Formalizing a language for institutions and norms

Marc Esteva[1] Julian Padget[2] Carles Sierra[1]

[1] Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
08193 Bellaterra, Barcelona, Spain.
marc@iiia.csic.es sierra@iiia.csic.es tel:+34-93-5809570
[2] Department of Computer Science
University of Bath, BATH BA2 7AY, UK
jap@cs.bath.ac.uk tel:+44-1225-826971

Abstract. One source of trust for physical trading systems is their physical assets and simply their presence. A similar baseline does not exist for electronic trading systems, but one way in which it may be possible to create that initial trust is through the abstract notion of an institution, defined in terms of norms [19] and the scenes within which (software) agents may play roles in different trading activities, governed by those norms. We present here a case for institutions in electronic trading, a specification language for institutions (covering norms, performative structure, scenes, roles, etc.) and its semantics and how this may be mapped into formal languages such as process algebra and various forms of logic, so that there is a framework within which norms can be stated and proven.

1 Introduction

Human interaction very often follows conventions, that is, general agreements on language, meaning, and behaviour. By following conventions humans decrease uncertainties about the behaviour of others, reduce conflicts of meaning, create expectations about the outcome of the interaction and simplify the decision process by restricting to a limited set the potential actions that may be taken. These benefits explain why conventions have been so widely used in many aspects of human interaction: trade, law, games, etc.

On some occasions, conventions become foundational and, more importantly, some of them become norms. They establish how interactions of a certain sort will and must be structured within an organization. These conventions, or norms, become therefore the essence of what is understood as human institutions [19]. This is so for instance in the case of auction houses, courts, parliaments or the stock exchange. Human institutions not only structure human interactions but also enforce individual and social behaviour by obliging everybody to act according to the norms.

The benefits obtained in human organizations by following conventions become even more apparent when we move into an electronic world where human interactions are mediated by computer programs, or agents. Conventions seem necessary to avoid conflicts in meaning, to structure interaction protocols, and to limit the action repertoire in a setting where the acting components, the agents, are endowed with limited rationality. The notion of electronic institution becomes thus a natural extension of human

institutions by permitting not only humans but also autonomous agents to interact with one another.

Considering the computer realization of an institution, we have the view that *all* interactions among agents are realized by means of *message interchanges*. Thus, we take a strong dialogical stance in the sense that we understand a multi-agent system as a type of *dialogical system*. The interaction between agents within an institution becomes an illocution exchange. In accordance with the classic understanding of illocutions (e.g. [2] or [23]), illocutions "change the world" by establishing or modifying the *commitments* or *obligations* of the participants. Therefore, formally speaking, an agent will be in this context any entity capable of establishing commitments. This notion becomes the cornerstone of the construction of institutions because otherwise no enforcement or penalty could ever be applied. In a sense, institutions exist because they are the warrants of the satisfaction of the commitments established by the participants.

In this paper we focus on the specification and potential animation of electronic institutions, and specifically on the formal modelling of agent interactions in the framework of electronic institutions. To do so, we explore the application of process algebraic models, using the Ambient calculus [6] (some history of process algebra, justification and citation of earlier work omitted here due to space limitations).

This paper is structured as follows. In section 2 we introduce the background and motivation for our work in electronic institutions, from which we abstracted the components and principles that constitute the institutional description language discussed in section 3, and which we subsequently use to express a modelling of the original FishMarket. Then, in section 4 we sketch a formal basis for the language in the Ambient Calculus and conclude (section 5) with a summary of related work.

