightarrow Q$. The transition function δ determines the progress of a trading agent's interagent from a communication state to another when either sending/receiving illocutions to/from trading agents and to/from institutional agents.
- λ is the output function: $\lambda : Q \times \Sigma \rightarrow \Omega$. The output function λ yields the new virtual place wherein the agent is to be situated after a state transition. In this way, we capture the agent flow from virtual location to virtual location within the marketplace. Moreover, it provides the monitoring agent with the information necessary to visually represent the location of the agent at each moment.

Before proceeding further, the definition above deserves some comments concerning the terms employed by the input alphabet. In order to comprehend why we employ such alphabet, we must take into account that we are trying to specify the behaviour of an interagent by representing the exchange occurring between the interagent and the market. Thus such specification puts aside the exchange of messages between a trading agent and the interagent and exclusively considers the communication flow between the interagent and the market. It is in the context of such exchange that the chosen alphabet makes sense since the contents of the messages sent and received by an interagent have been chosen to be encapsulated into objects.

But surely the best way to illustrate how to specify a trading agent's interagent behaviour, and so the notions introduced so far, is by example. Then Figures 5.15, 5.16 and 5.17 depict part of the specification corresponding to a buyer's interagent by means of an automaton[15]. Some simplifications have been introduced in this example. The most important has to do with the auction room, wherein, for the sake of simplicity, only the downward bidding protocol is considered. Furthermore, the *out_of_market* virtual location is considered.

Observe that Figures 5.15, 5.16 and 5.17 consider not only the illocutions that a buyer can utter but also its interagent's illocutions.

The set of states $Q = \{b_0, \ldots, b_{27}\}$ corresponds to the communication states of the interagent during its interactions with the institutional agents. The output function is represented by shading the area containing the communication states occurring at a same virtual location. In the particular case of buyers, the set of virtual places is defined as $\Omega = \{auction\_room, buyer\_admission, buyer\_settlement, out\_of\_market\}$. In the picture, labellings preceded by / stand for the reception of a message by the interagent, while labellings followed by / denote the sending of a message by the interagent. Illocutions sent by a buyer's interagent may correspond to either the translation of a legal illocution

[15]Notice that they all can be connected with one another in order to obtain a global specification.

uttered by the buyer or to an inner coordination illocution to coordinate the activity of institutional agents and interagents. For instance, in Figure 5.15 the transition from b_0 to b_1 occurs when the interagent posts a request for admission submitted by the buyer to the market, while the transition from b_{12} to b_{13} in Figure 5.16 corresponds to a bid acknowledgment generated by the buyer itself without intervention of the buyer.

Each non-final state is also extended with a transition that leads to the very same state corresponding to non-legal messages uttered by the buyer.

Recall that we argued that interagents are enabled to deal with agent failures. Thus interagents were said to pro-actively leave the market on behalf of the agent undergoing a failure. Before leaving the market, the interagent communicates the failure so that the pending payments of a buyer remain annotated. This is captured by the transition between b_{19} and b_{20} in Figure 5.16.

At this point, we must think of a method for representing the FSM-based specification of an interagent' behaviour so that it can be interpreted by the monitoring agent. Figure 5.18 contains the translation from the automata in Figures 5.15, 5.16 and 5.17 to a plain text description that can be conveniently parsed by the monitoring agent. Observe that the information displayed in Figure 5.18 can be divided into two parts:

- **State-Place mapping.** In the section headed by the *PLACES* label, we find the specification of the virtual place by a buyer when its interagent reaches a given state $b_i \in Q$. In other words, it captures the output function of the automaton.
- **Automaton Representation.** In the section headed by the *STATES* label, we find the definition of the automaton. Notice that each line describes each internal state $b_i \in Q$ by encoding the transition $\delta(b_i, \iota)$ where $\iota \in \Sigma$. Each transition, corresponding to each one of the arcs in Figures 5.15, 5.16 and 5.17, is represented as:

  $$(< particle >, < predicate >, < arguments\_type >, [values(: key < value >^*)], < next\_state >)$$

  where $< particle >, < predicate >, < argument\_type >, [values(< value >^*)]$ stands for the input ι and $< next\_state >$ stands for the new internal state $\delta(b_j, \iota)$, where b_j is the internal state when receiving the input.

Observe that some states are not mapped to any virtual location. That is because those states correspond to transition states between two virtual locations. In practice, the monitoring agent does not depict a transition from a virtual location to another till the buyer is accepted in the target location.

5.7.3 Monitoring Agent Architecture

Figure 5.19 depicts the architecture of the monitoring agent or auditor included in FM96.5. Likewise the other institutional agents, the monitoring agent has been also implemented in Java.

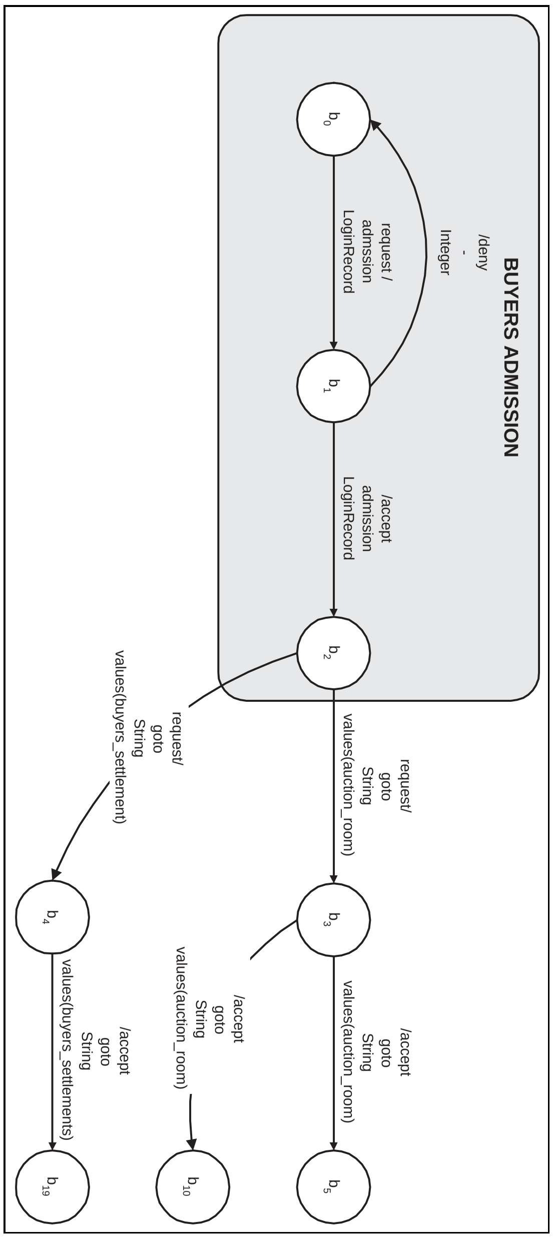


Figure 5.15: Communication behaviour of a buyer's trading agent in the admission room.

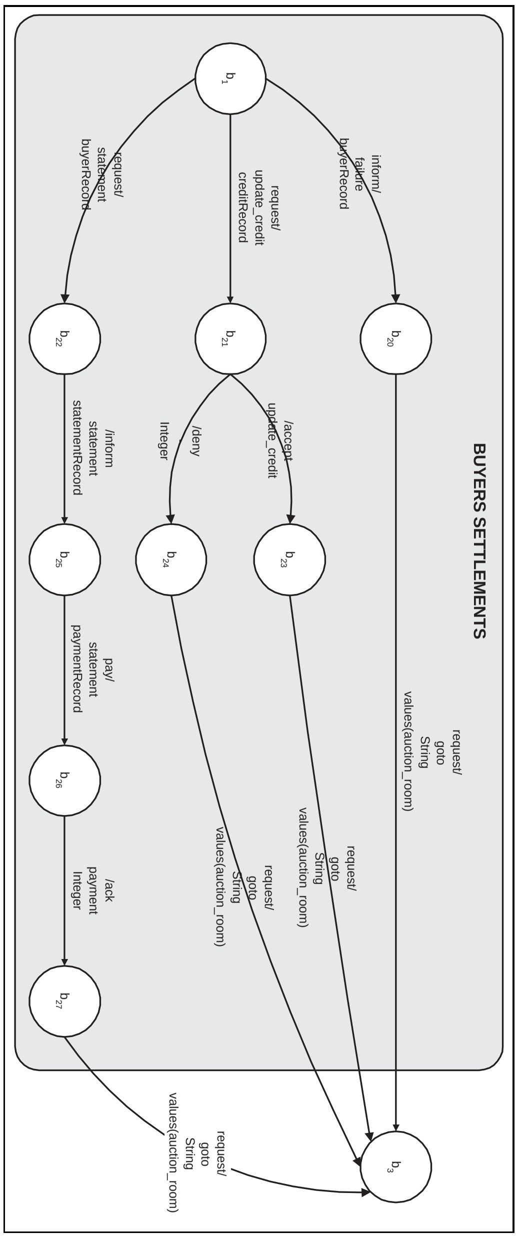

BUYERS SETTLEMENTS
b1
b20
b21
b22
b23
b24
b25
b26
b27
b3
inform/ failure buyerRecord
request/ update_credit creditRecord
request/ statement buyerRecord
/accept update_credit
/deny Integer
/inform statement statementRecord
pay/ statement paymentRecord
/ack payment Integer
request/ goto String values(auction_room)
request/ goto String values(auction_room)
request/ goto String values(auction_room)
request/ goto String values(auction_room)

Figure 5.16: Communication behaviour of a buyer's trading agent in the buyers' settlements room.

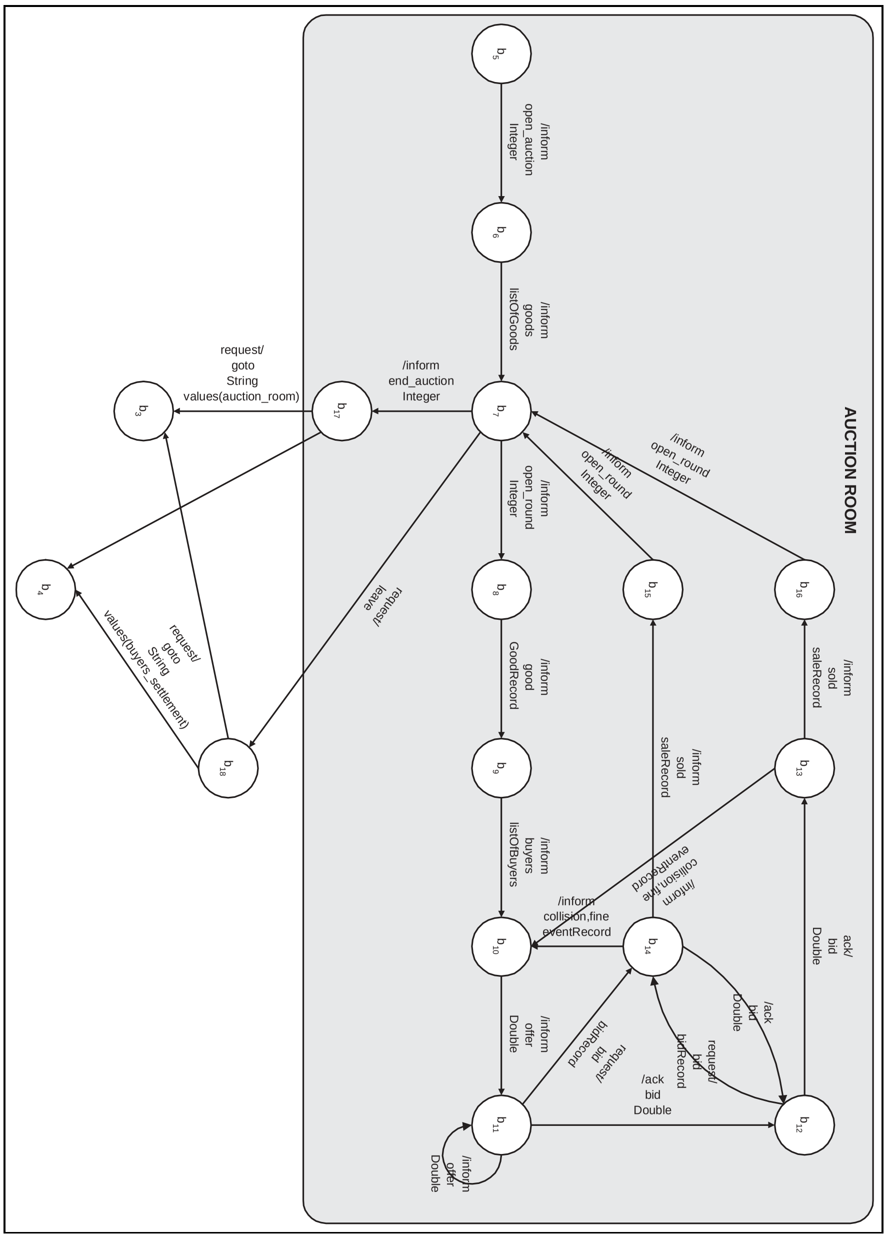

AUCTION ROOM
b5
/inform open_auction Integer
b6
/inform goods listOfGoods
b7
/inform open_round Integer
b8
/inform good GoodRecord
b9
/inform buyers listOfBuyers
b10
/inform offer Double
b11
/inform offer Double
/ack bid Double
b12
request/ bid bidRecord
/ack bid Double
request/ bid bidRecord
b14
ack/ bid Double
b13
/inform collision,fine eventRecord
/inform collision,fine eventRecord
/inform sold saleRecord
b15
/inform sold saleRecord
b16
/inform open_round Integer
/inform open_round Integer
/inform end_auction Integer
b17
request/ leave
request/ goto String values(auction_room)
b3
b18
request/ goto String values(buyers_settlement)
b4

Figure 5.17: Communication behaviour of a buyer's trading agent in the auction room.

```
PLACES
              buyers_admission b0 b1 b2
              auction_room b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16 b17
              buyers_settlements b19 b20 b21 b22 b23 b24 b25 b26 b27
STATES
              b0 (request,admission,structures.LoginRecord,b1)
              b1 (accept,admission,structures.marketRecord,b2) (deny,,Integer,b0)
              b2 (request,goto,String,values(auction_room),b3) (request,goto,String,values(buyers_settlements),b4)
              b3 (accept,goto,String,values(auction_room),b5)(accept,goto,sceneState,values(:scene auction_room),b10)
              b4 (accept,goto,String,values(buyers_settlements,b19)
              b5 (inform,open_auction,Integer,b6)
             . . .
             . . .
```

Figure 5.18: Encoding a buyer's interagent behaviour. A sample.

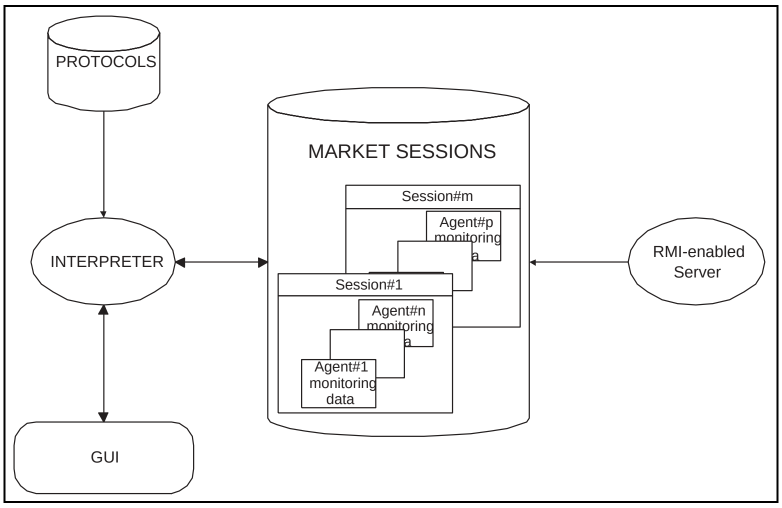


Figure 5.19: Monitoring Agent Architecture

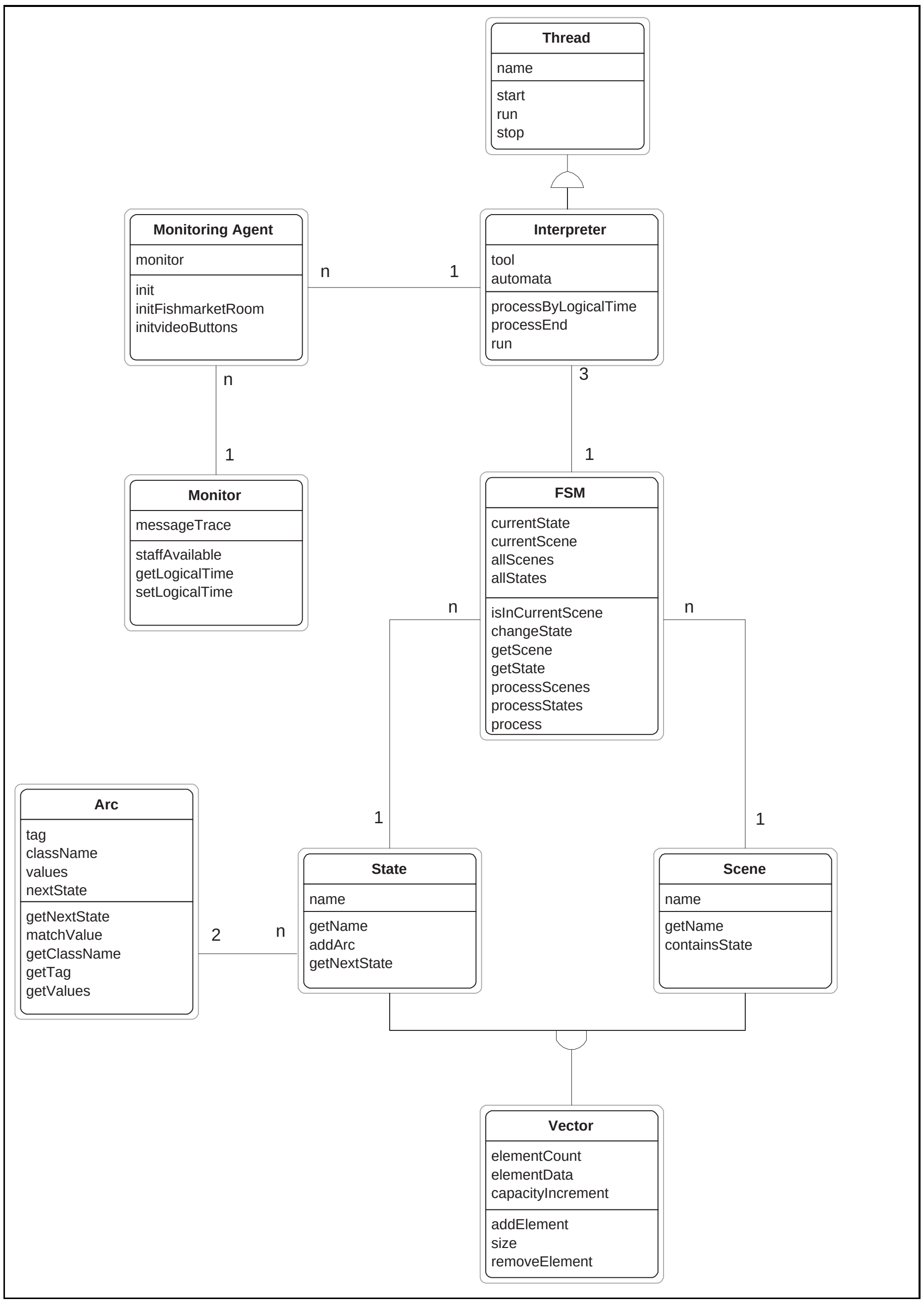


Figure 5.20: Coad design of the interpreter component of the monitoring agent.