2 A model institution

Much of our work since 1995 has taken as a reference point the physical market for auctioning fish in the town of Blanes on the Costa Brava. From this physical model we have abstracted what we call *scenes*, for the admission of buyers, the admission of sellers, the auction room (where a standard downward bidding/Dutch auction format is employed), buyers' settlement, sellers' settlement and a back-room accommodating the accountant, quality assessor and other institutional functionaries. Thus, for each activity that can take place in the institution, there is a corresponding scene, in which interactions between agents are articulated through agent group meetings that follow a well-defined communication protocol—in fact, in our institutions, agents may *only* interact within the context of a scene. This has been described and discussed in detail in [17], while a more general, but also more technical approach appears in [22]. This set of scenes and the connections between them—what roles agents may play in them, how many of each role, to which scenes they may move—constitute the *performative structure* for the electronic institution (see Figure 1). The diagram, as we will explain in subsection 3.2, presents a simplified version of the Fish Market performative structure. The purpose of this diagram is to show the different scenes which comprise the institution by means of a transition graph. Thus, the black circle on the left hand side denotes the start scene and that on the right hand side surrounded by a line is the end scene. In

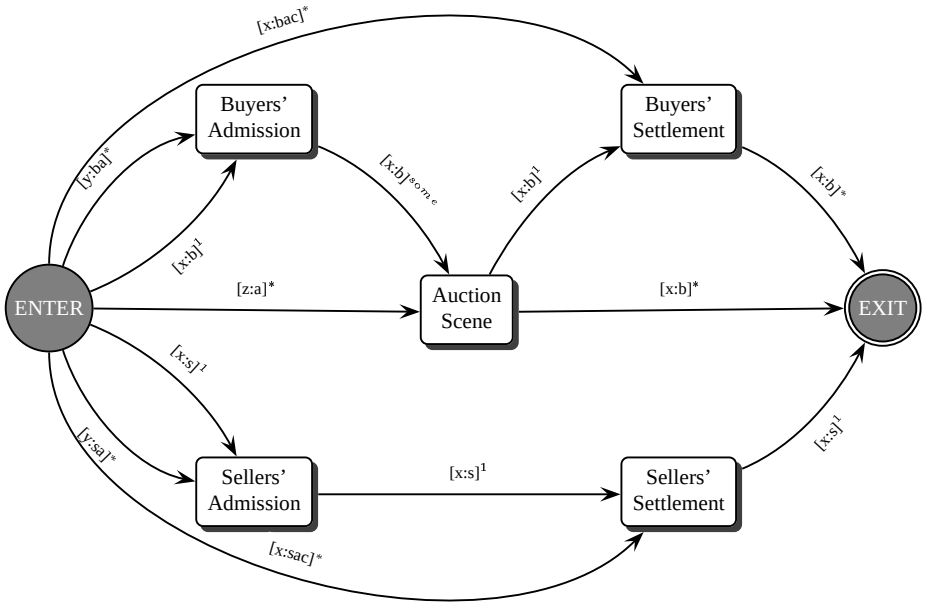

Fig. 1. Performative structure of the Fish Market

between, there are scenes (rectangles with rounded corners) and arcs connecting them. The arcs are labelled with variable:role pairs, where a = auctioneer, b = buyer, ba = buyers' admitter, bac = buyers' accountant, s = seller, sa = sellers' admitter and sac = sellers' accountant. The superscript on the labels indicates whether agents following the arc will start a new scene execution or whether they will join one, some or all active executions of the target scene. A * denotes that a new execution will start, a 1 denotes that agents will join just one execution of the target scene, a *some* denotes that agents will join a subset of the executions of the target scene and *all* denotes that agents will join all the executions of the target scene. For instance, there can be multiple instances of the auction scene, each one initiated by a different auctioneer. Then a buyer can choose to join some of them. It is helpful to know that the system is intended to be initialized by injecting the (staff) agents via the enter node, whence they traverse the performative structure to reach their allotted scenes. Subsequently, buyers and sellers will also enter the market via the enter node and follow the paths assigned to the roles they have adopted.

Within each scene, we use a transition graph labelled with illocutions to define the structure of a conversation and to identify the states at which an agent may join or leave the scene and which agents may say which illocutions (see the Buyers' Settlement scene in Figure 2). The purpose of this diagram is to formalize what an agent may say, which agents may say what, in what order things may be said and at what points a conversation may begin and end (denoted by the access and exit nodes). Each arc is labelled with an illocution, which comprises a particle (in this case, request, accept, deny, inform, or pay), a sender (a variable/role pair), a receiver (idem) and some content. The role

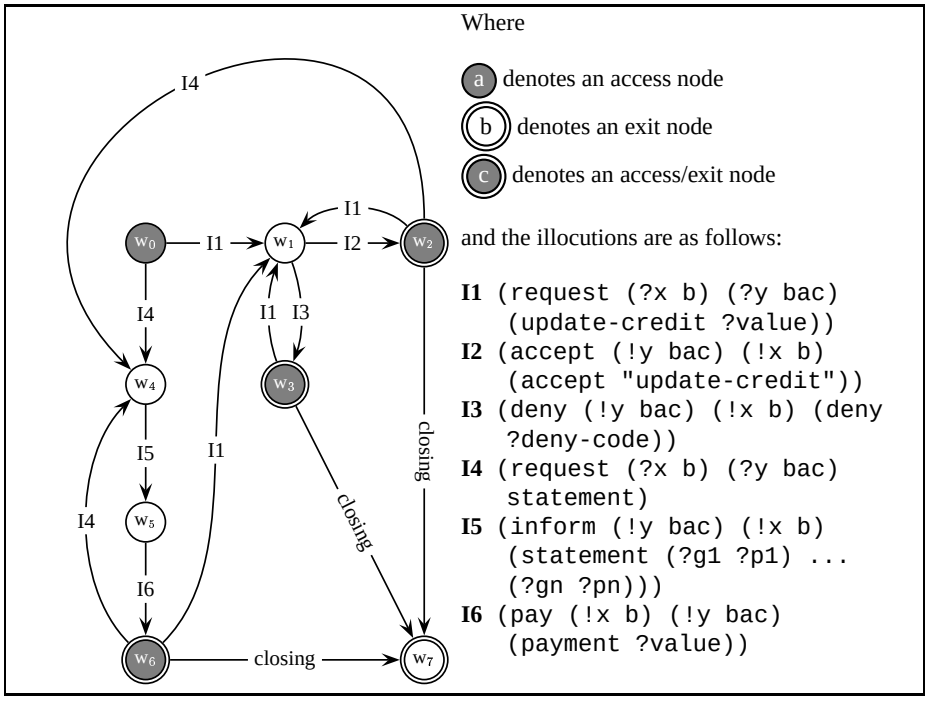

Fig. 2. Conversation graph and illocutions for Buyers' Settlement

identifiers are the same as those given for the performative structure of Figure 1. A fuller explanation of the language of illocutions appears in the next section.

A novel feature of our design is the use of so-called *governor* agents, which mediate between external agents and the institution. These serve several purposes: (i) the governors move around the scenes of the institution on behalf of their external agents because neither do we want to give an external agent potential access to institution internals, nor do we expect that any agent would normally want to put itself in such a position (ii) they may be able to answer questions posed by the external agent about the institution (iii) they shall ensure adherence to the performative structure and the conversation protocol (iv) they may communicate with other governor agents regarding the running of the institution.

3 A language for institutions

We have developed a simple declarative language for specifying the components of our electronic institutions. This language reflects what we have concluded are the key elements in the specification, namely the institution itself, its *performative structure*, the *scenes* making up the performative structure and the *normative rules*. An informal specification of the description language follows here and for the remainder of this section we examine its use in the specification of the (infamous) Fish Market. A complete and

A dialogic framework specification

dialogic-framework: a *name* with which to refer to this framework.
ontology: a *name* referring to a defined ontology (q.v.).
content-language: a *name* defining the content language. It has to be KIF[13], PROLOG or LISP. The intention is to allow for the encoding of the knowledge to be exchanged among agents using the vocabulary offered by the ontology. These propositions are then embedded in the language in accordance with speech act theory [23], by means of the *illocutionary particles*.
illocutionary-particles: a *list* of *names* of illocutionary particles to be used in the illocutions.
external-roles: a *list* of *names* of roles that external agents may play.
internal-roles: a *list* of *names* of roles that internal (staff) agents may play.
social-structure: a *list* of *triples* of two role *names* and the *name* of relationship between them.