Fundamentally, we distinguish five main components:

RMI-enabled server. A server in charge of collecting the monitoring information posted by market intermediaries and interagents. It has been realised as an RMI object whose methods can be remotely accessed via remote calls by market intermediaries and interagents. Notice that in this way institutional agents and interagents are responsible themselves of conveniently storing the monitoring information to be processed by the monitoring agent. For this purpose, both market intermediaries and interagents had their capabilities extended in order to enable them to post monitoring information via RMI calls.

Regarding interagents, observe that the incorporation of this extension habilitates them to hold three different communication channels:

- via pipes with trading agents;
- via TCP streams with institutional agents;
- via RMI with the monitoring agent.

Repository of market sessions. The repository is organised into directories, each one containing the monitoring information corresponding to a particular market session. Each market session is composed of a collection of files containing monitoring information. There are as many monitoring data files as agents participating in the market session. In general there are the monitoring data corresponding to the market intermediaries plus a varying number of monitoring data files corresponding to trading agents.

An agent's monitoring data file stores a list of all illocutions sent and received by the agent along with the logical time of either the sending or reception of each illocution.

Agents' Behaviour Library. Repository containing the specification of the trading agents' communication behaviour such as depicted in Figure 5.18.

Graphical user interface (GUI). The monitoring agent offers a graphical user interface composed of two main areas:

- the *virtual market place* (see Figure 5.13) on which both the agent and communication flow are graphically represented; and
- the *control panel* (see Figure 5.21), which allows for the interaction with the monitoring agent.

Since the monitoring agent can run as either a stand-alone application or an applet, participating agents can have access to the monitoring agent GUI through a standard browser in order to monitor live or past market sessions.

On the one hand, the control panel permits users interacting with the monitoring agent the video-like replay of live and saved market sessions. On

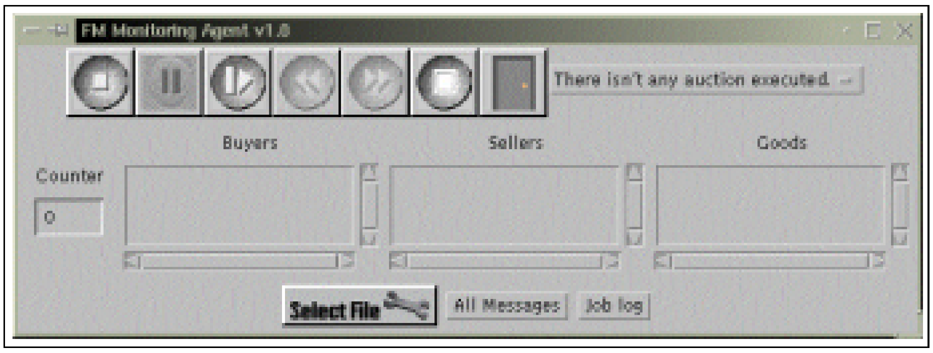


Figure 5.21: Monitoring Agent Control panel.

the other hand, apart from showing the agent flow, along with the most significant events occurring during a bidding round, the virtual market place panel also allows to access to agent-centered views of the communication flow (see Figure 5.14 where all illocutions sent and received by a given agent are shown).

Interpreter. The interpreter is a fundamental component of the monitoring agent. At the outset, the monitoring agent initialises the monitoring session by processing the activation of the market, i.e. by situating each market intermediary within the virtual location wherein its activities take place. Next, the monitoring agent starts an interpreter for each monitoring data file corresponding to a trading agent. Each interpreter runs in a separate thread processing the illocutions stored in the monitoring data file corresponding to its trading agent. An illocution is processed by an interpreter when the illocution's logical time is equal to the global logical time counted by the monitoring agent. If so, a transition in the automaton representing the communication behaviour of the agent occurs and probably a change in its virtual location. Therefore, the illocution is depicted in the virtual market place and also the transition of the agent from a virtual location to another when required.

Figure 5.20 shows the compositional design of the interpreter component. Observe that an interpreter is part of the monitoring agent and may include several finite state machine (FSM) specifications composed of states and arcs.

The RMI-enabled server, the GUI and the interpreters run concurrently thanks to the multi-threaded architecture of the monitoring agent.

5.7.4 Monitoring Agent-mediated Electronic Institutions

There are two major drawbacks of our approach for monitoring market sessions. On the one hand, it is not valid if we consider that trading agents may participate in multiple scenes (possibly belonging to different virtual locations) at the same time. On the other hand, the institutional agents' behaviour is not specified likewise trading agents' behaviour.

But perhaps the first drawback is the most important, though we think that can be overcome in a rather straightforward manner.

The monitoring of market sessions in FM96.5 cannot be generalised for monitoring *AmEIs* in general. This is mainly due to the fact that we originally chose illocutions as the atomic unit of monitoring information. Instead, we propose that interagents also enclose the scene wherein each illocution is uttered when posting monitoring information. Therefore, the monitoring agent would be capable of depicting the behaviour of each agent based on the specification of scenes, the monitoring information posted by interagents and institutional agents and a mapping from scenes to virtual locations that indicates where to virtually situate an agent participating in a given scene.

5.8 Summary

In this chapter we have put in practice the general computational model for building *AmEIs* proposed in Chapter 4 by deploying FM96.5, an actual agent-mediated electronic auction house. Some distinguishing features of the resulting electronic institution include:

- fair, live downward-bidding protocols;
- support for the participation of human and software traders of any type (openness);
- the accountability of the market activity that permits the global and local monitoring of market sessions.

The two first features are worth highlighting since FM96.5 was one of the first sytems achieving both. Nowadays there are more auction sites running live auctions (see for instance the auctions conducted at [Amazon, www]). Nonetheless to the best of our knowledge AuctionBot and FM96.5 were the first auction houses to permit the participation of software trading agents in market sessions. Furthermore, in both cases explicit support is provided for the development of software agents.

Originally, FM96.5 exclusively included a single auction protocol, namely the downward-bidding protocol. Thanks to the flexible design pursued in its development right from the beginning, it was readily evolved to include other auction protocols that allow it currently to operate as a multi-protocol auction house.

Version	Place	Language	Concerns	Advantages
FM96.0	IIIA	C (CGI)	Fast development	Demonstrability
FM96.1	IIIA & Naples	PVM	Synchronisation, Bidding protocol	Proof of concept
FM96.2	IIIA-Bath	C/MPI	Open Network	Portability
FM96.3	IIIA-Bath	C/MPI	Scalability, Market functionality	Modularity
FM96.4	IIIA-Bath	EULisp/MPI	Protocols	Expressiveness
FM96.5	IIIA-Bath	Java	Modularity, concurrency, functionality, fairness, livelihood of bidding protocol mediation, accountability	Full functionality Robustness Extendibility

Table 5.8: History of the implementations produced in the course of the fish market project.

Undoubtedly interagents are the key element in the development of FM96.5. Thus the use of mediation by means of interagents has demonstrated its usefulness along several directions:

(i) to achieve the openness of the market, allowing the participation of heterogeneous agents;

(ii) to contribute to the accountability processes undertaken by the market auditor (monitoring agent);

(iii) to locally manage eventual trading agents' exceptions so that the global market activity is not jeopardised; and

(iv) to be prepared to accommodate societal changes by managing the projections of the performative structure for trading agents as specifications.

Observe that the use of interagents allows us to provide answers to the questions posed in Chapter 3 when analysing the difficulties inherent to the development of open agent organisations.

At this point, we must acknowledge that all the advantages and niceties of FM96.5 did not come for free. They are the result of a systematic analysis of a more general problem –that of structured agent interactions– and the empirical testing of the conviction that multi-agent systems can be fruitfully applied to model and automate social interactions, such as the ones present in electronic trading. In [Napoli et al., 1996] a prototype implementation of a simple version of the fish market was presented. FM96.5 is a far more thorough implementation. In between we have addressed different aspects of the problem all in the framework of the *fish market project* [Fishmarket, www], and gone through the exercise of exploring specific technical or methodological issues (as shown in Table 5.8). Therefore, FM96.5 must be regarded as the resulting implementation emerging from of our previous empirical study.

Chapter 6

An Auction-based Agent-mediated Test-bed

In this chapter we present a framework for experimenting with auction-based trading scenarios. In these scenarios, trading (buyer and seller) heterogeneous (human and software) agents of arbitrary complexity participate in auctions under a collection of standardised market conditions and are evaluated according to their actual market performance. We argue that such competitive situations constitute convenient problem domains in which to study issues related with trading agent architectures in general and agent-based trading strategies in particular.

The proposed framework, *FM*, conceived and implemented as an extension of FM96.5 —the agent-mediated e-auction house described in Chapter 5—, constitutes a test-bed for trading agents in auction tournament environments.

We illustrate the practical usage of the resulting framework by commenting on the experiences acquired from actual tournaments hosted by FM involving some rudimentary buyer agents.

6.1 Motivation and Goals

Auctions are an attractive domain of interest for AI researchers in at least two areas of activity: web-based trading and resource allocation.

From the point of view of multi-agent interactions, auction-based trading is deceivingly simple. Trading within an auction house demands from buyers merely to decide on an appropriate price on which to bid, and from sellers, essentially only to choose a moment when to submit their goods. But those decisions —if rational— should profit from whatever information may be available in the market: participating traders, available goods and their expected re-sale value, historical experience on prices and participants' behaviour, etc. However, richness of information is not the only source of complexity in this domain. The actual conditions for deliberation are not only constantly changing and highly uncertain —new goods become available, buyers come and leave, prices keep on changing; no one really knows for sure what utility functions other agents have, nor what profits might be accrued— but on top of all that, deliberations are significantly time-bounded. Bidding times are constrained by the bidding protocol which in the case, for instance, of Dutch auctions —like the traditional fish market— proceeds at frenetic speeds.

Consequently, if a trading agent intends to behave aptly in this context, the agent's decision-making process may be quite elaborate. It could involve procedural information—when to bid, how to withdraw—, reasoning about individual needs and goals, information and reasoning about supply and demand factors— which may involve other agents' needs and goals— and assessment of its own and rivals' performance expectations—which in turn may require knowledge or reasoning about the external conditions that might affect the auction.

Evidently, many approaches can be taken to deal with this decision-making process. From highly analytical game-theoretic ones, to mostly heuristic ones. From very simple reactive traders, to deliberative agents of great plasticity. Moreover, it should be noted that the type of decision-making process involved in auctions is inherent in other common forms of trading and negotiation, and specifically in those that are being identified as likely applications of multi-agent systems [Giménez et al., 1998,Ygge and Akkermans, 1997,Huberman and Clearwater, 1995,Wellman, 1993]. However, it is not really obvious which of the many possible approaches for automatic trading strategies' modelling are better, or under what conditions. We do not intend to present any such evidence in this thesis, but instead to sketch a blueprint for the production, assessment and perhaps communication of such evidence. Actually, this chapter will focus on the description of a testbed—which permits the definition, activation and evaluation of experimental trading scenarios that we shall refer to as *tournaments*—and will illustrate how it can be used.

As the starting platform for that test-bed, we use FM96.5, the agent-mediated e-auction house thoroughly described in Chapter 5. Departing from FM96.5 we have constructed a framework wherein agent designers can perform *controlled experimentation* in such a way that a multitude of experimental market scenarios of varying degrees of realism and complexity can be specified, activated, and recorded; and trading agents compared, tuned and evaluated.

This exercise will ideally serve to show how one can conveniently devise experimental conditions to test specific features in agent architectures. How, for example, any-time strategies and off-line deliberation may be put to work coherently in a practical way. Or how and when reasoning about other agents' intentions and goals may be profitably turned into a trading advantage. Or how to couple a learning device with a human trader to discover market-dependent heuristics or with a trading agent so as to *watch* it perform the task. Or how to apply data mining techniques to discover patterns of behaviour of rival agents.

We trust this proposal may motivate AI theorists and developers to look into auctions as a challenging problem domain where they can investigate and put their creations through a strenuous test, but we realise that our proposed framework can serve other purposes as well. For instance, these tools may also interest economists who would like to examine issues of *Mechanism Design* under flexible theoretical and experimental conditions ([Varian, 1995]), since our trading scenarios may be seen as pseudo-markets with different degrees of indetermination. Moreover, financial regulatory bodies, and market developers may take advantage of this kind of framework for the design and experimentation with electronic market places, both in terms of those characteristics that new Internet-based trading institutions should have, but also in terms of features and components that new market practices may be requiring to facilitate agent-based trading that are practical, reliable and safe.

6.2 Test-bed Features

In order to obtain an auction tournament environment, more functionality has been added to FM96.5 to turn it into a *domain-specific* test-bed that models and simulates an e-auction house. A distinguishing feature of the resulting test-bed is that it is *realistic* since it has been built out of a complex real-world application.

Being an extension of FM96.5, FM inherits interagents, the mechanism of interaction between trading agents and the market. Consequently FM shows a crisp distinction between agents and the simulated world, a desirable requirement for any multi-agent test-bed. Furthermore, the use of interagents permits also to consider FM as an *architecturally neutral* environment since no particular agent architecture (or language) is assumed or provided, analogously to Tileworld [Pollack and Ringuette, 1990a], Tæms [O'Hare and Jennings, 1996], and Mice [Montgomery et al., 1992]. However, some support for agent developers is provided by including a library of agent templates in various languages (C, Java, and Lisp) for building agents. Furthermore, the test-bed also offers the possibility of generating customisable *dummy agents* at the aim of providing agent developers with contenders for training purposes.

FM inherits also all the auction protocols included in FM96.5, namely Dutch, English, First-price sealed-bid and Vickrey. All these auction protocols are classified as *single-sided* since bidders are uniformly of type buyer or uniformly of type seller.[1] *Double-sided* auctions admit multiple buyers and sellers at once. Figure 6.1 depicts a possible taxonomy for a small part of the auction space. The classification is made on the basis of whether the auction is single or double, bids are sealed (SB) or public (outcry), and prices are called in either ascending or descending order. FM contains the auction protocols hanging along the left branch, i.e. the classic auction types. Consequently FM can be classified as a multi-agent test-bed for classic auctions.

As to the systematisation of our experiments, the complete parametrisability of FM allows for the generation of different market scenarios. This capability of *scenario generation* appears as a fundamental feature of any multi-agent test-bed if it intends to guarantee the *repeatability* of the experiments to be con-

[1]Particularly single auctions have been the main focus of theoretical studies of auction [McAfee and McMillan, 1987].

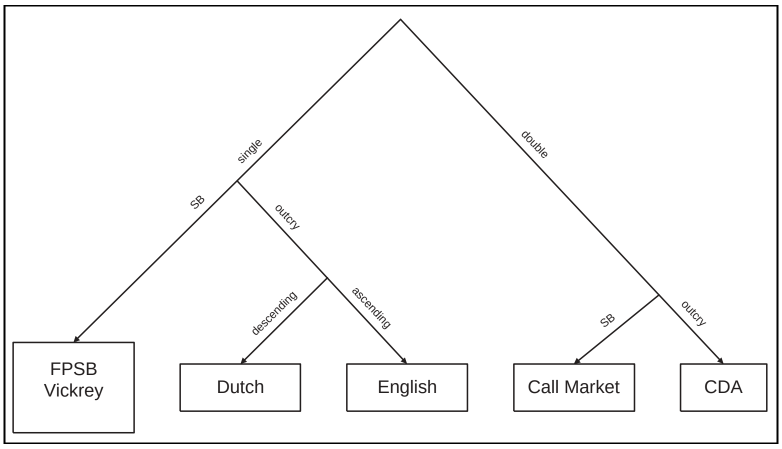
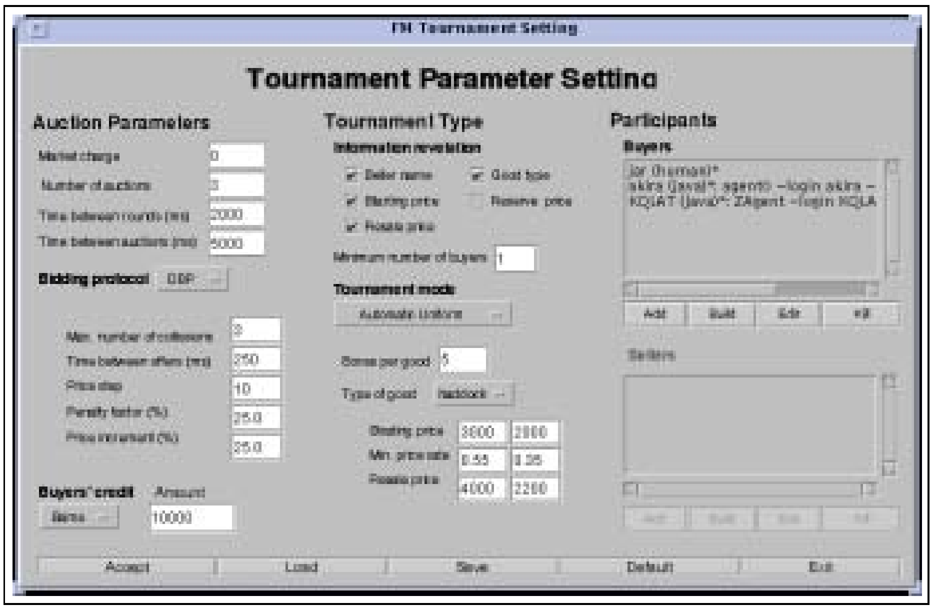


Figure 6.1: A classification of classic auction types [Wurman et al., 1998].

ducted likewise f.i. Tileworld [Pollack and Ringuette, 1990b], Phoenix [Cohen et al., 1989], DVMT [O'Hare and Jennings, 1996], TÆMS [O'Hare and Jennings, 1996]. Concretely, the customisability of FM allows for the specification, and subsequent activation, of a large variety of market scenarios: from simple toy scenarios to complex real-world scenarios, from carefully constructed scenarios that highlight certain problems to randomly generated scenarios useful for testing trading agents' average performance. Figure 6.2 displays a snapshot of the graphical display provided by FM to specify the particular features of a tournament scenario.

As to the matter of evaluating a trading agents' performance, FM keeps track of all events taking place during an auction, so that a whole auction can be audited step-by-step, and the evolving performance of all the agents involved in a tournament can be traced, calculated, and analysed. On the one hand, FM extends the database employed by FM96.5 such as shown in the design of Figure 6.3. On the other hand, the monitoring agent presented in Chapter 5 had its capabilities extended —likewise the rest of institutional agents as we discuss later on— in order to be capable of monitoring not only market sessions but tournament sessions. Furthermore, FM also offers graphical displays that plot the evolution of each agent during each competition.

Lastly we would like to mention a very important feature that seems to be somewhat skipped by test-bed designers: the problem of scalability. When running multi-agent experiments, an experimenter usually faces serious resource limitations that may prevent him from having all agents up and running. We say that FM is *scalability-aware* in the sense that it provides support for distributing an experimenter's agents across several machines in a network. This does not mean that all agents involved in a tournament must belong to the very same user. Tournament designers are free to define *open tournaments* accessible by agents owned by multiple users.


FM Tournament Setting

Tournament Parameter Setting

Auction Parameters

Market charge: 0
Number of auctions: 3
Time between rounds (ms): 2000
Time between auctions (ms): 5000
Bidding protocol: DBP
Min. number of collisions: 3
Time between offers (ms): 250
Price step: 10
Penalty factor (%): 25.0
Price increment (%): 25.0
Buyers' credit: Same; Amount: 10000

Tournament Type

Information revelation
✓ Seller name ✓ Good type
✓ Starting price ☐ Reserve price
✓ Resale price
Minimum number of buyers: 1
Tournament mode: Automatic Uniform
Boxes per good: 5
Type of good: haddock

Starting price	3000	2800
Min. price rate	0.55	0.35
Resale price	4000	2200

Participants

Buyers
jar (human)*
akira (java)*: agent0 –login akira –
KQIAT (java)*: ZAgent –login KQIA

Figure 6.2: FM Tournament Definition Panel

Notice that the resulting environment, FM, thus constitutes a multi-agent testbed where a very rich variety of experimental conditions can be explored systematically and repeatedly, and analysed and reported with lucid detail if needed. Table 6.2 summarises the features of FM.

6.3 Standard Market Conditions for Tournament Scenarios

A trading scenario will involve a collection of explicit conventions that characterise an artificial market. Such conventions define the bidding conditions (timing restrictions, increment/decrement steps, available information, etc.), the way goods are identified and brought into the market, the resources buyers may have available, and the conventions under which buyers and sellers are going to be evaluated. This proposal departs from our previous work presented in [Rodríguez-Aguilar et al., 1998c] and shares some commonalities with [Mullen and Wellman, 1996,Wurman et al., 1998] in the identification of auction parameters. In this section we discuss these underlying ideas from a formal point of view and introduce some of the elements needed to make precise specifications of actual tournament scenarios in Section 6.4. Any specification is intended to convey all the information necessary for a trading agent to participate in a

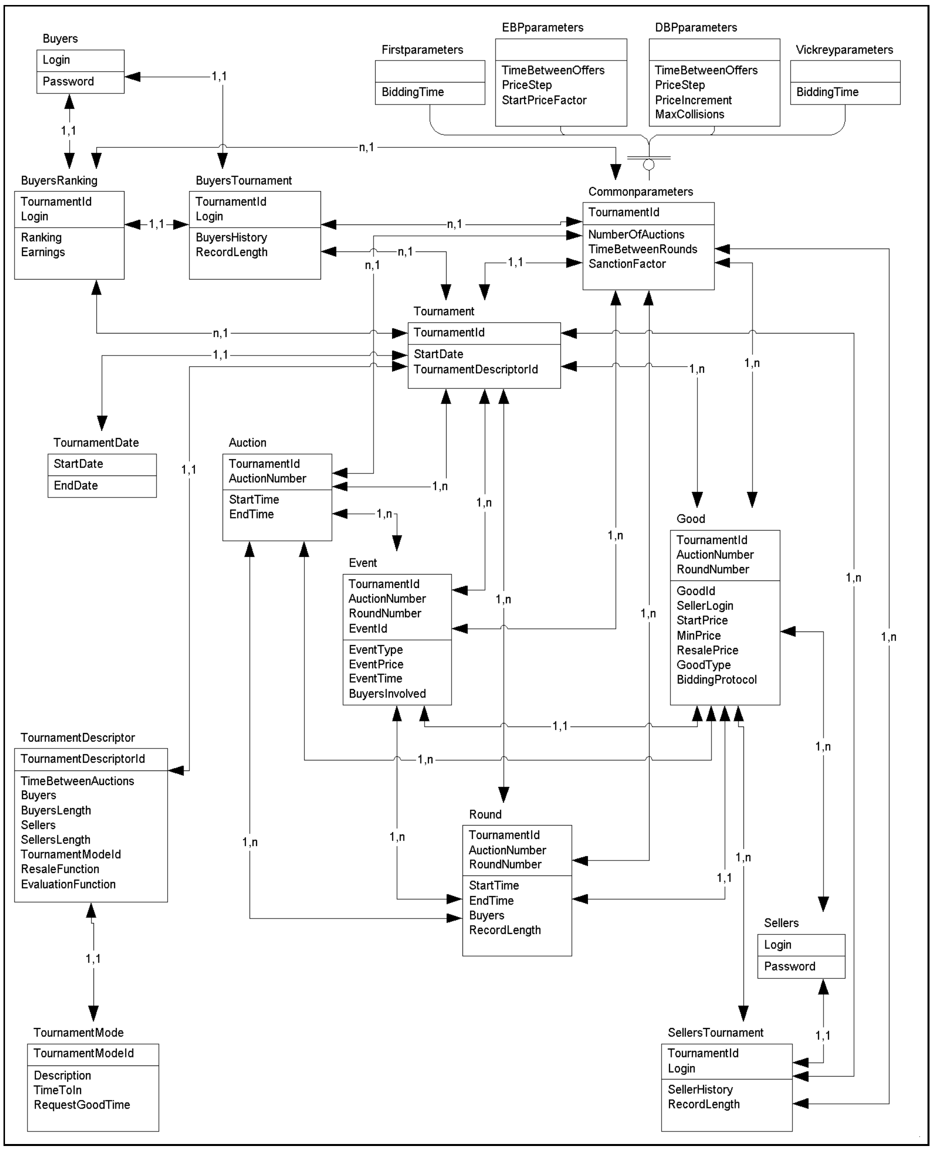


Figure 6.3: FM Database Entity-relation Database Model

Test-bed Features
■ domain-specific
■ realistic
■ architecturally neutral
■ scenario generation and reapeatibility capabilities
■ monitoring and evaluation facilities
■ library of agent templates (C,Java,Lisp)
■ dummy agents
■ scalability aware
■ open (multi-user) and closed (single-user) tournaments
■ market scenarios as tournament scenarios

Table 6.1: Features of the FM test-bed.

tournament.

We shall start by studying the characterizing parameters of the auction protocols governing the main activity within FM. For this purpose, next we introduce the notions of *round*, *auction* and *tournament* as the basic elements of a tournament scenario. Finally, we close this section defining how buyers and sellers may be evaluated.

6.3.1 Bidding Protocols' Descriptors

When auctioning a good, a seller can choose an auction protocol out of a set of auction protocols. Whatever the seller's choice, each auction protocol can be characterised by a set of parameters that we refer to as *bidding protocol dynamics descriptors*, so that different instantiations of such descriptors lead to different behaviours of their corresponding bidding protocols.

The algorithm in Figure 6.4 codifies the downward bidding protocol such as described in Section 5.5.1. The description helps us to explicitly identify the parametrisation of the bidding protocol.

Six parameters that control the dynamics of the bidding process are implicit in this protocol definition. We shall enumerate them now, and require that they become instantiated by the tournament designer as part of a tournament definition.

Definition 6.3.1 (DBP Dynamics Descriptor). We define a Downward Bidding Protocol Dynamics Descriptor $\mathcal{D}_{DBP}$ as a 5-tuple $\langle \Delta_{price}, \Delta_{offers}, \Sigma_{coll}, \Pi_{sanction}, \Pi_{rebid} \rangle$ such that

- $\Delta_{price} \in I\!\!N$ (price step). Decrement of price between two consecutive quotations uttered by the auctioneer.

```
Function round ($\mathcal{B}_r^i, g_r^i, p, coll, \mathcal{D}_{DBP}$) =
let Function check_credit($b_i$) =
 if $\mathcal{C}_r^i(b_i) \geq p$ then
   update_credit($b_i, p$);
   sold($g_i^r, b_i, p$);
 else if $\mathcal{C}_r^i(b_i) \geq p * \Pi_{sanction}$ then
       update_credit($b_i, p * \Pi_{sanction}$);
       round($B_r^i, g_r^i, p * (1 + \Pi_{rebid}), 0, \mathcal{D}_{DBP}$);
      else
       round($B_r^i - \{b_i\}, g_r^i, p * (1 + \Pi_{rebid}), 0, \mathcal{D}_{DBP}$);
in
 offer($g_r^i, p$);
 wait($\Delta_{offers}$);
 let $B = \{b_i | \mathbf{bid}(b_i) = \mathbf{true}, b_i \in B_r^i\}$ in
  case
   $||B|| = 0$ :  if $p = p_\omega$ then withdraw($g_r^i$);
                  else round($B_r^i, g_r^i, p - \Delta_{price}, 0, \mathcal{D}_{DBP}$);
   $B = \{b_i\}$ : check_credit ($b_i$);
   $||B|| > 1$ :  if coll $< \Sigma_{coll}$ then
                   round($B_r^i, g_r^i, p * (1 + \Pi_{rebid}), coll + 1, \mathcal{D}_{DBP}$);
                  else check_credit(random_select($B$));
  end case
 end
end

DBP($B_r^i, g_r^i$) = round($B_r^i, g_r^i, p_\alpha, 0$)
```

Figure 6.4: Downward bidding protocol

- $\Delta_{offers} \in I\!N$ (time between offers). Delay between consecutive price quotations.
- $\Sigma_{coll} \in I\!N$ (maximum number of successive collisions). This parameter prevents the algorithm from entering an infinite loop. This occurs when two or more buyers repeatedly submit the very same bid.
- $\Pi_{sanction} \in I\!R$ (sanction factor). This coefficient is utilized by the buyers' manager to calculate the amount of the fine to be imposed on buyers submitting unsupported bids.
- $\Pi_{rebid} \in I\!R$ (price increment). This value determines how the new offer is calculated by the auctioneer from the current offer when either a collision, or an unsupported bid occur.

Note that the identified parameters impose significant constraints on the trading environment. For instance, Δ_{offers} and Δ_{rounds} affect the agents' time-boundedness, and consequently the degree of situatedness viable for bidding strategies.

Analogously, we can also identify the parameters that determine the dynamics of the rest of auction protocols contained in FM. Thus, the following definitions account for the dynamic descriptors of the UBP, FPSB and Vickrey protocols.

Definition 6.3.2 (UBP Dynamics Descriptor). We define an Upward Bidding Protocol Dynamics Descriptor $\mathcal{D}_{UBP}$ as a 6-tuple $\langle \Delta_{price}, \Delta_{offers}, \Pi_{sanction}, \Pi_{start} B_t, \tau \rangle$ such that

- $\Delta_{price} \in I\!\!N$ (price step). Decrement of price between two consecutive quotations uttered by the auctioneer.
- $\Delta_{offers} \in I\!\!N$ (time between offers). Delay between consecutive price quotations.
- $\Pi_{sanction} \in I\!\!R$ (sanction factor). This coefficient is utilised by the buyers' manager to calculate the amount of the fine to be imposed on buyers submitting unsupported bids.
- $\Pi_{start} \in I\!\!R$ (price increment). This value determines how the starting price is calculated by the auctioneer from the reserve price.
- $B_t \in I\!\!N$ is the total bidding time.
- $\tau \in I\!\!N$ is the identifier of the tie-breaking method necessary to undo collisions when the closing time is reached.

Definition 6.3.3 (FPSB Dynamics Descriptor). We define both an FPSB Dynamics Descriptor $\mathcal{D}_{UBP}$ as a tuple $\langle \Delta_{bidding}, \Pi_{sanction}, \tau \rangle$ such that

- $B_t \in I\!\!N$ is the total bidding time.
- $\Pi_{sanction} \in I\!\!R$ (sanction factor) is the coefficient utilised by the buyers' manager to calculate the amount of the fine to be imposed on buyers submitting unsupported bids.
- $\tau \in I\!\!N$ is identifier of the tie-breaking method necessary to undo collisions when the closing time is reached.

Notice that a *Vickrey dynamics descriptor* is defined likewise an FPSB dynamics descriptor since both protocol are dynamically equivalent, only deferring in their resolutions.

6.3.2 Tournament Descriptor

By *auction rounds* we shall refer to the set of parametrisable ontological elements involved in each bidding round.

Definition 6.3.4 (Auction Round). For a given round r of auction i we define the *auction round* $\mathcal{A}_r^i$ as the 4-tuple

$$\mathcal{A}_r^i = \langle \mathcal{B}_r^i, g_r^i, \mathcal{C}_r^i, d_r^i \rangle$$

where

- $\mathcal{B}_r^i$ is a non-empty, finite set of buyers' identifiers such that $B_r^i \subseteq \mathcal{B}$, the set of all participating buyers.

- $g_r^i = \langle \iota, \tau, p_\alpha, p_{rsv}, s_j, p_\omega, p_{rsl}, b_k \rangle$ is a good where ι stands for the good identifier, τ stands for the type of good, $p_\alpha \in I\!\!N$ stands for the starting price, $p_{rsv} \in I\!\!N$ stands for the reserve price, $s_j \in \mathcal{S}$—the set of all participating sellers—is the seller of the good, $p_\omega \in I\!\!N$ stands for the sale price, p_{rsl} stands for the expected resale price, and $b_k \in \mathcal{B}_r^i$ is the buyer of the good. Notice that g_r^i is precisely the good to be auctioned during round r of auction i, and that p_ω and b_k might take on empty values when the round is over, denoting that the good has been withdrawn.

- $\mathcal{C}_r^i : \mathcal{B}_r^i \longrightarrow I\!\!R$ assigns to each buyer in $\mathcal{B}_r^i$ his available credit during round r of auction i.

- d_r^i stands for an instance of a bidding protocol dynamics descriptor.

Finally, our notion of *auction* is based on the definition above.

Definition 6.3.5 (Auction). We define an auction $\mathcal{A}^i$ as a sequence of *auction rounds*

$$\mathcal{A}^i = [\mathcal{A}_1^i, \ldots, \mathcal{A}_{r_i}^i]$$

Each auction is devoted to the auctioning of a particular *lot* of goods. Typically a tournament session (and a market session too) will be composed of a sequence of auctions such as defined above.

So far we have identified all the essential elements characterising bidding rounds: the participating buyers and their credits, the sellers and their goods and those features typifying the bidding protocol. On the basis of these definitions, we are ready to determine what elements and parameters are necessary to wholly characterise a tournament scenario, i.e. all the relevant information needed by an agent to participate in an auction-based tournament, compiled in the definition of *tournament descriptor*. A tournament descriptor is intended to be the sole information on which trading agents count prior to the starting of a tournament session.

Definition 6.3.6 (Tournament Descriptor). We define a Tournament Descriptor $\mathcal{T}$ as the 11-tuple

$$\mathcal{T} = \langle n, \Delta_{auctions}, \Delta_{rounds}, \mathcal{D}, \mathcal{P}_B, \mathcal{P}_S, \mathcal{B}, \mathcal{S}, \mathcal{F}, C, M, \epsilon, E \rangle$$

such that:

- n is the tournament length expressed either as the number of auctions to take place during a tournament or the closing time.

- $\Delta_{auctions}$ is the time between consecutive auctions.

- $\Delta_{rounds} \in I\!N$ (time between rounds) stands for the delay between consecutive rounds belonging to the same auction.
- $\mathcal{D}$ is a finite set of bidding protocols' dynamics descriptors.
- $\mathcal{PS}_B$ is the projection of the institution's performative structure from the point of view of buyer agents, i.e. the parts of the market accessible to buyers.
- $\mathcal{PS}_S$ is the projection of the instituion's performative structure from the point of view of seller agents, i.e. the parts of the market accessible to sellers.
- $\mathcal{B} = \{b_1, \ldots, b_p\}$ is a finite set of identifiers corresponding to all participating buyers.
- $\mathcal{S} = \{s_1, \ldots, s_q\}$ is a finite set of identifiers corresponding to all participating sellers.
- $\mathcal{F} = [\mathcal{F}^1, \ldots, \mathcal{F}^n]$ is a sequence of supply functions. A supply function $\mathcal{F}^i : (T \times \mathcal{S} \to I\!N \times I\!R \times I\!R \times I\!R) \to 2^{T \times \mathcal{S} \times I\!R \times I\!R \times I\!R}$ outputs the lot of goods to be auctioned during auction i, where T and $\mathcal{S}$ stand respectively for a finite set of good types and a finite set of seller identifiers. Each lot is composed of a set of goods described by a good type, a seller identifier, a starting price, a reserve price and a resale price. Depending on the supply function, part of the information describing a given good can be omitted.
- $C : \mathcal{B} \to I\!N$ is the credit initially endowed to each buyer. For some tournaments, all buyers are assigned the same credit, while for others they may either have assigned different credits or alternatively declare themselves the credit they want to have available.
- $M = \langle b, s, r, r' \rangle$ where $b, s, r, r' \in \{0, 1\}$ is the information revelation mask. It determines whether the identity of buyers (b) and sellers (s) is revealed to the contenders, and whether the reserve price (r) and expected resale price (r') of a good are revealed too.
- ϵ stands for the fees charged to an agent for participating in a bidding round.
- $E = \langle E_b, E_s \rangle$ is a pair of winner evaluation functions that permit to calculate respectively the score of buyers and sellers.

From the definition follows that a tournament descriptor contains:

- all the relevant parameters that characterise the dynamics of the auctioning process;
- the procedural information that allows trading agents to participate in the market by means of their interagents;

- the degree of information revelation (transparency) (i.e. the degree of uncertainty concerning the identity of traders and some particular, relevant features of goods); and
- the way the performance of trading agents is evaluated.

A remarkable aspect of the definition of tournament descriptors roots in the inclusion of the specification of performative structures for buyers and sellers. These are included to define which scenes are accessible to buyers and sellers during a tournament, and which protocols are employed within each one of these scenes. For instance, though evidently an auction scene is always considered, it may only include the auctioning of goods under a single or a limited collection of bidding protocols. As shown in next section, the tournaments hosted so far by FM include an auction scene in which goods are solely auctioned under the conventions of the downward bidding protocol.

6.3.3 Specifying Tournaments

It is the task of the experimenter to conveniently set up the parameters of the tournament descriptor in order to generate the desired type of tournament scenario. For this purpose, FM provides the graphical configuration tool shown in Figure 6.2 to assist the tournament designer to configure tournament scenarios.

Additionally FM incorporates the so-called *tournament modes* that constrain the type of tournament descriptor that can be defined. The purpose of this standard tournament modes is to allow an experimenter to define tournament scenarios of different degrees of complexity: from toy scenarios where, for instance, the same lot of goods is repeated over and over with complete information to actual-world auction scenarios.

Thus in FM tournament designers can choose among the following standard tournament modes:

One auction (data set) This mode permits a tournament designer to specify a fixed set of goods to be repeatedly auctioned a finite number of times. Notice that no sellers are involved in this type of tournament.

Automatic The lots of goods to be auctioned are artificially generated by the sellers' admitter based on supply functions of arbitrary complexity specified by the tournament designer in the set $\mathcal{F}$. Notice that likewise *one auction (data set)* no sellers are allowed to participate in these tournaments.

This tournament mode allows to artificially generate a large variety of markets. For instance, markets with more demand than supply or the other way around, markets with high quality goods more appropriate for restaurant owners, or markets with large supply of low-quality goods more appropriate for wholesale buyers[2]. In general, this tournament mode allows to create tournaments focusing on particular market scenarios.

[2]Note that for all the examples we consider fishmarket-like tournament scenarios.

Type	*#Boxes*	Starting price(p_α)	Resale price(p_{rsl})	Reserve price(p_{rsv})
cod	$U[1,15]$	$U[1200,2000]$	$U[1500,3000]$	$U[0.4,0.5]$
tuna fish	$U[1,15]$	$U[800,1500]$	$U[1200,2500]$	$U[0.3,0.45]$
prawns	$U[1,15]$	$U[4000,5000]$	$U[4500,9000]$	$U[0.35,0.45]$
halibut	$U[1,15]$	$U[1000,2000]$	$U[1500,3500]$	$U[0.4,0.6]$
haddock	$U[1,15]$	$U[2000,3000]$	$U[2200,4000]$	$U[0.35,0.55]$

Table 6.2: Example of uniform distributions employed to generate heterogeneous lots.

Uniform This mode is a particular case of the preceding tournament mode. Lots of goods are randomly generated by the sellers' admitter based on uniform distributions in $\mathcal{F}$ defined by the tournament designer. Notice that again no sellers are involved in the resulting tournaments either. Table 6.2 shows some examples of uniform distributions that can be employed for generating lots of goods.