Fig. 3. Elements of a dialogic framework

```
(define-dialogic-framework
  fm-dialogic-framework as
  ontology = fm-ontology
  content-language = PROLOG
  illocutionary-particles = (request accept deny inform commit pay)
  external-roles = (buyer seller)
  internal-roles = (boss buyer-admitter seller-admitter
                   auctioneer buyer-accountant seller-acountant)
  social-structure =((boss < buyer-admitter) (boss < seller-admitter)
                   (boss < auctioneer) (boss < buyer-accountant)
                   (boss < seller-accountant) (buyer incompatible seller))
)

(define-dialogic-framework
  buyer-settlement-df as
  ontology = buyer-settlement-ontology
  content-language = PROLOG
  illocutionary-particles = (request inform accept deny pay)
)
```

Fig. 4. The FishMarket and Buyer settlement scene dialogic frameworks

detailed description of the language, called ISLANDER, can be found in[11]. In next subsections we present an informal static semantics for each of the components.

3.1 Ontologic and communicational components: the dialogic framework

The dialogic framework [18], determines the illocutions that can be exchanged between the participants. In order to do so, an ontology is defined that fixes what are the possible values for the concepts in a given domain, e.g goods, participants, roles, locations, time intervals, etc. The elements of the dialogic framework are summarized in Figure 3.

We consider that the communication language *expressions* are constructed as formulae of the type $(\iota\,(\alpha_i : \rho_i)\,(\alpha_j : \rho_j)\,\varphi\;\tau)$ where ι is an *illocutionary particle*, α_i and α_j are terms which can be either agent variables or agent identifiers, ρ_i and ρ_j are terms

which can be either role variables or role identifiers, φ is an expression in the *content language* and τ is a term which can be either a time variable or a time constant.

Two examples of dialogic frameworks appear in Figure 4: (i) for the FishMarket as a whole, noting in particular the illouctionary particles that may be used to tag an illocution, the roles, the role hierarchy and the declaration that an agent may not be a buyer and a seller at the same time. (ii) for the buyer settlement scene, where the particles are restricted to those stated, but the other aspects are inherited from the institutional dialogic framework.

3.2 Social components: scenes and performative structure

We begin by explaining precisely what, for our needs, we regard to be the purpose of a scene. The activities in an electronic institution are the composition of multiple, distinct, 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 communication protocols.

A scene protocol is specified by a directed graph where the nodes represent the different states of the conversation and the directed arcs are labelled with illocution schemes or timeouts. The graph has a single initial state (unreachable once left) and a set of final states representing the different endings of the conversation. There is no arc connecting a final state to some other state.

Normally, the correct evolution of a conversation protocol requires a certain number of agents to be present for each role participating in the protocol. Thus, we define a minimum and a maximum number of agents for each role and this constraint is enforced by the institution. However, because we aim at modelling multi-agent conversations whose set of participants may dynamically vary, scenes allow for agents either joining or leaving at particular points during a conversation. We specify this by identifying, for each role, a set of access and exit states. The final state(s) appear in the exit states of each role, to permit all the agents to leave when the scene is finished. The initial state must be in the set of access states for those roles whose minimum is greater than zero, since this constraint implies that an agent playing that role is required in order to start the scene.

The notion of performative structure is perhaps the most complex since it models the relationships among scenes. In particular, we wish to highlight that although conversations (scenes) are quite widely viewed 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, organizations 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.