This tournament mode is intended to generate scenarios wherein the average performance of buyer agents can be tested. Along with *one auction (data set)* it must be considered as a mode to generate game-like scenarios.

One auction (with sellers) Once all participating sellers have submitted their goods, the same auction is repeated over and over with the same lot of goods. This tournament mode is particularly useful to test the adaptivity of trading agents to an actual market scenario.

Fishmarket The mode closest to the workings of an actual auction house[3]. The tournament designer simply specifies the starting and closing times. During that period of time buyers and sellers can enter, submit goods, bid for goods, and leave at will. *Fishmarket* is the more realistic mode, standing for an actual market scenario.

Depending on the tournament mode chosen by the experimenter, some features of the tournament descriptor will be either enabled or disabled in the *parameter setting panel* at Figure 6.2. Notice that all parameters identified as part of the tournament descriptor lie down on the *parameter setting panel.*

Once specified a tournament scenario, there are still two fundamental issues to be considered. On the one hand, there is the matter of explicitly stating how a tournament specification is encoded and conveyed to trading agents so that these can conveniently parse it. On the other hand, how institutional agents, and interagents, have their behaviours altered to operate in the several tournament modes.

Considering the first issue, upon reception of the tournament descriptor (in the shape of a serialized Java object), interagents transform it into a string-based

[3]We name it *fishmarket* for historical reasons, though the term must not be misleading since under this mode goods can be auctioned through several auction protocols.

message analogously to the messages in Tables 5.5 and 5.6 (see Section 5.4.3) that is subsequently conveyed to trading agents. The actual syntax of the message corresponds to message #32 in Table 6.3. In it the *tournament_descriptor* predicate is accompanied by a long list of parameters encoding the tournament definition.

Observe that neither evaluation functions nor supply functions are part of the tournament descriptor conveyed to trading agents. Instead of encoding them and posting them to trading agents, the evaluation of both types of functions is offered as a service by interagents (see messages 2 and 3 in Table 6.3) that can be requested by trading agents at any time. In this way, a buyer can request his interagent for the expected composition, if available, of the i-th lot to be auctioned or for the score of his rivals. However, notice that supply and evaluation functions must be made public prior to the tournament so that agent developers eventually consider them for designing their trading strategies.

#	Predicate	Parameters
32	*tournament_descriptor*	$auction\ n\ \Delta_{auctions}\ \Delta_{rounds}\ \epsilon\ bidding\_protocols\ dbp\ \Delta_{price}\ \Delta_{offers}\ \Sigma_{coll}\ \Pi_{sanction}\ \Pi_{rebid}\ UBP\ \Delta_{price}\ \Delta_{offers}\ \Pi_{sanction}\ \Pi_{start}\ FPSB\ B_t\ \Pi_{sanction}\ vickrey\ B_t\ \Pi_{sanction}\ (buyers\ \{buyerlogin^* \mid \#buyers\}\ credit\ \{credit^* \mid credit \mid unkown\}\ sellers\ \{sellerlogin^* \mid Market\}\ mode\ \{automatic, uniform\ one\_auction\_data, one\_auction\_sellers, fishmarket\}$
33	*get_goods*	$auction\_number$
34	*eval*	$\{buyerlogin \mid all\}$
35	*score*	$buyerlogin\ score$
36	*ranking*	$\{buyerlogin\ score\}^*$

Table 6.3: Messages handled by interagents to encode the tournament descriptor and manage the evaluation of supply and evaluation functions.

Regarding institutional agents, some extensions are needed in order to enable them to act not only in market scenarios but also in tournament scenarios. But changes are not so significant. They basically reduce to slight changes in their roles' specifications. Thus, while the *fishmarket* mode does not require institutional agents to alter the behaviour that they exhibit in FM96.5, other tournament modes such as *automatic*, *one auction (data set)* and *uniform* require that the sellers' admitter and the sellers' accountant do not offer any access or services to sellers since these are not allowed to participate. Some changes in the institutional agents' behaviour are also required. For instance, in the *RFG*(request for goods) scene, the sellers' admitter itself composes a lot of goods to be conveyed to the auctioneer.

Observe that FM96.5 becomes one of the operation modes offered by FM. In fact, FM has been evolved from FM96.5 respecting its architecture but benefitting from its flexibility and extendibility to enlarge its functionality. This is the reason why we claim FM to be an agent-mediated test-bed.

n	21				
$\Delta_{auctions}$	*5000 msec*				
Δ_{rounds}	*3000 msec*				
$\mathcal{D}$	$\{d_{DBP}\} = \{\langle 10ptas, 1000msec, 3, 0.25, 0.25\rangle\}$				
PS_B	$(\{tournament\_admission, Dutch\_auction, output\},$ $\{(tournament\_admission, Dutch\_auction), (Dutch\_auction, output)\})$				
PS_S	$\emptyset$				
$\mathcal{B}$	*warakaman akira fishbroker tindalos dolphin f2422080 panipeixos josnat satan xanquete e0934125*				
$\mathcal{S}$	$\varnothing$				
$\mathcal{F}^i(i = 1 \ldots n)$	τ	*#Boxes*	p_α	p_{rsl}	p_{rsv}
	cod	$U[1, 15]$	$U[1200, 2000]$	$U[1500, 3000]$	$U[0.4, 0.5]$
	tuna fish	$U[1, 15]$	$U[800, 1500]$	$U[1200, 2500]$	$U[0.3, 0.45]$
	prawns	$U[1, 15]$	$U[4000, 5000]$	$U[4500, 9000]$	$U[0.35, 0.45]$
	halibut	$U[1, 15]$	$U[1000, 2000]$	$U[1500, 3500]$	$U[0.4, 0.6]$
	haddock	$U[1, 15]$	$U[2000, 3000]$	$U[2200, 4000]$	$U[0.35, 0.55]$
C	$C(b) = 50000ptas.\ \forall b \in \mathcal{B}$				
M	$< 1, 0, 0, 1 >$				
ϵ	0				
E	$\langle E_b, E_s \rangle = \langle \sum_{a=1}^{n} \beta_a^r \left[\frac{4}{r^2}(\pi(r - \pi))\right], \varnothing\rangle$				

Table 6.4: UPC'97 Tournament Descriptor

6.4 The Fish Market Tournaments

In this section we illustrate the type of tournament scenarios that can be generated with the aid of FM by reporting on the experiences gained with the *Fishmarket* tournaments. These tournaments took off in 1997 at the Technical University of Catalonia (UPC). Thereafter they have been regularly (yearly) organised at the UPC till a final, international tournament held in the framework of the *Fourth International Conference on Autonomous Agents* [Béjar and Rodríguez-Aguilar, 2001]. The purpose of these tournaments is to spur research on both trading agents' architectures and trading strategies.

In what follows we comment on the type of scenarios and outcoming results of three tournaments samples[4].

6.4.1 UPC'97

The first *Fishmarket* tournament took place at the Technical University of Catalonia in 1997 and involved a group of undergraduate students. For this very first tournament, we opted for a rather simple scenario fully featured by the tournament descriptor in Table 6.4.

We opted for a simple scenario characterised by the tournament descriptor in Table 6.4. There are some comments to be made on the resulting scenario:

[4]For more detailed information, we address the reader to the tournament web page where all tournament experiences held so far are thoroughly reported [Fishmarket, www].

- All buyer agents were assigned the same credit (50000 ptas.) at the beginning of each auction of the tournament.
- Because the tournament mode was set to *uniform*, the number of fish boxes for each type of fish (τ) were randomly generated for each auction $\mathcal{A}^i$, and the starting price (p_α), resale price (p_{rsl}), and reserve price (p_{rsv}) of each box were also randomly generated according to the uniform distributions in Table 6.2. All distributions except those referring to the reserve prices were known by contenders.
- Agent templates for buyer agents were provided in Java, C, and Common Lisp.
- The chosen evaluation function (E_b) calculates the performance for each buyer at round number r of auction number a based on the accumulated benefits (β_a^r), the accumulated number of purchases (π), and the number of rounds (r) where that buyer is active.

In these tournaments we tried to keep things as simple as possible concerning the performative structure specification of buyers. The performative structure can be regarded as a simplified, slightly modified version of the performative structure in Figure 5.1 that can be also represented as a simple graph of protocols such as depicted in Figure 6.5. In fact, a single FSM specification was constructed by connecting the scenes in the graph to produce the specification in Figure 6.6 accounting for the specification of the protocol that buyers must use for interacting with their interagents.

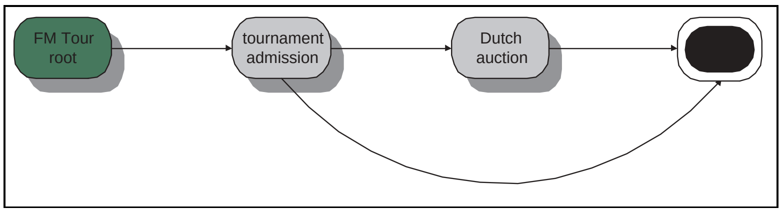


Figure 6.5: Performative structure for buyer agents in the *Fishmarket* tournaments.

Notice that the simplicity of this definition does not prevent us from defining more complex tournaments in which a larger performative structure was employed containing other scenes (f.i. negotiation, settlements, etc.).

In spite of the simplicity of the rudimentary agents taking part in this tournament, some considerations are worth reporting:

- The experimental conditions defined (mainly starting prices, available endowments, and evaluation functions) favoured voracious strategies (buy as much as possible, as soon as possible).

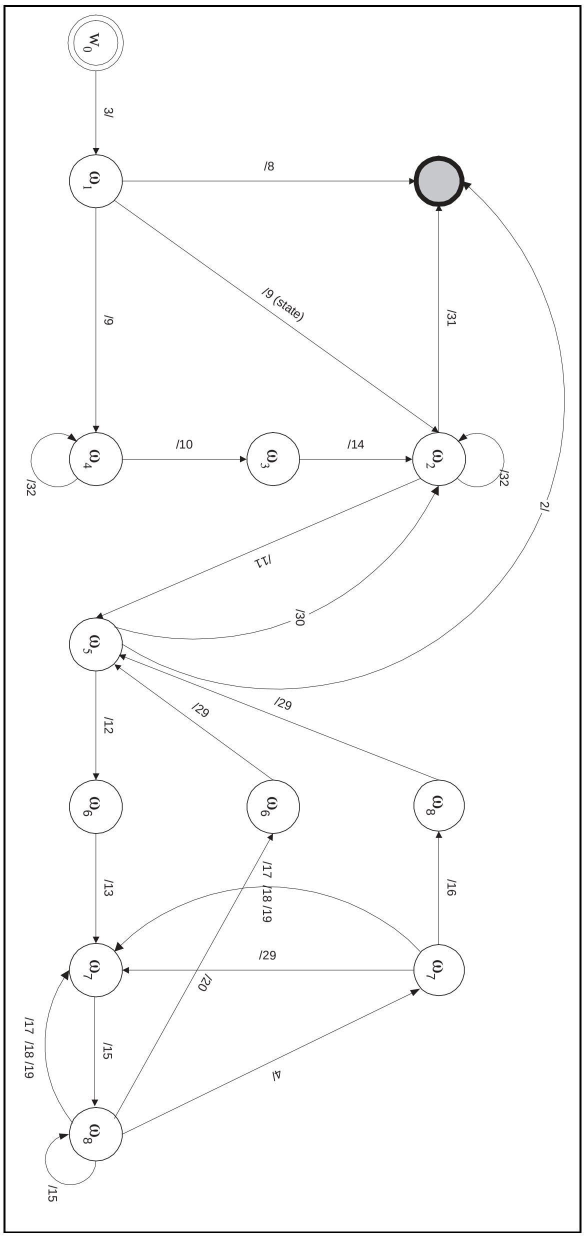


Figure 6.6: Communication protocol used by buyer agents to interact with interagents.

- The setting of time-delays (like Δ_{offers}, Δ_{rounds} and $\Delta_{auctions}$) acted against deliberative agents.
- Most of the contenders calculated their bids by multiplying the starting price by a bidding rate that was dynamically tuned depending on the success or failure of the agent's former bids.
- Surprisingly, those agents using their opponents' models assumed that all the contenders acted according to the strategy described above.
- Curiously, the winning agent did not need as much information as the others to succeed in this tournament. His extremely simple, purely reactive strategy consisted in bidding when the ratio between the resale price and the offer price was considered to be high enough, and this sufficed to beat the rest of agents.

De Toro (in [de Toro, 1997]) devised variants to these tournament conditions and showed that deliberative agent performance, relative to simple reactive heuristics, improved with scarcity of resources and experience, as long as time delays between rounds and between auctions were kept above a threshold[5].

6.4.2 EPFL'99

The third *Fishmarket* tournament took place at the École Polytechnique Fédérale de Lausanne (EPFL) and involved this time a group of graduate students.

Some changes apply to the tournament descriptor in Table 6.4. Firstly, the credit endowed to each buyer was reduced to 10000 ptas. In this way, buyers had to face different situations at each auction depending on the size of the lot: from scarce auctions (few products, much market money) to auctions with excess of supply. Secondly, the evaluation function is defined for each buyer b as $E(b) = \sum_{i=1}^{n} ln(i+1)B_i(b)$ where $B_i(b)$ stands for the accumulated net benefit of buyer b during the i-th auction. This evaluation function was intended to promote those buyers that improved their performance over time adapting better than others.

This competition turned out to be highly levelled and very interesting with respect to the variety of strategies developed by the contenders. Next we comment some relevant features concerning the most successful agents:

- No agent did model his rivals.
- The benefit of goods was employed for elaborating strategies in different manners. Thus, while agents *did* and *Bond* based their strategies on the expected benefits of goods, buyer *baidy* considered the average benefit obtained by successful bidders in past rounds and buyer *LostAgent* defined values of desired benefits (difference between each resale price and his own bid) for each type of good. *LostAgent* tunes these values depending

[5]Using a more standard relative-performance common-value evaluation function.

on success and failure. When succeeding, *LostAgent* lowers his degree of aggressiveness, whereas he increases it when losing.

- The most winning agent showed the most sophisticated approaches, characterised by attempting at pursuing adaptability.
- Agents employing more conservative strategies, such as *did*, obtained very high scores in auctions with high supply (lengthy auctions), while agents employing more aggressive strategies, such as *Bond*, did very well in scarce auctions (short auctions).
  - The winner, *aai4*, employed a rule base making use of the category of goods —goods were classified according to their expected resale price—, his own market opportunity, and the market opportunity of his rivals (considered as a whole). The result of applying these rules for each bidding round was a collection of factors used for generating the buyer's bid. These factors were tuned prior to the tournament by means of off-line training.
  - Buyer *yves* presented an interesting approach based on the use of case-based reasoning for determining his bid. His default strategy consisted in being very conservative. Then, it employed the information generated during past auctions to obtain the most similar cases to the current bidding round. He departed from the assumption that similar situations lead to similar results. Thus the similar cases were used for calculating the average benefit obtained by the winners. From this average benefit, *yves* tuned his default bid.

    The major drawbacks of this approach have to do with the length of the tournament and the cases selected. Tournaments were composed of at most twenty auctions, and so eventually could not count on past cases similar enough to the current bidding round. On the other hand, some cases may not be worth considering since they correspond to small benefits. And yet, *yves* took them also into account.

6.5 AMEC'2000

At the time of writing, the last tournament was held in Barcelona in the framework of the *Fourth International Conference on Autonomous Agents (Agents'2000)*.

Some changes apply to the tournament descriptor in Table 6.4 summarised by the tournament descriptor in Tables 6.5. A total of sixteen agents participated in the tournament. All the students had time to study the environment, and to experiment with toy agents provided by the platform and agents from previous tournaments. Nevertheless the source code of agents participating in former tournaments was not available.

Each agent development team was asked —before running the tournament— for identifying what they considered as relevant information to design agents' bidding strategies. They agreed on the relevance of the following information:

n	10				
$\Delta_{auctions}$	*8000 msec*				
Δ_{rounds}	*4000 msec*				
$\mathcal{D}$	$\{d_{DBP}\} = \{\langle 50.0EUR, 500msec, 3, 0.25, 0.25\rangle\}$				
P_B	P_{DBP}				
P_S	$\emptyset$				
$\mathcal{B}$	$\{b_1, b_2, b_3, b_4\}$				
$\mathcal{S}$	$\varnothing$				
$\mathcal{F}^i(i = 1 \ldots n)$	τ	$\#Boxes$	p_α	p_{rsl}	p_{rsv}
	cod	$U[1, 15]$	$U[1200, 2000]$	$U[1500, 3000]$	$U[0.4, 0.5]$
	tuna fish	$U[1, 15]$	$U[800, 1500]$	$U[1200, 2500]$	$U[0.3, 0.45]$
	prawns	$U[1, 15]$	$U[4000, 5000]$	$U[4500, 9000]$	$U[0.35, 0.45]$
	halibut	$U[1, 15]$	$U[1000, 2000]$	$U[1500, 3500]$	$U[0.4, 0.6]$
	haddock	$U[1, 15]$	$U[2000, 3000]$	$U[2200, 4000]$	$U[0.35, 0.55]$
C	$C(b) = 17.500EUR\ \forall b \in \mathcal{B}$				
M	$< 1, 0, 0, 1 >$				
ϵ	0				
E	$\langle E_b, E_s \rangle = \langle \sum_{k=1}^{n} ln(k+1) B_k(b), \varnothing \rangle$				

Table 6.5: AMEC'2000 Tournament Descriptor

From the market: Number of rounds, number of boxes, initial credit, remaining credit (after each bidding round), last benefit obtained.

From the goods: Starting price, resale price, reserve price (if the good is withdrawn), good type, ratio between sale price and resale price.

From the competitors: Mean benefit, accumulated benefit of the leading agent, remaining credit, behaviour.

From the agent state: Own benefit, remaining credit, number of boxes bought.

Because of the time restrictions, not all the information above was actually employed during the tournament. Each group reduced all the information above to a subset containing the most relevant infromation. Surprisingly there was a great consensus among most agent developers about what information had to be considered. First, the length of the auction. Almost all agent developers considered a length-based classification of auctions. Thus, the number of classes ranged from two to four, being three the most frequent value. Typically, classifications distinguised short-sized auctions (approximately 20 boxes), medium-sized auctions (approximately 45 boxes) and long-sized auctions (up to 75 boxes). Then each kind of auction leads to a different strategy:

- In short auctions aggressive strateges were mostly used, attempting at buying close to the starting price. If the credit is enough, this is an admissible strategy because the total market money is higher than the cost of all the goods. There is no time to consider the characteristics of the goods, because probably not all the money could be spent. The better good is the one with a better ratio between starting price and resale price.

- In medium auctions a more deliberative strategy is necessary. The total money of the agents is almost enough to buy all the goods, so the agents had to be selective and compete for the best goods. The last goods of the auction can be interesting because their price can be lower.
- In long auctions the planning is very important. The cost of the goods are more than the total market money. The agent has to decide what goods are interesting because its price and its position in the auction. It could be an interesting strategy to wait until all the competitors had spent all their money in order to obtain better prices. In this kind of auctions an accurate estimation of the reserve price is very important.

The other information from the auction that had almost all the consensus was the quotient between the total resale value of the goods of the auction and the total market money. This value can be uses as an estimation of the mean expected benefit. To outperform or underperform this value is an indicator of the performance of the agent. This measure is correlated to the behavior of the auction and allow to not to observe individually to each competitor.

This expected benefit can be updated during the auction by the bought of the agents. This allow to change the behavior of the agent because the raise or fall of the expected benefit.

Almost all the agents used this ratio as base value in order to decide its bid. If the initial benefit of the good is lower than the mean benefit, then the good is not interesting and, either the bid is not done, or the agent wait until the price drops to a more interesting one.

The agents used other complementary values to correct the bidding price obtained from the calculation of the mean benefit. For example, the benefit of the best agent, the remaining credit of the competitors and heuristical factors obtained by experimentation during the private auctions that were held before the official tournament.

Due to that in long auctions to wait until almost the end of the auction is a profitable policy, the estimation of the reserve price becomes important. Every agent has a way to estimate the reserve price. Some agents do the estimation dynamically, trying to learn this price from the auction, others used a constant percentage from the initial price. Obviously, the agents that try to estimate the reserve price dynamically obtained better results.

The strategies to determine the reserve price were diverse, but based on statistical estimation. Because the real reserve price is unknown, the difference between starting price and the lower price payed for the goods is a good initial estimation. This estimation can be corrected using the price observed when a good goes out of market, circumstance that can be observed in long auctions. Some agents tried to estimate the reserve price for each kind of good. Due to the relative shortness of observations those estimations were less accurate that those from the agents that tried to estimate a global reserve price.

Planning and learning were rare among the agents of the tournament. Some agents tried to plan beforehand the goods more attractive, estimating the optimal bid and distributing the available money among them. All allowed a dynamical redistribution of the bids if the chosen goods were bought by another competitor.

Just two agents tried to use learning between auctions to improve their performance. The first, used the comparison between the benefit obtained and the benefit of its competitors in order to reduce or increase the bidding price in the next auction. the second used a more sophisticated learning mechanism based on reinforcement learning. This strategy used Q-learning in order to decide the optimal benefit for each good from the own actions and the actions of its competitors.

The competition was organized in three eliminatory rounds. The first round divided the agents randomly in four groups. Each group competed in a tournament. From each group only the two best were chosen.

In this round the agents with a weak strategy obtained a significant less performance than the more elaborated agents. This year, in contrast with previous tournaments, the level of cooperation between the groups were very low. Only a small number of agents participated on private tournaments. Most of the agents that were eliminated in this round were the non cooperative ones. This gives an idea of how important is cooperation and interaction during the developement of agents.

The second round paired the winning agents of the first and second group and the agents of the third and fourth group. In this round also the two best of each group passed to the final round.

In this round the competition was hardest. In the first group the difference among the three firsts agents were very short. In the second group there was a clear difference between the first two agents and the other two competitors.

The strategies of the winners of this round were not significantly different from the rest, but included some the agents that used some kind of learning and adaptation.

Surprisingly, the winner of the final round was the agent with the simplest strategy of the four competing agents. Those are the four agents of the final round and their strategies:

HumbleJES: This is the winner agent. The basis of this agent is the ratio between the resale price of the remaining goods and the total credit of the agents. This ratio is weighted using a value that is an estimation of the desired benefit. This expected benefit is a constant that is not changed during the competition.

This value is used to estimate the bid for the actual good. This price is corrected with the information about the money available for the other agents. If this value is greater than the price that can be paid by their competitors, it is adjusted to a little more than this quantity. If the competitors can not buy the good, then the price is adjusted to the estimated reserve price.

garsa: This is the second agent. The basis of this agent is also the expected benefit obtained as a ratio of the resale price of the goods and the money available, but in this case, this ratio is calculated at the beginning of each auction. This value is modified using the behavior of the other agents. If the rest of agents bid to a price higher that the calculated, the value is not touched. If the other agents bid to a lower price, the benefit is adjusted to obtain a bid slightly higher that the bid of the competitors, increasing the own benefit.

This agent detects when the competitors have not enough money to buy more goods. When this happens, the bid is adjusted to a statistically estimated reserve price.

The Pretender: This is the third agent. This is the more sophisticated agent, it uses reinforcement learning (Q-learning)in order to learn what is the better price for a good. It uses a probability matrix indexed by resale price and expected benefit. This matrix stores the probability distribution of the optimal benefit for a given resale price. The matrix was initialized with a priori probability distributions obtained from the private tournaments.

Three different reinforcements are used during the auction. A positive reinforcement if the current bid is successful and is considered a good bid, a negative reinforcement if it is considered that the actual bid benefit has to change and a negative reinforcement if the actual did benefit of the agent is high. A set of rules allow to decide what kind of reinforcement is necessary. These rules evaluate different information, as the number of remaining rounds, the performance of the competitors or the number of competitors with enough money. The learning is done in each auctions, so the information of the previous auctions is not maintained.

This probability matrix adapts to the behavior of the market, and predicts the most probable benefit that the competitors desire to obtain. This information allow to advance the bid and to buy before than the competitors.

TokOchons: This is the fourth agent. The strategy of this agent uses two information. The first is a variation of the ratio between the resale price of the remaining goods and the remaining market money. This information allow to guess the expected benefit. The second source of information is a function that give a measure of how interesting is a good. This function combines the relative and absolute benefit obtained for a given bid.

This bid is corrected using different parameters. The more interesting is a value that measures the proportion of the market money that the agent owns. If the proportion is great, this means that the agent almost has not competitors, so, the expected benefit can be raised.

This agent stores the past auctions in order to analyze them. If the current auction has a similar number of rounds that a past auction, its information is recalled. If in this past auction some money was not spent, the bids are raised in order to spent all the money, increasing the benefit by buying more goods. If the winner of this past auction obtained a benefit higher than ours, the expected benefit for the current auction is raised in order to pay less for the goods.

Figure 6.7 depicts the evolution of the objective function that measured the performance of agents. We see that agent **HumbleJES** performs significantly better that the others from the very start of the competition. The rest of agents are in a tie until auction number seven, when agent **garsa** starts outperforming the other two agents. It seems that the learning procedures of this two agents are not a real advantage against the other two strategies.

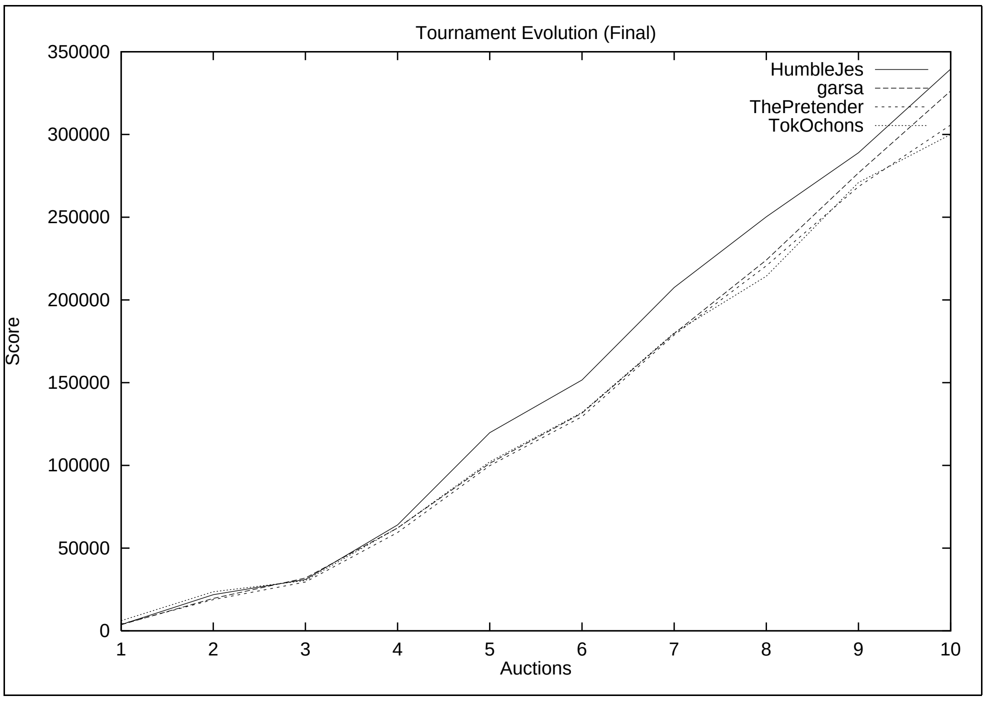


Figure 6.7: Evolution of the last round of the AMEC tournament.

Some conclusions can be drawn from this tournament. First of all, that more sophisticated strategies has not evident advantage against simple ones. The best agents use an strategy based on market information without neither trying to model the other agents not use learning from experience to improve their performance. This does not means that this characteristics are not desirable. An adequate learning policy could overperform simple strategies in a more dynamic environment.

The other conclusion is the significance of competition in the developement of this kind of agents. At has been said, only the agents from the people that decided to share their knowledge and competed in private tournaments were successful. The need to test a strategy are crucial for its developement. It is difficult to have success without interaction.

6.6 Summary

Research in e-commerce is producing an increasing number of agent-based markets (refer to Chapter 2 for a complete survey on computational markets). Such efforts have succeeded in producing electronic markets where both buying and selling agents can trade on behalf of their users. Nonetheless there is the intricate matter of providing agent developers (and agent users) with some support to help them face the arduous task of designing, building, and tuning their trading agents, before letting them loose in wildly competitive scenarios. In this sense we regard as paradoxical the lack of test-beds for trading agents. We have attempted to contribute in that direction. We have developed a test-bed, FM, that can be used to test and tune trading agents, as an extension of an actual agent-mediated electronic auction house.

The resulting test-bed is domain-specific and realistic because it has been grown out of FM96.5, the actual-world computational market presented in Chapter 5.

FM allows for the generation, recording and repeatability of competitive auction-based trading scenarios of varying degrees of complexity. Such scenarios may correspond to either open or closed tournaments, depending on whether they are accessible to trading agents owned by multiple users or by a single user respectively. Experiments defined and subsequently hosted by FM can be monitored with the aid of the extended version of the monitoring agent inherited from FM96.5, and their performance can be obtained thanks to the built-in evaluation facilities.

The participation of agents is mediated by the interagents inherited from FM96.5. Notice that the use of interagents makes FM architecturally neutral. However, in order to provide some support to agent developers, FM includes a library of agent templates. In addition to this, the capability of generating dummy agents is also included for training purposes. Lastly, FM has been constructed to be scalability aware, in the sense that it provides support for distributing an experimenter's agents across several machines in a network.

Our proposal is close to the *Double auction* tournaments held by the Santa Fe Institute [Andrews and Prager, 1994] where the contenders competed for developing optimised trading strategies. Though similar enough, our approach has a wider scope. We are interested not only in providing a framework for testing agent strategies and building trading agents [Vidal and Durfee, 1996, Garcia et al., 1998b, Boutilier et al., 1999], or in the use of artificial intelligence to study economic markets [Rajan and Slagle, 1996]. We are also interested in the study of market conditions and market conventions, thus our emphasis on the flexibility of the specification framework, and the generality of the underlying definitions. Thus FM shares many commonalities with AuctionBot [Wurman et al., 1998], a highly versatile online auction server that permits the generation of a wide range of auction environments wherein both human and software agents can participate. AuctionBot has already proven its usefulness as a research platform hosting large-scale experiments to study computational market mechanisms and agent strategies.

Finally, we would like to comment on what we regard as an interesting problem that could be investigated with the aid of the test-bed presented in this chapter. Observe that FM permits the creation of several auction-based market scenarios, each one possibly exhibiting a different behaviour. Thus we could think of having agents simultaneously participating in some markets so that should not only care about their trading strategies in a single market, but also about choosing the most appropriate market. This is in fact the actual problem that agents must face over the Internet when intending to acquire goods at an on-line auctions.

Chapter 7

Conclusions and Future Work

This thesis addressed the formalisation, design and construction of agent-mediated electronic institutions. The proposed methodology is exemplified through the practical realisation of an agent-mediated electronic auction house that is subsequently evolved into a test-bed, an experimental environment, for trading agents.

In what follows we present the lessons learned from the several problems faced during this exercise.

7.1 Formal Specification of Electronic Institutions

Although up to date most of the work produced by MAS research has focused on *closed* systems —systems in which agents and their infrastructure are designed, developed and enacted under centralised control [Klein, 2000]—, *open systems* [Hewitt, 1986] —systems whose components are unknown beforehand, can change over time and can be both human and software agents developed by different parties— have recently started to be considered by agent researchers as arguably the most important application of multi-agent systems. Typical examples of open systems include electronic marketplaces, virtual supply chains or collaborative design applications to name a few.

It is our view that open systems require the deployment of a normative environment which encapsulates the rules of the game, including any (formal or informal) form of constraint aimed at shaping agents' interactions. In other words, a framework within which interactions take place, defining what individuals are forbidden and permitted and under what conditions. We argue that a clear distinction must be drawn between the rules and the players of open systems.

We have proven that much can be learned in the exercise of producing such normative environments by adopting a mimetic strategy based on the formal introduction of organisational structures. More concretely, institutions —which have for long demonstrated their effectiveness in dealing with similar issues in human societies— are also effective for the successful realisation of open systems. Notice that the ultimate goal of engineering electronic institutions —the counterparts of institutions in electronic environments— requires the precise, formal, rigorous identification of all their components along with their relationships and behaviours. Since mostly open systems can be regarded as critical applications, the use of a formal technique is more than motivated because we cannot afford to have the requirements misunderstood. Along this exercise we have observed that it is fundamental to adopt a *macro*, societal perspective that helps identify the processes in the system and their relationships, the roles of the participating agents and their associated rights, and the obligations and restrictions deriving from agents' observable, external actions.

Several major advantages derive from producing (graphical) formal specifications of electronic institutions:

- The process of creating the description and performing the analysis allows the modeller to gain a dramatically improved understanding of the modelled institution.
- Graphical specifications are extremely easy to understand since they are similar to the informal diagrams employed by engineers and designers while designing, constructing and analysing a system.
- A specification offers an explicit description of both states and actions which any subsequent implementation is aimed at capturing.
- Formal reasoning can be applied in order to analyse and validate the properties of the specification.
- The knowledge acquired after the specification of scenes and institutions naturally leads to the creation of *scene patterns* and *institutional patterns* for future reuse.

Although it's apparent the urgent need for methodologies and techniques that underpin the development of agent systems, in general these are still at a very embryonic stage. Along these lines, we understand our contribution as a modest, early approach in the field, and not as a breakthrough. In general, we argue that in order for any methodology to be successfully deployed, there is the fundamental issue of counting on a well-defined semantics which allows the formal reasoning which founds validation techniques for complex and critical applications. And yet there is still a long way to go. As an example of the embryonic stage of the field, consider the recent extensions of UML which have recently appeared at the time of writing [Odell et al., 2000]. Although interesting enough since they build upon UML, which has succeeded as a de-facto design tool in industry, UML still lacks of precise semantics. Therefore, UML does not support so well the formality and rigour needed to early detect errors in requirements and design. It is not surprising then that this is widely admitted as one of the major challenges for UML in order to enhance its applicability to the modelling of complex systems [uml2000, www].

7.2 Agent-based Architectures for Electronic Institutions

We have also presented a computational model that arises from and fully captures our formal view of electronic institution in an effort to link theory and practice. For this purpose, we build upon the key notion of *mediation*. We argue that mediation is capital to found the realisation of infrastructures for agent-based systems in general and for electronic institutions in particular. Thus, we defend that infrastructures for agent-based systems in general, and for electronic institutions in particular, can be successfully deployed making use of middle-agents, mediators, that take charge of the cumbersome interaction issues inherent to these types of systems. This observation motivated our introduction of a special type of facilitator, the so-called *interagent*, an autonomous software agent that mediates the interaction between each agent and the agent society wherein this is situated. Thus, interagents constitute the unique mean through which agents interact within a multi-agent scenario. Whereas an agent's external behaviour is managed by an interagent, his individual logics, knowledge, reasoning, learning and other internal capabilities must be exploited and managed by the agent himself. Recall that not only an interagent is capable of handling the interactions of a given scene protocol, but the whole navigation of an agent across all the activities involved in an institution. Furthermore, an interagent must keep track of the commitments and prohibitions applying to a given agent in order to determine how these enlarge or restrict his actions.

But not only interactions within an electronic institution are mediated by agents, but also its services, delegated to the so-called *institutional agents*. We refer to institutions whose interactions and services are mediated by agents as *agent-mediated electronic institutions*. We have proposed a novel computational model to fully realise electronic institutions by means of the articulation of institutional agents and interagents. Thus, the (institutional) normative environment is accomplished by means of the coordinated, cooperative activity of institutional agents and interagents. Or, in other words, an agent-mediated electronic institution takes the shape of a cooperative agent society: institutional agents and interagents work together to enforce the institutional rules and to offer institutional services.

Next we summarise the main features and benefits of our computational model for agent-mediated electronic institutions:

- *Purely agent-oriented design* founded on institutional agents and interagents.
- *Flexible agent architecture.* All institutional agents share the very same, highly flexible architecture. Nevertheless, they exhibit different behaviours

in accordance with the responsibilities they're endowed with expressed as specifications. Likewise, interagents do all share the same architecture, deferring solely on the scenes being mediated for their customer agents. A major benefit arises from employing such generic, flexible architectures: savings in development effort when carrying out institutional changes.

- *Macro(societal) and micro(agent-centered) views.* As a follow-up of the item above, notice that not only our computational model is concerned with societal issues (scenes, performative structures, roles, etc.) but also with the inners of the agents that make possible the institutional environment.
- *Enforcement of institutional rules.* The mediation of interagents guarantees that agents abide by the institutional roles when participating in activities (scenes) and when transiting from activity to activity.
- *Openness.* Any agent (no matter his architecture nor his language) can participate in an institution as long as he can communicate with an interagent.
- *Scalability.* The services offered by an institution can be distributed among several institutional agents or simply delegated to the very same agent. Furthermore, interactions with external agents can be either handled by a single agent or by multiple agents (e.g. one interagent per agent).
- *Modularity.* The notions of agents and their roles and performative structures and their scenes allow us to introduce a much higher degree of modularity that what we would obtain by plainly using objects.
- *Security* issues concerning external agents can be effectively handled via their interagents. Notice that external agents are not allowed to directly interact with either institutional agents or other agents within the institution. Instead, an agent must be granted access to the institution through his assigned interagent in order to have his illocutions conveyed to their addressees. In this way we reduce the likelihood of an agent jeopardising the institution.
- *Robustness.* Malicious behaviour and failures of external agents can be effectively controlled by coordinating interagents and institutional agents. For instance, bid spamming in FM96.5 is easily controlled by having interagents plugging off spammers.
- *Accountability support.* Since all illocutions uttered and received by agents within an electronic institution are filtered by interagents, they can all be kept for accountability purposes.

We have demonstrated the pragmatic value of our computational model developing an actual agent-mediated electronic auction house. Apart from exhibiting the features and benefits listed above, to the best of our knowledge FM96.5

(along with AuctionBot) was the first auction house that permitted the participation of software trading agents in market sessions. Furthermore, it includes a fair, live implementation of the challenging Dutch auction protocol.

7.3 Agent Test-beds

North [North, 1990] argues that a distinguishing feature of institutions is the clear distinction drawn between the rules and the players. Most of this thesis has focused on the rules through the formal specification, design and realisation of institutions' normative environments. However, departing from the agent-mediated electronic auction house presented in Chapter 5 to illustrate the pragmatic value of our computational model, we have shifted our interest from the rules to the players to produce an original, novel test-bed for trading agents in auction markets. In this way we have contributed to providing agent developers with support to help them face the intricate matter of designing, building, and tuning their trading agents, before letting them loose in auction marketplaces. It is our view that such experimental support is fundamental in the construction of trading agents, since these must be regarded as highly critical applications.

Notice that the lack of agent test-beds is mainly due to the costly development effort required by this type of systems. But this is not the case with our testbed. Evolving the agent-mediated electronic auction house presented in Chapter 5 to the testbed presented in Chapter 6 has not been expensive because of the high flexibility of our computational model. Thus realising different experimental scenarios amounts to specifying and loading different behavioural (social and internal) specifications into institutional agents. Needless to say that this favours the future specification of new experimental scenarios.