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. Likewise, there

A scene specification

roles: a *list* of *names* of role that may participate in the scene.

dialogic-framework: the *name* of the dialogic framework to be used for communication within the scene.

states: a *list* of the *names* of the states of the conversation graph.

initial-state: a *name* identifying the initial state.

final-states: a *list* of *name*s identifying final states.

access-states: a *list* of *pair*s of role *name* and a *list* of states, identifying which roles may join at which states.

exit-states: a *list* of *pairs* of role *name* and a *list* of states, identifying which roles may leave at which states.

agents-per-role: a *list* of *triples* of role *name*, minimum *integer* and maximum *integer*, defining the constraints on the population of a particular role.

connections: a *list* of the transitions between scene states. Each one comprises a preceding state *name*, a succeeding state *name*, and either an *illocution-scheme* with some constraints over scenes variables which must be satisfied to progress through this transition or a timeout that will trigger the transition when will expire.

Fig. 5. Elements of a scene specification

```
(define-scene
  buyer-settlement-scene as
  roles = (buyer buyer-accountant)
  scene-dialogic-framework =  buyer-settlement-df
  states = (w0 w1 w2 w3 w4 w5 w6 w7)
  initial-state = w0
  final-states = (w7)
  access-states = ((buyer (w0 w2 w3 w6))
                   (buyer-accountant (w0)) )
  exit-states = ((buyer (w2 w3 w6 w7)) (buyer-accountant (w7)))
  agents-per-role = ((0 <= buyer <= 1)
                     (1 <= buyer-accountant <= 1) )
  connections = ((w0 w1 buyer-settlement-i1)
                 (w1 w2 buyer-settlement-i2)
                 (w1 w3 buyer-settlement-i3)
                 (w0 w4 buyer-settlement-i4)
                 (w4 w5 buyer-settlement-i5)
                 (w5 w6 buyer-settlement-i6)
                 (w2 w1 buyer-settlement-i1)
                 (w3 w1 buyer-settlement-i1)
                 (w6 w1 buyer-settlement-i1)
                 (w2 w4 buyer-settlement-i4)
                 (w3 w4 buyer-settlement-i4)
                 (w6 w4 buyer-settlement-i4)
                 (w2 w7 closing)
                 (w3 w7 closing)
                 (w6 w7 closing)
                )
)
```

Fig. 6. FishMarket buyer settlement scene

may be multiple concurrent instantiations of a scene, so we must also consider whether the agents following the arcs from one scene to another are allowed to start a new scene execution, whether they can choose to join just one or a subset of the active scenes, or even join all the active scenes. Furthermore, we may associate constraints with each arc, such that and agent must satisfy the constraint in order to traverse the arc. In order to capture the relationship between scenes we use a special type of scenes the so-called transitions. The type of a transition allows to express agents synchronization, choose points where agents can decide which path to follow or parallelisation points where agents are sent to more than one scene. Transitions can be seen as a type of routers in the context of a performative structure.

From a structural point of view, performative structures' specifications must be regarded as networks of scenes. The connections among the scenes defines which agents depending on their role can move from one scene to other(s) through the defined transitions. In other words, the performative structure defines which scenes can be reached by each one of the different roles.

In Figure 6 we give a concrete example of a scene specification, that shows the buyer settlement scene (thus corresponding to the pictorial presentation of the conversation graph in Figure 2). The dialogic framework for this scene was given earlier (Figure 4), while the remainder of the scene specification is effectively a textual description of the conversation graph complete with illocution labels.

At a higher level, Figure 8 shows the declaration of the scenes comprising the Fish-Market and the relationships between them. In effect, this is the textual counterpart to the diagram in Figure 1. In the diagram transitions are omitted for simplicity because there are no synchronisation or choice points, and only one agent can progress at each time through each one of the arcs. Also the connections that will allow staff agents to leave are omitted. This is, a path that connects each scene with the exit scene labelled with the role of the satff agent in charge of it. The textual specification also expresses some constraints not present in the diagram, such as limits on the number of instances of a particular scene and conditions upon connections (see for example, the connection from auction room to transition t9, which has a condition stating that an agent may only take this arc if it has not acquired the obligation to make a payment).