But more importantly, the exercise of constructing an auction-based testbed has led us to infer the general, desirable features of an open agent test-bed:

- *Realistic.* Although at the expense of being domain-specific, it is desirable that testbeds support empirical work in realistic scenarios. It is our view that testbeds must be intended to help researchers progress in their solutions to real-world, instead of artificial, problems.

- *Generation and repeatability of scenarios with varying degrees of complexity.* From simple toy scenarios to complex real-world scenarios, from carefully constructed scenarios that highlight certain problems to randomly generated scenarios useful for testing agents' average performance.

- *Architecturally neutral.* The participation of agents possibly written in different languages and possibly with different architectures should be allowed.

- *Scalability aware.* The possibility of distributing agents across several machines has demonstrated to be necessary to tackle scalability problems. Executions in multiple processors eventually guarantee the appropriate performance of experiments which otherwise would collapse.

- *Open and closed experimental scenarios.* Open scenarios are intended to allow the participation of agents owned by multiple users (multi-user), whereas closed scenarios are intended for a single user's controlled experimentation.

- *Monitoring and Evaluation facilities.* Testbeds must keep track of all events taking place during an experiment, so that a whole empirical session can be audited step-by-step, and the evolving performance of all the agents involved can be traced, calculated, and analysed.

- *Agent development support.* Since the major concern of agent developers must be their agents' logics in the testbed's domain, it is desirable to provide them with agent templates to ease and speed their development. Needless to say that such templates are eventually committed to some particular architecture, but still they can save agent developers' time preventing them to depart from scratch.

- *Generation of customizable training agents.* Facilities for the generation of customisable agents for training purposes.

Analogously to our exercise of proving the pragmatic value of our computational model for electronic institutions, we have also illustrated the value of the testbed developed in Chapter 6 to host several auction contests, along the lines of the *Double auction* tournaments held by the Santa Fe Institute [Andrews and Prager, 1994] where the contenders competed for developing optimised trading strategies.

7.4 Future Work

Nowadays, there is an evident lack of methodologies for designing, implementing, analysing and ultimately validating agent-based systems in general, and electronic institutions in particular. One of the key reasons roots in the inadequacy of existing software development approaches such as early pointed out in [Fisher et al., 1997]. Nonetheless it is widely agreed by the agent community that there is an urgent need of methodological foundations for agent-based systems. This fact has motivated and spurred the very recent appearance of some purely agent-based software engineering approaches such as [Wooldridge et al., 2000] and [Odell et al., 2000]. Nonetheless, agent-oriented software engineering is still in its infancy, and therefore a remarkable amount of research is bound to be produced in this area in the very near future. These research efforts must not only include and handle the notion of agent as first class citizen, but also societal and organisational patterns for which agents become players with various roles.

Although extensions of object-oriented software engineering techniques, such as UML, may prove successful and eventually become daily tools for multi-agent systems' engineers, we still believe that formal techniques will have a say. Needless to say that formal techniques coming from the domain of process algebra, structural operational semantics, temporal and modal logics or Petri nets have been largely employed for the specification, design and verification of distributed and concurrent systems. For instance, coloured Petri nets have demonstrated to be useful in the specification and validation of network protocols [Jen, 1995b].

The work reported in Chapter 3 has attempted at making headway in this matter, and yet there remain major, open issues that must be solved by future work in order to enable formal reasoning: the formal analysis of specifications. Formal analysis techniques are needed in order to study static (structural) and dynamic (behavioural) properties of specifications. On the one hand, static properties can be derived from an electronic institution's specification, without taking into account its dynamics. The main purpose of static properties will be to characterise and classify specifications into subclasses. On the other hand, dynamic properties can be studied from two different perspectives. First, simulation appears as the most straightforward kind of analysis. However, in order to implement simulators, it is necessary to count on a well-defined semantics which unambiguously defines the behaviour of electronic institutions from specifications. Though simulation can prove to be extremely useful for the understanding and debugging of an electronic institution —in particular, when designing and validating large systems—, it is important to develop formal analysis methods. We regard these methods as an indispensable complement to simulation techniques.

Once specified an electronic institution a sound implementation must follow. It is our view that agent engineers are to need guidance through both the intricate specification and development phases. Thus, we envision the construction of development environments as a fundamental future task if we aim at obtaining assisted principled developments. Ideally development environments must succeed in providing institutions' builders with the necessary tools to cope with the high complexity inherent to the types of systems that they aim at constructing. These tools must not only support the construction of electronic institutions, but also, and very importantly, the analysis and debugging of the system so that designers can be assisted in diagnosing faulty systems both at design and run time.

Notice that in Chapter 3 we pictured a component-based formal conception of electronic institutions. Scenes can be regarded as independent components which model different types of multi-agent conversation. Peformative structures can be also regarded as independent components that are finally integrated into an electronic institution. And lastly, the specification of an electronic institution can also be seen as a component itself. From this follows that specifications of scenes, performative structures and electronic institutions might be organised into repositories, ready to be used as building block for new specification. If so, in practice, designers could easily compose a new specification of an electronic institution by eventually retrieving and tuning available specifications. If that is not the case, and totally new specifications are needed, these can be easily realised as intuitive graphical specifications, drawings, like those shown, for instance, in Figures 3.3 and 3.12. In this case, it is very likely that the designer makes hand of design patterns also available as part of the repository. For these purposes, we suggest the implementation of a development environment that permits designers to specify electronic institutions employing the graphical language introduced in Chapter 3. Moreover, the environment might also offer a textual specification language as the counterpart of the graphical language. Complementarily, the environment must also include facilities for the specification and development of institutional agents and interagents based on their general models presented in Chapter 4. Notice that in both cases we contemplate the development of both types of agents as a largely automated task, since for all institutions we shall be using the same general models of institutional agents and interagents. They will be differing in the specification each agent is endowed with.

Observe that we argue in favour of the construction of development environments that support both the specification and development of both the rules and the players. However, much effort must be dedicated to the production of infrastructures for agents (the players). At present, agent developers create separately the infrastructures required by their applications in an *ad-hoc* manner. And indeed infrastructures are needed to support agents, yet it seems apparent that much effort is replicated and unfortunately the resulting infrastructures are costly and usually only fulfil general requirements but those of the particular applications they are intended to support. Therefore, further developments may prevent reusing the same infrastructure or, in the best case, may require a re-engineering effort. Initiatives such as DARPA COABS [coabs, www] attempt at making headways in this direction —the COABS research community is developing a prototype *agent grid* as an infrastructure for the run-time integration of heterogeneous multi-agent and legacy systems.

Next we shall discuss further extensions and uses of the test-bed presented in Chapter 6. Recall that the auction house fully described in Chapter 5 is no more that one of the working modes of our test-bed.

Firstly, we regard as necessary the extension of the library of price-fixing mechanisms with the inclusion of additional auction protocols. In particular, not only single and multi-item auctions might be included, but also combinatorial auction where bidders can bid on combination of items [Sandholm, 1999a] —auction protocols largely used in multitude of markets such as f.i. electricity markets, equities trading and bandwith auctions to name a few. Complementarily, negotiation protocols are planned also to be introduced in order to turn our test-bed into a general experimental environment for exchanges with support for both automated and non-automated price-fixing mechanisms.

Secondly, more elaboration is needed in order to generate more realistic scenarios. Let us step back to Chapter 1 to recall that when referring to the actual fish market, we distinguished several types of buyers (wholesale buyers, fishmongers, restaurant owners). It is apparent that a buyer in a morning session (participated by a reduced number of wholesale buyers) is to behave differently to a buyer in an afternoon market session (participated by large numbers of fishmongers and restaurant owners). In general, when designing an agent's trading strategy for an actual scenario, it is fundamental to pitch the agent against others to evaluate it in the most similar scenarios to those that the agent is deemed to confront. For this purpose, it would be desirable for the agent designer to count on the possibility of generating artificial scenarios composed of agents with different profiles. In order to enable this possibility, some extensions should be made to the test-bed in order to provide facilities to generate populations of agents with different profiles.

Lastly we envision our test-bed as an appropriate tool to undertake research in both sequential strategies and multi-market auction strategies. Most agent research concerning bidding strategies has been applied to Double auction marketplaces [Park et al., 1999,Cliff and Bruten, 1998,Preist and van Tol, 1998,Friedman and Rust, 1991]. Limited work has considered the design of bidding strategies for sequential auctions using other auction protocols apart from Double auction. To the best of our knowledge, only Boutilier's [Boutilier et al., 1999] work and our own work [Garcia et al., 1998b] have started addressing this issue. Bidding in sequential auctions is difficult and with no doubt a challenging, open research problem. Notice that to determine his valuation for an item, the bidder needs to guess what items he will receive in the future. This requires speculating about what his rivals will bid in the future, and in turn what these bid in the future depends on what they believe the others will do.

As to simultaneous bidding strategies —i.e. bidding strategies for simultaneously participating in several auctions at the same time—, this is an issue still at his early infancy (f.i. [Preist, 2000] in the context of multiple marketplaces and [Excelente-Toledo et al., 2001] in the context of multi-agent coordination). Preist presents an algorithm that allows an agent to participate in many auctions, capable of coordinating the agent's bids across them. It is surprising the lack of work concerning this issue in spite of its value for both consumer-to-consumer and business-to-business marketplaces, and we believe that our test-bed can help experimental research in this direction.

Finally, we comment on the issues concerning reputation systems, which we find of pragmatic value to cope with deception and fraud in open agent-mediated electronic institutions. We believe that research in reputation systems is challenging and still in its infancy and so much work is needed in order to consolidate open agent-based systems. We find compelling that open agent-mediated electronic institutions count on reputation systems that provide guidance to participants when choosing partners in their activities.

Why reputation? Reputation matters. In fact, reputation is the main basis of our economic system. People stop doing business with those they don't trust, or who mistreat them, or those who just don't seem to be reputable.

At present, the mechanisms appearing in the literature have fallen short to cover a broad range of delicate issues. Firstly, there is the issue of combining reputation values stemming from several sources of information. But then, the reliability of sources of information must be considered since agents might be lying when providing reputation information. Preliminary models have considered this issue [Schillo et al., 1999] tring to find cheaters to undervalue their credibility. However it seems to us that a different approach is needed, and perhaps the deployment of mechanisms that make agents act as truthfully as possible can succeed.

In general, reputation systems are not foolproof. Another major source of cheating is the discriminatory use of rating among agents. Thus, a group of agents might collude to destroy the reputation of a common competitor, whereas they migth also collude rating one another well enough to wake the interest of other agents in the community. This is a particular case of collusive behaviour among agents to cheat on reputation systems, but more generally collusive behaviour is a tremendously delicte problem in cybercommunities. Collusive behaviour in auctions and to orchestrate a distributed denial of services attack are just a two important examples of this tpe of phenomenon. Although highly challenging (f.i. in the economics literature collusive behaviour has not been dealt with profoundly and although it is admitted to be a very hard and important problem, just a few works have made headway in this issue [Hendricks and Porter, 1989,Porter and Zona, 1993]).

Turning our attention to reputation values, we observe that the vast majority of research considers reputation as a unidemensional. Contrarily, we argue that reputation is clearly multi-dimensional. For instance, an agent migth be able to deliver perfect quality but repeatedly late. Both dimensions, quality and delivery time, should be considered by other agents when assessing an eventual deal.

And then, what to do with reputation values? No much work has been done relating reputation and decision-making processes. Mostly the focus has been on the use of reputation to avoid to choose *wrong* partners in the context of task delegation [Biswas et al., 2000]. Nevertheless, to the best of our knowledge reputation information has not been integrated yet into negotiation mechanisms. We believe that it is worthy researching on how agent's trustwortyness can affect the strategic behaviour of negotiating parties.

Finally, in some cases reputation cannot only be used as a discriminatory value for trading partners, but also for institutions themselves. Think of multiple electronic institutions scattered around in the Internet. For instance, actual fish markets may have their electronic counterparts, and so a network of electronic fish markets could exist over the Internet. Since the fish market is in charge of assessing quality to incoming fish (by fixing a starting price), it might be the case that the institution overrates (some) sellers' products, eventually producing market inefficiencies. Feedback provided by buyers after acquiring goods can be valuable to rate sellers and for the institution itself to discover whether sellers are being overrated (ideally the institution should employ this information to tune its quality assessment process). But reputation information from different marketplaces can help agents decide where to buy. From the reputation information produced in a marketplace, an agent can infer the expected quality of different types of goods. Furthermore, such information might be taken into account by the agent when elaborating his bidding strategy by considering both the expected quality and the institutional quality assessment.

Appendix A

AmEI Document Type Definition

In this Appendix we provide the *Document Type Definitions* for the XML-based textual representations of the graphical specifications of scenes, performative structures, and electronic institutions presented in Chapter 3[1].

SCENE.DTD

```
<!ELEMENT SCENE (DESCRIPTION,DF,ROLES,SD*,STATE+,LINK*)>
<!ATTLIST SCENE
  ID              CDATA               #REQUIRED
>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT DF (ONTOLOGY,LANGUAGE,CL,PERFORMATIVES)>
<!ELEMENT ONTOLOGY (#PCDATA)>
<!ELEMENT LANGUAGE (#PCDATA)>
<!ELEMENT CL (#PCDATA)>
<!ELEMENT PERFORMATIVES (PNAME+)>
<!ELEMENT PNAME (#PCDATA)>
<!ELEMENT ROLES (ROLE+)>
<!ELEMENT ROLE (ROLEID,MIN,MAX)>
<!ELEMENT ROLEID (#PCDATA)>
<!ELEMENT MIN (#PCDATA)>
<!ELEMENT MAX (#PCDATA)>
<!ELEMENT TOP (#PCDATA)>
<!ELEMENT SD (ROLEPAIR+)>
<!ELEMENT ROLEPAIR (ROLEID,ROLEID)>
<!ELEMENT STATE (ACCESS?,EXIT?,EDGELIST*)>
```

[1]The DTDs are available at http://www.iiia.csic.es/ jar/PHD/AMEI/XML

```
<!ATTLIST STATE
  ID                CDATA                #REQUIRED
  TYPE              (initial|intermediate|final) "intermediate"
>
<!ELEMENT ACCESS (ROLEID+)>
<!ELEMENT EXIT (ROLEID+)>
<!ELEMENT EDGELIST (EDGE+)>
<!ATTLIST EDGELIST
  XML-LINK          CDATA                #FIXED "EXTENDED"
  ROLE              CDATA                #IMPLIED
  TITLE             CDATA                #IMPLIED
  INLINE            (TRUE|FALSE)         "TRUE"
  CONTENT-ROLE      CDATA                #IMPLIED
  CONTENT-TITLE     CDATA                #IMPLIED
  SHOW              (EMBED|REPLACE|NEW)  "EMBED"
  ACTUATE           (AUTO|USER)          "USER"
  BEHAVIOR          CDATA                #IMPLIED
>
<!ELEMENT EDGE (ILLOCUTION)>
<!ATTLIST EDGE
  ID                CDATA                #REQUIRED
  XML-LINK          CDATA                #FIXED "LOCATOR"
  ROLE              CDATA                #IMPLIED
  HREF              CDATA                #REQUIRED
  TITLE             CDATA                #IMPLIED
  SHOW              (EMBED|REPLACE|NEW)  "NEW"
  ACTUATE           (AUTO|USER)          "USER"
  BEHAVIOR          CDATA                #IMPLIED
>
<!ELEMENT ILLOCUTION (#PCDATA)>
```

PS.DTD

```
<!ELEMENT PS (DESCRIPTION,ROLES,NODE+,TRANSITION+,DSD?)>
<!ATTLIST PS
  ID                CDATA                #REQUIRED
>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT ROLES (ROLEID+)>
<!ELEMENT NODE (ARCLIST?)>
<!ATTLIST NODE
  ID                CDATA                #REQUIRED
  XML-LINK          CDATA                #FIXED "SIMPLE"
  ROLE              CDATA                #IMPLIED
  HREF              CDATA                #REQUIRED
  TITLE             CDATA                #IMPLIED
  INLINE            (TRUE|FALSE)         "TRUE"
  CONTENT-ROLE      CDATA                #IMPLIED
  CONTENT-TITLE     CDATA                #IMPLIED
  SHOW              (EMBED|REPLACE|NEW)  "REPLACE"
  ACTUATE           (AUTO|USER)          "USER"
  BEHAVIOR          CDATA                #IMPLIED
  MAX               CDATA                #REQUIRED
>
<!ELEMENT ARCLIST ANY>
<!ATTLIST ARCLIST
  XML-LINK          CDATA                #FIXED "EXTENDED"
  ROLE              CDATA                #IMPLIED
  TITLE             CDATA                #IMPLIED
  INLINE            (TRUE|FALSE)         "TRUE"
  CONTENT-ROLE      CDATA                #IMPLIED
  CONTENT-TITLE     CDATA                #IMPLIED
  SHOW              (EMBED|REPLACE|NEW)  "EMBED"
  ACTUATE           (AUTO|USER)          "USER"
  BEHAVIOR          CDATA                #IMPLIED
>
<!ELEMENT ARC (LABEL,CONDITION?)>
<!ATTLIST ARC
  ID                CDATA                #REQUIRED
  XML-LINK          CDATA                #FIXED "LOCATOR"
  ROLE              CDATA                #IMPLIED
  HREF              CDATA                #REQUIRED
  TITLE             CDATA                #IMPLIED
  SHOW              (EMBED|REPLACE|NEW)  "NEW"
  ACTUATE           (AUTO|USER)          "USER"
  BEHAVIOR          CDATA                #IMPLIED
>
<!ELEMENT ROLEID (#PCDATA)>
<!ELEMENT LABEL (AGENTVAR,ROLEID)+>
<!ELEMENT AGENTVAR (#PCDATA)>
<!ELEMENT CONDITION (#PCDATA)>
<!ATTLIST CONDITION
  ML                CDATA                #REQUIRED
>
<!ELEMENT TRANSITION (ARCLIST?)>
<!ATTLIST TRANSITION
  ID                CDATA                #REQUIRED
  TYPE              (AND|OR|AND-OR|OR-AND)  "AND"
>
<!ELEMENT DSD (ROLEPAIR+)>
<!ELEMENT ROLEPAIR (ROLEID,ROLEID)>
```

AMEI.DTD

```
<!ELEMENT AMEI (DESCRIPTION,ROLESET,INHERITANCE?,SSD?,PS,NORMS?)>
<!ATTLIST AMEI
  ID                CDATA                #REQUIRED
>
<!ELEMENT ROLESET (ROLEITEM+)>
<!ELEMENT ROLEITEM (ROLEID,DESCRIPTION,CARDINALITY)>
<!ELEMENT ROLEID (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT CARDINALITY (#PCDATA)>
<!ELEMENT INHERITANCE (INHITEM+)>
<!ELEMENT INHITEM (ROLEID,ROLEID)>
<!ELEMENT SSD (ROLEPAIR+)>
<!ELEMENT DSD (ROLEPAIR+)>
<!ELEMENT ROLEPAIR (ROLEID,ROLEID)>
<!ELEMENT PS ANY>
<!ATTLIST PS
  ID                CDATA                #REQUIRED
  XML-LINK          CDATA                #FIXED "SIMPLE"
  ROLE              CDATA                #IMPLIED
  HREF              CDATA                #REQUIRED
  TITLE             CDATA                #IMPLIED
  INLINE            (TRUE|FALSE)         "TRUE"
  CONTENT-ROLE      CDATA                #IMPLIED
  CONTENT-TITLE     CDATA                #IMPLIED
  SHOW              (EMBED|REPLACE|NEW)  "REPLACE"
  ACTUATE           (AUTO|USER)          "USER"
  BEHAVIOR          CDATA                #IMPLIED
>
<!ELEMENT NORMS (NORMITEM*)>
<!ATTLIST NORMS
  ML                CDATA                #REQUIRED
>
<!ELEMENT NORMITEM (#PCDATA)>
```

Bibliography

[Dav, 1993] (1993). *Software Requirements. Objects, Functions and States.* Prentice Hall International, Inc.

[Gei, 1994] (1994). *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press.

[Dor, 1994] (1994). *Simulating Societies*, chapter The EOS Project: Modelling Upper Paleolithic Social Change. UCL Press.

[Jen, 1995a] (1995a). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer.

[Jen, 1995b] (1995b). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 2. Springer.

[Fis, 1996] (1996). *Foundations of Distributed Artificial Intelligence*, chapter AGenDA - A General Testbed for Distributed Artificial Intelligence Applications, pages 401–427. John Wiley & Sons, Inc.

[Som, 1996] (1996). *Foundations of Distributed Artificial Intelligence*, chapter The Evolution of the CooperA Platform, pages 365–447. John Wiley & Sons, Inc.

[Bra, 1997] (1997). *Software Agents*, chapter KAoS: Toward an Industrial-strength Open Agent Architecture. AAAI/MIT Press.

[Pro, 1999] (1999). *Proceedings of the International Workshop on Knowledge-based Planning for Coalition Forces.*

[Adauction, www] Adauction (www). Adauction URL. http://www.-Adauction.com.

[Agentbuilder, www] Agentbuilder (www). Agentbuilder URL. http://www.-agentbuilder.com.

[Agha, 1986] Agha, G. (1986). *Actors, A Model of Concurrent Computation in Distributed Systems.* The MIT Press.

[Aho and Ullman, 1972] Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing of *Series in Automatic Computation*. Prentice-Hall.

[Albers et al., 1999] Albers, M., Jonker, C. M., Karami, M., and Treur, J. (1999). An electronic market place: Generic agent models, ontologies and knowledge. In *Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'99)*, pages 211–228.

[Alberts, 1993] Alberts, L. K. (1993). *YMIR: An Ontology for Engineering Design*. PhD thesis, University of Twente.

[Amazon, www] Amazon (www). Amazon URL. http://www.amazon.com.

[Andrews and Prager, 1994] Andrews, M. and Prager, R. (1994). *Genetic Programming for the Acquisition of Double Auction Market Strategies*, pages 355–368. The MIT Press.

[AuctionLand, www] AuctionLand (www). AuctionLand URL. http://www.-neomax.com.

[AuctionLine, www] AuctionLine (www). AUCTIONLINE. http://www.-auctionline.com.

[Austin, 1962] Austin, J. L. (1962). *How to Do Things With Words*. Oxford University Press.

[Barbuceanu and Fox, 1995] Barbuceanu, M. and Fox, M. S. (1995). Cool: A language for describing coordination in multi-agent systems. In *Proceedings of the First International Conference in Multi-Agent Systems (ICMAS-95)*, pages 17–24. AAAI Press.

[Barbuceanu et al., 1998] Barbuceanu, M., Gray, T., and Mankovski, S. (1998). Coordinating with obligations. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, pages 62–69.

[Bargainfinder,] Bargainfinder. BargainFinder URL. http://bf.cstar.ac.com/bf.

[Barkley, 1997] Barkley, J. (1997). Comparing simple role based access control models and access control lists. In *Proceedings of the Second ACM Workshop on Role-based Access Control*.

[Barkley et al., 1997] Barkley, J., Cincotta, A. V., Ferraiolo, D. F., Gavrilla, S., and Kuhn, D. R. (1997). Role based access control for the world wide web. In *20th National Information System Security Conference*.

[Becht et al., 1999] Becht, M., Gurzki, T., Klarmann, J., and Muscholl, M. (1999). Rope: Role oriented programming environment for multiagent systems. In *Proceedings of the Fourth IFCIS Conference on Cooperative Information Systems (CoopIs'99)*.

[Beegent, www] Beegent (www). Bee-gent URL. http://www2.toshiba.co.jp/-beegent.

[Béjar and Rodríguez-Aguilar, 2001] Béjar, J. and Rodríguez-Aguilar, J. A. (2001). To Bid or not To Bid. Agent Strategies in Electronic Auction Games. In Dignum, F. and Cortés, U., editors, *Agent-mediated Electronic Commerce III. Current Issues in Agent-based Electronic Commerce Systems*, number 2003 in Lecture Notes in Artificial Intelligence, pages 173–191. Springer-Verlag.

[Belakhdar and Ayel, 1996] Belakhdar, O. and Ayel, J. (1996). Modelling approach and tool for designing protocols for automated negotiation in multi-agent systems. In van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away*, number 1038 in Lecture Notes in Artificial Intelligence, pages 100–115. Springer-Verlag.

[Bidfind, www] Bidfind (www). bidfind URL. http://www.bidfind.com.

[Biswas et al., 2000] Biswas, A., Sen, S., and Debnath, S. (2000). Limiting deception in groups of social agents. *Applied Artificial Intelligence Journal.* To appear.

[Boutilier et al., 1999] Boutilier, C., Goldszmidt, M., and Sabata, B. (1999). Sequential auctions for the allocation of resources with complementarities. In *Proceedings of the Sixteenth International Joint Conference in Artificial Intelligence (IJCAI-99).*

[Brazier et al., 1998] Brazier, F., Jonker, C., and Treur, J. (1998). Principles of compositional multi-agent system development. In cuena, J., editor, *Proceedings of the 15th IFIP World Computer Congress (WCC'98), Conference on Information Technology and Knowledge Systems (IT&KNOWS'98)*, pages 347–360.

[Brazier et al., 1997] Brazier, F. M. T., Keplicz, B. D., Jennings, N. R., and Treur, J. (1997). Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6(1):67–94.

[Briot, 1998] Briot, J.-P. (1998). Agents and concurrent objects. jean-pierre briot interviews less gasser. *IEEE Concurrency.*

[Brookshear, 1989] Brookshear, J. G. (1989). *Theory of Computation, Formal Languages, Automata, and Complexity.* The Benjamin/Cummings Publishing.

[Buhr et al., 1997] Buhr, R. J. A., Elammari, M., Gray, T., Mankowski, S., and Pinard, D. (1997). Understanding and defining the behaviour of systems of agents with use case maps. In *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97).*

[Burmeister, 1996] Burmeister, B. (1996). Models and methodologies for agent-oriented analysis and design. In *Working Notes of the KI'96 Workshop on Agent-oriented Programming and Distributed Systems.* DFKI Document D-96-06.

[Carley and Gasser, 1999] Carley, K. M. and Gasser, L. (1999). *Distributed Artificial Intelligence*, chapter Computational Organization Theory. MIT Press, Cambridge, MA.

[Castelfranchi, 1995] Castelfranchi, C. (1995). Commitments: From individual intentions to groups and organizations. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 41–48, Menlo Park, CA. AAAI Press.

[Castelfranchi, 1996] Castelfranchi, C. (1996). Prescribed mental attitudes in goal-adoption and norm-adoption. In Conte, R. and Falcone, R., editors, *Proceedings of the ICMAS-96 Workshop on Norms, Obligations, and Conventions.*

[Chauhan, 1997] Chauhan, D. (1997). *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation.* PhD thesis, ECECS Department, University of Cincinnati.

[Chavez et al., 1997] Chavez, A., Dreilinger, D., Guttman, R., and Maes, P. (1997). A real life experiment in creating an agent marketplace. In *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97).*

[Chavez and Maes, 1996] Chavez, A. and Maes, P. (1996). Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90.

[Cliff and Bruten, 1998] Cliff, D. and Bruten, J. (1998). Less than humans: Simple adaptive trading agents for cda markets. In *Proceedings of the 1998 Symposium on Computation in Economics, Finance, and Engineering: Economic Systems.*

[coabs, www] coabs (www). Control of Agent Based Systems. http://-coabs.globalinfotek.com.

[Cohen et al., 1989] Cohen, P., Greenberg, M., Hart, D., and Howe, A. (1989). Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3):33–48.

[Cohen and Levesque, 1991] Cohen, P. R. and Levesque, H. J. (1991). Confirmation and joint actions. In *Proceeding of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91).*

[Cohen and Levesque, 1995] Cohen, P. R. and Levesque, H. J. (1995). Communicative actions for artificial agents. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 65–72, Menlo Park, CA. AAAI Press.

[Collins et al., 1998] Collins, J., Youngdahl, B., Jamison, S., Mobasher, B., and Gini, M. (1998). A Market Architecture for Multi-agent Contracting. In *Proceedings of the Second International Conference on Autonomous Agents (AGENTS'98)*, pages 285–292.

[Collins and Lee, 1998] Collins, J. C. and Lee, L. C. (1998). Building electronic marketplaces with the zeus agent tool-kit. In Noriega, P. and Sierra, C., editors, *Agent Mediated Electronic Commerce*, number 1571 in Lecture Notes in Artificial Intelligence.

[Connolly, 1997] Connolly, D. (1997). *XML, Principles, Tools and Techniques.* O'REILLY.

[Corkill and Lesser, 1983] Corkill, D. D. and Lesser, V. (1983). The use of meta-level control for coordination in a distributed problem solving network. In Bond, A. H. and Gasser, L., editors, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756. Karlsruhe, Federal Republic of Germany, Morgan Kaufmann Publishers.

[Cost et al., 1999] Cost, R. S., Chen, Y., Finin, T., Labrou, Y., and Peng, Y. (1999). Modeling agent conversations with colored petri nets. In *AGENTS'99 Workshop on Specifying and Implementing Conversation Policies.*

[Cost et al., 1998] Cost, R. S., Finin, T., Labrou, Y., Luan, X., Peng, Y., Soboroff, I., Mayfield, J., and Boughannam, A. (1998). Jackal: a java-based tool for agent development. In *AAAI-98 Workshop on Software Tools for Developing Agents.*

[Davis and Smith, 1983] Davis, R. and Smith, R. (1983). Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109.

[de Toro, 1997] de Toro, M. C. (1997). A hybrid buyer agent architecture for the fishmarket tournaments. Master's thesis, Universitat Autonoma de Barcelona.

[de Velde, 1997] de Velde, W. V. (1997). Co-habited mixed reality. In *IJCAI'97 Workshop on Social Interaction in Communityware.*

[Decker, 1987] Decker, K. S. (1987). Distributed problem solving: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 17:729–740.

[Decker, 1995] Decker, K. S. (1995). *Environment Centered Analysis and Design of Coordination Mechanisms.* PhD thesis, University of Massachussets.

[Decker, 1996] Decker, K. S. (1996). *Distributed Artificial Intelligence Testbeds*, chapter 3. John and Wiley & Sons, Inc. Edited by G. M P. O'Hare and N. R. Jennings.

[Dellarocas and Klein, 1999] Dellarocas, C. and Klein, M. (1999). Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces. In *Proceedings ACM Conference on Electronic Commerce (EC-99).*

[Diller, 1990] Diller, A. (1990). *Z An Introduction to Formal Methods.* John Wiley & Sons, Inc.

[d'Inverno et al., 1998a] d'Inverno, M., Kinny, D., and Luck, M. (1998a). Interaction protocols in agentis. In *Proceedings of the Third International Conference on Multi-agent Systems (ICMAS-98)*, pages 112–119.

[d'Inverno et al., 1998b] d'Inverno, M., Kinny, D., Luck, M., and Wooldridge, M. (1998b). A formal specification of dmars. In Rao, A. S. and Wooldridge, M., editors, *Fourth International Workshop on Agent Theories, Architectures and Languages*, number 1365 in Lecture Notes in Artificial Intelligence, pages 155–176. Springer-Verlag.

[d'Inverno and Luck, 1999] d'Inverno, M. and Luck, M. (1999). Development and application of a formal agent framework.

[Dooley, 1976] Dooley, R. A. (1976). Repartee as a graph. *Appendix B in Longacre*, (76):348–358.

[Durfee et al., 1987] Durfee, E. H., Lesser, V. R., and Corkill, D. D. (1987). Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, C-36(11):1275–1291. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 268–284, Morgan Kaufmann, 1988.).

[eBay, www] eBay (www). http://www.eBay.com.

[Economist, 1999] Economist, T., editor (1999). *A Survey of Business and the Internet. The Net Imperative.*

[Esteva et al., 2000a] Esteva, M., Rodríguez-Aguilar, J. A., Arcos, J. L., Sierra, C., and Garcia, P. (2000a). Institutionalising open multi-agent systems. a formal approach. In *Proceedings of the Fourth International Conference on Multi-agent Systems (ICMAS-00).*

[Esteva et al., 2000b] Esteva, M., Rodríguez-Aguilar, J. A., Sierra, C., Arcos, J. L., and Garcia, P. (2000b). *European Perspectives on Agent-mediated Electronic Commerce*, chapter On the Formal Specification of Electronic Institutions. Springer-Verlag. to appear (expected 2000).

[Etzioni, 1964] Etzioni, A. (1964). *Modern Organizations.* Englewood Cliffs, NJ, Prentice-Hall.

[Excelente-Toledo et al., 2001] Excelente-Toledo, C. B., Bourne, R. A., and Jennings, N. R. (2001). Reasoning about commitments and penalties for coordination. In *Proceedings fo the Fifth International Conference on Autonomous Agents*. To appear.

[Farhoodi et al., 1991] Farhoodi, F., Proffitt, J., Woodman, P., and Tunnicliffe, A. (1991). Design of organizations in distributed decision systems. In *AAAI Workshop on Cooperative Heterogeneous Intelligent Systems*, Anaheim,CA.

[Ferber and Gutknetch, 1998] Ferber, J. and Gutknetch, O. (1998). A meta-model for the analysis of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 128–135.

[Ferraiolo et al., 1999] Ferraiolo, D. F., Barkley, J. F., and Kuhn, D. R. (1999). A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information Systems Security*, 1(2).

[Ferraiolo et al., 1995] Ferraiolo, D. F., Cugini, J. A., and Kuhn, D. R. (1995). Role-based access control (rbac): Features and motivations. In *Annual Computer Security Applications Conference*. IEEE Computer Society Press.

[Finin et al., 1995] Finin, T., Labrou, Y., and Mayfield, J. (1995). Kqml as an agent communication language. In Bradshaw, J., editor, *Software Agents*. MIT Press, Cambridge. invited chapter.

[FIPA, 1998] FIPA (1998). Fipa 97 specification version 2.0 part 2. Technical report, FIPA - Foundation for Intelligent Physical Agents.

[Firefly, www] Firefly (www). Firefly URL. http://www.firefly.com.

[Fischer, 1993] Fischer, K. (1993). The rule-based multi-agent system magsy. In *Cooperative Knowledge-based Systems SIG Proceedings Workshop*, Keele, UK.

[Fischer and al., 1996] Fischer, K. and al. (1996). Intelligent agents in virtual enterprises. In *First International Conference on the Practical Application of Intelligent Agents and Multi-agent Technology (PAAM'96)*.

[Fisher et al., 1997] Fisher, M., Muller, J., Schroeder, M., Staniford, G., and Wagner, G. (1997). Methodological foundations for agent-based systems. *The Knowledge Engineering Review*, 12(3):323–329.

[Fisher and Wooldridge, 1997] Fisher, M. and Wooldridge, M. (1997). On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, 6(1).

[Fishmarket, www] Fishmarket (www). The Fishmarket Project. http://-www.iiia.csic.es/Projects/fishmarket.

[Forum, 1994] Forum, M. P. I. (1994). Mpi: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4).

[Franklin and Grasser, 1996] Franklin, S. and Grasser, A. (1996). *Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 21–35. Springer Verlag.

[Friedman and Rust, 1991] Friedman, D. and Rust, J., editors (1991). *The Double Auction Market: Institutions, Theories, and Evidence.* Addison-Wesley Publishing Company. Proceedings of the Workshop on Double Auction Markets held June, 1991, Santa Fe, New Mexico.

[Galan and Baker, 1999] Galan, A. K. and Baker, A. D. (1999). Multi-agent communication in jafmas. In *AGENTS'99 Workshop on Specifying and Implementing Conversation Policies.*

[Garavel, 1996] Garavel, H. (1996). An overview of the eucalyptus toolbox. In Vrezocnik, Z. and Kapus, T., editors, *Proceedings of the COST247 International Workshop on Applied Formal Methods in System Design*, University of Maribor, Slovenia.

[Garcia et al., 1998a] Garcia, P., Giménez, E., Godo, L., and Rodríguez-Aguilar, J. A. (1998a). Possibilistic-based design of bidding strategies in electronic auctions. In *The 13th biennial European Conference on Artificial Intelligence (ECAI-98).*

[Garcia et al., 1998b] Garcia, P., Godo, L., Rodríguez-Aguilar, J. A., and Gimenez, E. (1998b). Bidding strategies for trading agents in auction-based tournaments. In Sierra, C. and Noriega, P., editors, *Agent-Mediated Electronic Trading*, number 1571 in Lecture Notes in Artificial Intelligence, pages 151–165. Springer-Verlag.

[Gasser et al., 1987] Gasser, L., Braganza, C., and Herman, N. (1987). *Distributed Artificial Intelligence*, chapter MACE: A flexible test-bed for distributed AI research, pages 119–152. Pitman Publishers.

[Gavrila and Barkley, 1998] Gavrila, S. I. and Barkley, J. F. (1998). Formal specification for role based access control user/role and role/role relationship management. In *Proceedings of the Third ACM Workshop on Role-based Access Control.*

[Genesereth and Fikes, 1992] Genesereth, M. R. and Fikes, R. E. (1992). Knowldege interchange format version 3.0 reference manual. Technical Report Report Logic–92–1, Logic Group, Computer Science Department, Standford University.

[Genesereth and Ketchpel, 1994] Genesereth, M. R. and Ketchpel, S. P. (1994). Software agents. *Communications of the ACM, Special Issue on Intelligent Agents*, 37(7):48–53.

[Giménez et al., 1998] Giménez, E., Godo, L., Rodríguez-Aguilar, J. A., and Garcia, P. (1998). Designing bidding strategies for trading agents in electronic auctions. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 136–143.

[Ginsberg, 1991] Ginsberg, M. L. (1991). Knowledge interchange format: The kif of death. *AI Magazine*, 12(3):57–63.

[Gosling, 1996] Gosling, J. (1996). *The Java Programming Language*. Addison-Wesley, Reading.

[Gossip, www] Gossip (www). Gossip URL. http://www.tryllian.com.

[Gruber, 1993a] Gruber, T. R. (1993a). Toward principles for the design of ontologies used for knowledge sharing. In Guarino, N. and Poli, R., editors, *Software Agents*. Kluwer Academic.

[Gruber, 1993b] Gruber, T. R. (1993b). A translation approach to portable ontology specifications. Technical Report KSL–92–71, Knowledge Systems Laboratory, Computer Science Department, Standford University.

[Guttman and Maes, 1998] Guttman, R. and Maes, P. (1998). Agent-mediated integrative negotiation for retail electronic commerce. In *Proceedings of the First International Workshop on Agent-mediated Electronic Trading*. to appear.

[Guttman et al., 1998] Guttman, R. H., Moukas, A. G., and Maes, P. (1998). Agent-mediated electronic commerce: A survey. *The Knowledge Engineering Review*, 13(2):147–159.

[Hanks et al., 1993] Hanks, S., Pollack, M. E., and Cohen, P. R. (1993). Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magazine*, pages 17–42.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, (8):231–274.

[Hendricks and Porter, 1989] Hendricks, K. and Porter, R. H. (1989). Collusions in auctions. *Annales d'Economie et de Statistique*, (15–16):217–230.

[Hewitt, 1986] Hewitt, C. (1986). Offices are open systems. *ACM Transactions of Office Automation Systems*, 4(3):271–287.