3.3 Normative rules and institutions

The norms which govern an organization are one of the key sources of trust for potential participants, since they define the commitments, obligations and rights of participating agents. As described so far, the performative structure constrains the behaviour of participating agents at two levels:

1. *intra-scene:* Scene protocols dictate for each agent role within a scene what can be said, by whom, to whom, and when.
2. *inter-scene:* The connections between 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 restrictions on agents attempting to reach a target scene.

These norms are, in effect, local. But, it is the agent's actions *within* a scene that may have consequences that either limit or expand its acting possibilities in subsequent

A performative structure specification

scenes: a *list* comprising a *name* for the scene, the *class* of the scene. If there can be multiple instantiations of a scene, this will be denoted by the word 'list' after the class name.

transitions: a *list* comprising a *name* for the transition and the *class* of the transition.

connections: a *list* containing the connections from scenes to transitions and from transitions to scenes. In the first case the connection is expressed by the source scene *name*, the target transition *name*, a *list* of *pairs* of *agent-variable* and role *name*, and a *list* of constraints that will restrict agents movements. In the second case is expressed by the source transition *name*, the target scene *name*, a *list* of *pairs* of *agent-variable* and role *name*, and a *name* defining if a new execution of the scene will be created or if the agent(s) will go to one, some or all current executions.

initial-scene: the *name* of the initial scene—from one of those given in **scenes**.

final-scene: the *name* of the final scene—from one of those given in **scenes**.

Fig. 7. Elements of the performative structure

```
(define-performative-structure
  fm-performative-structure as
  scenes = ((enter root-scene) (exit output-scene)
            (buyer-admission buyer-admission-scene)
            (seller-admission seller-admission-scene)
            (auction-room auction-room-scene list)
            (buyer-settlement buyer-settlement-scene)
            (seller-settlement seller-settlement-scene))
  transitions = ((t1 AND-AND)  (t2 AND-AND)  (t3 AND-AND)
                 (t4 AND-AND)  (t5 AND-AND)  (t6 AND-AND)
                 (t7 AND-AND)  (t8 AND-AND)  (t9 AND-AND)
                 (t10 AND-AND) (t11 AND-AND) (t12 AND-AND) (t13 AND-AND) )
  connections =
    ((enter t1 ((x buyer-admitter)))
     (t1 buyer-admission ((x buyer-admitter)) new)
     (enter t2 ((x buyer)))
     (t2 buyer-admission ((x buyer)) one)
     (enter t3 ((x seller-admitter)))
     (t3 seller-admission ((x seller-admitter)) new)
     (enter t4 ((x seller)))
     (t4 seller-admission ((x seller)) one)
     (enter t5 ((x auctioneer)))
     (t5 auction-room ((x auctioneer)) new)
     (enter t6 ((x buyer-accountant)))
     (t6 buyer-settlement ((x buyer-accountant)) new)
     (enter t7 ((x seller-accountant)))
     (t7 seller-settlement ((x seller-accountant)) new)
     (buyer-admission t8 ((x buyer)))
     (t8 auction-room ((x buyer)) some)
     (auction-room t9 ((x buyer)) (not obligued(x,pay,buyer-settlement)))
     (t9 exit ((x buyer)) one)
     (auction-room t10 ((x buyer)))
     (t10 buyer-settlement ((x buyer)) one)
     (buyer-settlement t11 ((x buyer)))
     (t11 exit ((x buyer)) one)
     (seller-admission t12 ((x seller)))
     (t12 seller-settlement ((x seller)) one)
     (seller-settlement t13 ((x seller)))
     (t13 exit ((x seller)) one) )
   initial-scene = enter
   final-scene = exit
)
```

Fig. 8. Performative structure for the FishMarket

scenes. The consequences we have identified take two different forms. Some actions create commitments for future actions, which may be interpreted as obligations. Other actions may affect the paths an agent may take through the performative structure because it may change which constraints are satisfied. Both types of consequences will need to be observed and maintained by an institution on a per agent basis.