[Huberman and Clearwater, 1995] Huberman, B. A. and Clearwater, S. (1995). A multi-agent system for controlling builging environments. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 171–176. AAAI Press.

[IEEE, 1993] IEEE (1993). Special issue on computer support for concurrent engineering. *IEEE Computer*, 26(1).

[Iglesias et al., 1999] Iglesias, C. A., Garijo, M., and Gonzalez, J. C. (1999). A survey of agent-oriented methodologies. Lecture Notes in Artificial Intelligence. Springer-Verlag.

[InterAUCTION, www] InterAUCTION (www). InterAUCTION. http://-www.interauction.com.

[Jacobson et al., 1996] Jacobson, I., Christerrson, M., Jonsson, P., and Overgaard, G. (1996). *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley.

[Jango, www] Jango (www). Jango URL. http://www.jango.com.

[JDBC, www] JDBC (www). JDBC. The JDBC Database Access API. http://splash.javasoft.com/products/jdbc/index.html.

[Jennings, 1995] Jennings, N. R. (1995). Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250.

[Jennings et al., 1998] Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-agent Systems*, 1:275–306.

[Kaminka and Tambe, 1999] Kaminka, G. and Tambe, M. (1999). I'm ok, you're ok, we're ok: Experiments in distributed and centralized socially attentive monitoring. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, pages 213–220.

[Kasbah,] Kasbah. Kasbah URL. http://kasbah.media.mit.edu.

[Kendall, 1998] Kendall, E. A. (1998). Agent roles and role models: New abstractions for intelligent agent system analysis and design. In *Proceedings of Intelligent Agents for Information and Process Management (AIP'98).*

[Kimborough and Moore, 1997] Kimborough, S. O. and Moore, S. (1997). On automated message processing in electronic commerce and work support systems: Speech act theory and expressive felicity. *Transactions on Information Systems*, 15(4):321–367.

[Kinny et al., 1996] Kinny, D., Georgeff, M., and Rao, A. (1996). A methodology and modelling technique for systems of bdi agents. In de Velde, W. V. and Perram, J. W., editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-agent World*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 56–71. Springer-Verlag.

[Kitano et al., 1997] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997). Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, pages 340–347.

[Klein, 2000] Klein, M. (2000). The challenge: Enabling robust open multi-agent systems. Project proposal.

[Klein and Dellarocas, 1999] Klein, M. and Dellarocas, C. (1999). Exception handling in agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, pages 62–68.

[Koning, 2000] Koning, J.-L. (2000). *European Perspectives on Agent-mediated Electronic Commerce*, chapter Designing Interaction Protocols for Electronic Commerce Applications. Springer-Verlag. To appear (expected 2000).

[Koning et al., 1998] Koning, J.-L., Franois, G., and Demazeau, Y. (1998). Formalization and pre-validation for interaction protocols in multiagent systems. In Prade, H., editor, *Proceedings of the 13th European Conference on Artificial Intelligence*, pages 298–302. John Wiley & Sons, Ltd.

[Kortenkamp et al., 1997] Kortenkamp, D., Nourbakhsh, I., and Hinkle, D. (1997). The 1996 AAAI Mobile Robot Competition and Exhibition. *AI Mag.*, 18(1):25–32.

[Kuwabara, 1995] Kuwabara, K. (1995). Agentalk: Coordination protocol description for multi-agent systems. In *Proceedings of the First International Conference on Multi-agent Systems.*

[Labrou, 1997] Labrou, Y. (1997). *Semantics for an Agent Communication Language.* PhD thesis, University of Mariland Baltimore County.

[Lamport, 1978] Lamport, L. (1978). Time, clocks and the ordering of events in a distributed system. *Comm. of the ACM*, 21(7):558–565.

[Lesser and Corkill, 1983] Lesser, V. and Corkill, D. (1983). The distributed vehichle monitoring testbed. *AI Magazine*, 4(3):63–109.

[Lesser, 1995] Lesser, V. R. (1995). Multiagent systems: An emerging subdiscipline of ai. *ACM Computing Surveys*, 27:340–342.

[Lesser, 1998] Lesser, V. R. (1998). Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous Agents and Multi-Agent Systems*, 1:89–111.

[Lin et al., 1999] Lin, F., Norrie, D. H., Shen, W., and Kremer, R. (1999). A schema-based approach to specifying conversation policies. In *AGENTS'99 Workshop on Specifying and Implementing Conversation Policies.*

[Martin et al., 1998] Martin, F. J., Plaza, E., and Arcos, J. L. (1998). Knowledge and experience reuse through communication among competent (peer) agents. To appear in International Journal of Software Engineering and Knowledge Engineering.

[Martín et al., 2000a] Martín, F. J., Plaza, E., and Rodríguez-Aguilar, J. A. (2000a). Conversation protocols: Modelling and implementing conversations in agent-based systems. In Greaves, M. and Dignum, F., editors, *Issues in Agent Communication*, number 1916 in Lecture Notes in Artificial Intelligence. Springer-Verlag. To appear.

[Martín et al., 2000b] Martín, F. J., Plaza, E., and Rodríguez-Aguilar, J. A. (2000b). An infrastructure for agent-based systems: An interagent approach. *International Journal of Intelligent Systems*, 15(3). to appear.

[Martín et al., 1998] Martín, F. J., Plaza, E., Rodríguez-Aguilar, J. A., and Sabater, J. (1998). Java interagents for multi-agent systems. In *Proceedings of the AAAI-98 Workshop on Software Tools for Developing Agents.*

[Mayfield et al., 1996] Mayfield, J., Labrou, Y., and Finin, T. (1996). Evaluation of kqml as an agent communication language. In Wooldridge, M. and Müller, J., editors, *Intelligent Agents II*, pages 347–360. Springer Verlag.

[McAfee and McMillan, 1987] McAfee, R. P. and McMillan, J. (1987). Auctions and bidding. *J. Ec Lit.*, XXV:699–738.

[Milgrom and Webber, 1982] Milgrom, P. R. and Webber, R. J. (1982). A theory of auctions and competitive bidding. *Econometrica*, 50(5):1089–1122.

[Montgomery et al., 1992] Montgomery, T., Lee, J., Musliner, D., Durfee, E., Dartmouth, D., and So, Y. (1992). Mice users guide. Technical report, Dept. of Electrical Engineering and Computer Science, Univ. of Michigan. Technical Report, CSE-TR-64-90.

[Moore, 1999] Moore, S. A. (1999). On conversation policies and the need for exceptions. In *AGENTS'99 Workshop on Specifying and Implementing Conversation Policies.*

[Moss and Edmonds, 1997] Moss, S. and Edmonds, B. (1997). A formal preference-state model with qualitative market judgements. *Omega – the International Journal of Management Science*, 25(2):155–169.

[Mullen and Wellman, 1996] Mullen, T. and Wellman, M. (1996). Market-based negotiation for digital library services. In *Second USENIX Workshop on Electronic Commerce, Oakland, CA.*

[Muller and Pischel, 1994] Muller, J. P. and Pischel, M. (1994). An architecture for dynamically interacting agetns. *International Journal of Cooperative Information Systems*, 3(1):25–45.

[mysimon, www] mysimon (www). Mysimon.com. http://www.mysimon.com.

[Napoli et al., 1996] Napoli, C., Giordano, M., Furnari, M., Sierra, C., and Noriega, P. (1996). A pvm implementation of the fishmarket. In *IX International Symposioum on Artificial Intelligence, Cancun, Mexico.*

[Ndumu et al., 1999] Ndumu, D. T., Nwana, H. S., Lee, L. C., and Collins, J. C. (1999). Visualising and debugging distributed multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, pages 326–333.

[Nguyen et al., 1993] Nguyen, D., Hanks, S., and Thomas, C. (1993). The truckworld manual. Technical Report 93-09-08, Department of Computer Science and Engineering, University of Washington.

[Nodine and Unruh, 1999] Nodine, M. H. and Unruh, A. (1999). Constructing robust conversation policies in dynamic agent communities. In *AGENTS'99 Workshop on Specifying and Implementing Conversation Policies*.

[Noriega, 1997a] Noriega, P. (1997a). *Agent-Mediated Auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona. Also to appear in IIIA mongraphy series.

[Noriega, 1997b] Noriega, P. (1997b). *Agent-Mediated Auctions: The Fishmarket Metaphor*. Number 8 in IIIA Monograph Series. Institut d'Investigació en Intel.ligència Artificial (IIIA). PhD Thesis.

[Noriega and Sierra, 1996] Noriega, P. and Sierra, C. (1996). Towards layered dialogical agents. In *Third International Workshop on Agent Theories, Architectures, and Languages, ATAL-96*.

[North, 1990] North, D. (1990). *Institutions, Institutional Change and Economics Perfomance*. Cambridge U. P.

[Odell et al., 2000] Odell, J., Parunak, H. V. D., and Bauer, B. (2000). Extending uml for agents. Submitted.

[O'Hare and Jennings, 1996] O'Hare, G. M. P. and Jennings, N. R. (1996). *Foundations of Distributed Artificial Intelligence*. John Wiley & Sons, Inc.

[Onsale,] Onsale. Onsale. http://www.onsale.com.

[Opensite, www] Opensite (www). Opensite Technologies URL. http://www.-opensite.com.

[pa So and Durfee, 1998] pa So, Y. and Durfee, E. H. (1998). *Simulating Organizations. Computational Models of Institutions and Groups*, chapter Designing Organizations for Computational Agents, pages 47–63. The MIT Press.

[Padget and Bradford, 1998] Padget, J. and Bradford, R. (1998). A π-calculus model of a spanish fish market. In Sierra, C. and Noriega, P., editors, *Agent-Mediated Electronic Trading*, number 1571 in Lecture Notes in Artificial Intelligence, pages 166–188. Springer-Verlag.

[Padget et al., 1993] Padget, J., Bretthauer, H., and Nuyens, G. (1993). An overview of eulisp. *Lisp and Symbolic Computation*, 6(1/2):9–98.

[Park et al., 1999] Park, S., Durfee, E., and Birmigham, W. (1999). An adpative bidding strategy based on stochastic modelling. In *Proceedings of the Third International Conference on Autonomous Agents*.

[Parsons et al., 1998] Parsons, S., Sierra, C., and Jennings, N. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*. to appear.

[Parunak, 1996] Parunak, H. V. D. (1996). Visualizing agent conversations: Using enhanced dooley graph for agent design and analysis. In *Proceedings of the Second International Conference on Multi-Agent Systems*.

[Patil et al., 1992] Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., McKay, D., Finin, T., Gruber, T. R., and Neches, R. (1992). The darpa knowledge sharing effort: Progress report. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*.

[Pattison et al., 1987] Pattison, H. E., Corkill, D. D., and Lesser, V. R. (1987). *Distributed Artificial Intelligence*, chapter Instantiating Descriptions of Organizational Structures, pages 59–96. Pitman Publishers.

[PersonaLogic,] PersonaLogic. PersonaLogic URL. http://www.personalogic.-com.

[Pitt and Mamdani, 1999] Pitt, J. and Mamdani, A. (1999). A protocol-based semantics for an agent communication language. In *Proceedings of the International Joint Converence in Artificial Intelligence (IJCAI'99)*, pages 486–491.

[Plaza et al., 1999] Plaza, E., Arcos, J. L., Noriega, P., and Sierra, C. (1999). Competing agents in agent-mediated insitutions. *Personal Technologies*, (2):212–220.

[Pnueli, 1986] Pnueli, A. (1986). Specification and development of reactive systems. In *Information Processing 86*. Elsevier Science Publishers.

[Pollack and Ringuette, 1990a] Pollack, M. and Ringuette, M. (1990a). Introducing the tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189.

[Pollack and Ringuette, 1990b] Pollack, M. and Ringuette, M. (1990b). Introducing the tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189.

[Porter and Zona, 1993] Porter, R. H. and Zona, J. D. (1993). Detection of bid rigging in procurement auctions. *Journal of Political Economy*, 101(3):518–538.

[Preist, 2000] Preist, C. (2000). Algorithm design for agents which participate in multiple simultaneous auctions. In *Proceedings of the Third International Workshop on Agent-mediated Electronic Trading (AMEC III).*

[Preist and van Tol, 1998] Preist, C. and van Tol, M. (1998). Adaptive agents in a persistent shout double auction. In *Proceedings of the 1st International Conference on the Internet, Computing and Economics.* ACM Press.

[Rajan and Slagle, 1996] Rajan, V. and Slagle, J. R. (1996). The use of artificially intelligent agents with bounded rationality in the study of economic markets. In *AAAI-96.*

[Riehle and Gross, 1998] Riehle, D. and Gross, T. (1998). Role model based framework design and integration. In *Proceedings of the 1998 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'98)*, pages 117–133. ACM Press.

[RMI, www] RMI (www). RMI. Java Remote Method Invocation. http://splash.javasoft.com/products/jdk/rmi/index.html.

[Roche and Schabes, 1997] Roche, E. and Schabes, Y. (1997). *Finite State Language Processing.* The MIT Press.

[Rodríguez-Aguilar et al., 1998a] Rodríguez-Aguilar, J. A., Martín, F. J., Giménez, F. J., and Gutiérrez, D. (1998a). Fm0.9beta users guide. Technical report, Institut d'Investigació en Intel.ligència Artificial. Technical Report, IIIA-RR98-32.

[Rodríguez-Aguilar et al., 2001] Rodríguez-Aguilar, J. A., Martín, F. J., Giménez, F. J., Gutiérrez, D., Molina, O., and Mateos, M. (2001). Fm 100 users' guide. Technical report, Institut d'Investigació en Intel.ligència Artificial. Technical Report. in press.

[Rodríguez-Aguilar et al., 1998c] Rodríguez-Aguilar, J. A., Martín, F. J., Noriega, P., Garcia, P., and Sierra, C. (1998c). Competitive scenarios for heterogeneous trading agents. In *Proceedings of the Second International Conference on Autonomous Agents (AGENTS'98)*, pages 293–300.

[Rodríguez-Aguilar et al., 1998b] Rodríguez-Aguilar, J. A., Martín, F. J., Noriega, P., Garcia, P., and Sierra, C. (1998b). Towards a test-bed for trading agents in electronic auction markets. *AI Communications*, 11(1):5–19.

[Rodríguez-Aguilar et al., 2000] Rodríguez-Aguilar, J. A., Martín, F. J., Noriega, P., Garcia, P., and Sierra, C. (2000). Towards a formal specification of complex social structures in multi-agent systems. In Padget, J. A., editor, *Collaboration between Human and Artificial Societies*, volume 1624 of *Lecture Notes in Artificial Intelligence*, pages 284–300. Springer-Verlag.

[Rodríguez-Aguilar et al., 1997] Rodríguez-Aguilar, J. A., Noriega, P., Sierra, C., and Padget, J. (1997). Fm96.5 a java-based electronic auction house. In *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology(PAAM'97)*, pages 207–224.

[Rosenschein and Genesereth, 1985] Rosenschein, J. S. and Genesereth, M. R. (1985). Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, Los Angeles, CA.

[Sandholm, 1999a] Sandholm, T. (1999a). An algorithm for optimal winner determination in combinatorial auctions. Technical Report WUCS-99-01, Washington University.

[Sandholm, 1999b] Sandholm, T. (1999b). emediator: A next generation electronic commerce server. Technical Report WUCS-99-02, Washington University.

[Sandhu et al., 1996] Sandhu, R., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role based access control models. *IEEE Computer*, 29(2).

[Schillo et al., 1999] Schillo, M., Funk, P., and Rovatsos, M. (1999). Who can you trust: Dealing with deception. In *Procedings of the Autonomous Agents Workshop on Deception, Fraud and Trust in Agent Societies, Autonomous Agents.*

[Scott, 1992] Scott, W. R. (1992). *Organizations: Rational, Natural, and Open Systems.* Englewood Cliffs, NJ, Prentice Hall.

[Searle, 1969] Searle, J. R. (1969). *Speech acts.* Cambridge U.P.

[Shardanand and Maes, 1995] Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating 'word of mouth'. In *Proceedings of the Computer-Human Interaction Conference (CHI'95).*

[Sierra and Noriega, 1998] Sierra, C. and Noriega, P. (1998). Institucions electròniques. In *Proceedings of the Primer Congrès Català d'Intel.ligència Artificial.*

[Singh, 1998] Singh, M. P. (1998). Developing formal specifications to coordinate heterogeneous autonomous agents. In *Proceedings of the Third International Conference on Multi-agent Systems (ICMAS-98)*, pages 261–268.

[Smith et al., 1998] Smith, I. A., Cohen, P. R., Bradshaw, J. M., Greaves, M., and Holmback, H. (1998). Designing conversation policies using joint intention theory. In *Proceedings of the Third International Conference on Multi-agent Systems (ICMAS-98)*, pages 269–276.

[SSL, www] SSL (www). SSLAVA. SSLava(tm) Information Center. http://-www.phaos.com.

[Stone and Veloso, 1996] Stone, P. and Veloso, M. (1996). Multiagent systems: A survey from a machine learning perspective. Submitted to IEEE Transactions on Knowledge and Data Engineering (TKDE).

[Sun, www] Sun (www). JOS. Java Object Serialization. http://chatsubo.javasoft.com/current/serial/index.html.

[Suttner and Sutcliffe, 1996] Suttner, C. B. and Sutcliffe, G. (1996). *ATP System Competition*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 146–160. Springer Verlag.

[Tete-a-Tete,] Tete-a-Tete. Tete-a-Tete URL. http://ecommerce.media.mit.-edu/Tete-a-Tete.

[uml2000, www] uml2000 (www). UML 2000 WORKSHOP Dynamic Behaviour in UML Models: Semantic Questions. http://www.disi.unige.it/person/-ReggioG/UMLWORKSHOP/CALLFORPAPER.html.

[Varian, 1995] Varian, H. R. (1995). Economic mechanism design for computerized agents. In *First USENIX Workshop on Electronic Commerce.*

[Vidal and Durfee, 1996] Vidal, J. M. and Durfee, E. H. (1996). Building agent models in economic societies of agents. In *Workshop on Agent Modelling (AAAI-96).*

[Weiss and Send, 1996] Weiss, G. and Send, S., editors (1996). *Adaption and Learning in Multiagent Systems.* Springer Verlag.

[Wellman, 1993] Wellman, M. P. (1993). A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, (1):1–23.

[Werner, 1987] Werner, E. (1987). *Distributed Artificial Intelligence*, chapter Cooperating Agents: A Unified Theory of Communication and Social Structure, pages 3–36. Pitman Publishers.

[Winograd and Flores, 1988] Winograd, T. and Flores, F. (1988). *Understanding Computers and Cognition.* Addison Wesley.

[Wooldridge et al., 1999] Wooldridge, M., Jennings, N. R., and Kinny, D. (1999). A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99).*

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-agent Systems*, 3(3). To appear.

[Wurman et al., 1998] Wurman, P. R., , Wellman, M. P., and Walsh, W. E. (1998). The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents. In *Second International Conference on Autonomous Agents (AGENTS'98)*, pages 301–308.

[Yellin and Strom, 1997] Yellin, D. M. and Strom, R. E. (1997). Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333.

[Ygge and Akkermans, 1996] Ygge, F. and Akkermans, H. (1996). Power load management as a computational market. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*.

[Ygge and Akkermans, 1997] Ygge, F. and Akkermans, H. (1997). Making a case for multi-agent systems. In Boman, M. and de Velde, W. V., editors, *Advances in Case-Based Reasoning*, number 1237 in Lecture Notes in Artificial Intelligence, pages 156–176. Springer-Verlag.

[Zacharia, 1999] Zacharia, G. (1999). Collaborative reputation mechanisms for online communities. Master's thesis, Massachusetts Institute of Technology.