For instance, a trading agent winning a bidding round within an auction house is obliged subsequently to pay for the acquired good. Considering the performative structure in Figure 1 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 fulfill their pending payments. From this example, we can deduce that norms must define the actions that will provoke the activation of the norm, the obligations that agents will have and the actions that agents must carry out in order to fulfill the obligations.

As we are specifying dialogical institutions, agents actions are expressed as a pair of illocution scheme and scene where it is uttered. We need both components because the same illocution could appear in more than one scene. The scene gives the context in which the illocution must be interpreted and of course, this affects the consequences that the utterance of the illocution has. That is to say, the same illocution may have different consequences in different scenes because it is uttered in a different context.

As we have said some of the terms of an illocution scheme are variables. The activation of a norm may depend on the values of these variables in the uttered illuctions. For instance, we specify a norm that says that if a buyer submits a bid which exceeds his credit limit, then the auctioneer is obliged to sanction him (see the `sanction` norm in Figure 10). Each time a buyer submits a bid the value of his bid and the level of his current credit determines whether the norm is activated and whether the auctioneer has to sanction him. These restrictions are specified as boolean expressions over illocution scheme variables and a norm will not be activated if they are not satisfied.

In order to represent the deontic notion of obligation (see [26] for background details) we set out the predicate *Obl* as follows:

$$Obl(x, \psi, s) = \text{agent } x \text{ is obliged to do } \psi \text{ in scene } s.$$

where ψ is taken to be an illocution scheme. We denote the set of obligations by Obl and any concrete obligation by $obl_i \in Obl$. Next we introduce some specific rules, the so-called *normative rules*, to capture which agent actions (illocutions) have consequences that need to be recorded. Given a performative structure and a metalanguage, the normative rules will have the following schema:

$$\begin{aligned}
(s_1, \gamma_1) \wedge \ldots \wedge (s_m, \gamma_m) \wedge & \quad \text{LHS part 1: scene/illocution scheme pairs} \\
e_1 \wedge \ldots \wedge e_n \wedge & \quad \text{LHS part 2: } \begin{cases} \text{boolean expressions over} \\ \text{illocution scheme variables} \end{cases} \\
\neg(s_{m+1}, \gamma_{m+1}) \wedge \ldots \wedge \neg(s_{m+n}, \gamma_{m+n}) & \quad \text{LHS part 3: } \begin{cases} \text{negated scene/illocution} \\ \text{scheme pairs} \end{cases} \\
\rightarrow obl_1 \wedge \ldots \wedge obl_p & \quad \text{RHS: obligations that hold}
\end{aligned}$$

A norm specification

antecedent: a *list* comprising an arbitrary number of *pairs* of scene *name* and illocution-scheme *name* and a *list* of boolean expressions over illocution scheme variables.

defeasible-antecedent: a *list* comprising an arbitrary number of *pairs* of scene *name* and illocution-scheme *name*.

consequent: a *list* of obl *predicate*s.

Fig. 9. Elements of a norm specification

```
(define-norm obligation2pay as
  antecedent =
  ((auction-room
    (inform (?y auctioneer) (?x buyer) (sold(?good-id ?price ?x)))))
  defeasible-antecedent =
  ((buyer-settlement (inform (!x buyer) (?y buyer-accountant)
     (payment(!price)))))
  consequent =
  ((obl !x (inform (!x buyer) (?y buyer-accountant)
                   (payment(!price))) buyer-settlement))
)

(define-norm sanction as
  antecedent =
  ((auction-room
    (commit (?x buyer) (?y auctioneer) (bid(?good-id ?price))))
   (< (credit !x) !price))
  defeasible =
  ((auction-room (inform (!y auctioneer) buyer (sanction(!x)))))
  consequent =
  ((obl !y (inform (!y auctioneer) buyer (sanction(!x)))))
)
```

Fig. 10. The `obligation2pay` and `sanction` norms

where $\neg$ marks a defeasible negation. The meaning of these rules is that if the illocutions $(s_1, \gamma_1), \ldots, (s_m, \gamma_m)$ have been uttered, the expressions $e_1, \ldots, e_n$ are satisfied and the illocutions $(s_{m+1}, \gamma_{m+1}), \ldots, (s_{m+n}, \gamma_{m+n})$ have *not* been uttered, the obligations $obl_1, \ldots, obl_p$ hold. Therefore, the rules have two components, the first one is the causing of the obligations to be activated (for instance winning an auction round by saying "mine" in a downwards bidding protocol, generating the obligation to pay), comprising parts 1 and 2 of the left hand side and the second is part 3 of the left hand side that defeats the obligations (for instance, paying the amount of money due for the round which was won).

Clearly, an external agent may not fulfill its obligations. As agents are autonomous and the institution accepts agents developed by other people, those agents cannot be forced to utter particular illocutions. Thus, it follows that the institutions cannot force agents to fulfill their obligations. However, the institution does know the obligations that each agent has acquired and can thus detect when an agent does not fulfill its obligations and hence violates the norms. Moreover, the institution can restrict the actions that an agent can carry out while it has not fulfilled some or all of its obligations.

As we can see in Figure 9 norms are specified in the following form: the actions that provoke the activation of the norm and the restrictions over illocution scheme variables

An institution specification

dialogic-framework: a *name* of a dialogic framework
performative-structure: a *name* of a performative structure.
norms: a *list* of *name*s of norms.

Fig. 11. Elements of an institution specification

```
(define-institution
  fish-market as
  dialogic-framework = fm-dialogic-framework
  performative-structure = fm-performative-structure
  norms = (obligation2pay sanction)
)
```

Fig. 12. The FishMarket institution

expressed in the antecedent, the actions that agents must carry out in order to fulfill the obligations expressed in the defeasible antecedent, and the set of obligations expressed on the consequent. The antecedent defines the set of illocutions that when uttered in the corresponding scene satisfying the boolean expressions will trigger the norm making the set of obligations expressed in the consequent hold. The defeasible antecent defines the illocutions that agent must utter in the defined scenes in order to fulfill the obligations.

Finally, we are in a position to combine a performative structure, a dialogic framework and a set of norms to construct an institution (see Figure 11 and 12).

4 Grounding the language

Up to this point in the paper, we have presented an attempt at a formal description of an electronic institution and illustrated the use of that language with the relatively well-known FishMarket system. However, although we have described the static semantics of the language in informal terms, the specification is still abstract in computational terms. We see process algebra as a key intermediate level, which can formalize notions of place and action, sitting between our institution language and an actual executable form of the institution and which can establish a verifiable link between the different levels of abstraction. We also observe that the type systems and logics which abstract from the ambient calculus offer suitable frameworks in some cases for expressing low-level norms of limited scope, such as mobility, message types and message orderings (this view is expanded upon in [20]). Thus, we have four objectives in trying to use process algebra with locations to formalize the specification of institutions:

(i) To obtain a precise description of the components of the institution and their interactions.
(ii) To provide a formal framework within which the design can be verified.
(iii) To provide a formal framework within which the design can be validated against standard correctness requirements for distributed systems.
(iv) To provide a formal framework within which the design can be animated to verify institutional norms.

However, it is the first and last issues that hold the most interest for us: the first, because without it we can do nothing and the last because we see norms as the key to generating reputation for and trust in electronic trading institutions. We will now set out how we can relate the components of our institutional specification to the elements of process algebra with locations.

The two axes of the Ambient calculus (AC) are *communication* and *mobility*. The fundamental unit of AC is the notion of an ambient, which is a place within which processes may interact by writing messages into the ambient and reading them from it—hence reading and writing are decoupled and asynchronous. It is not possible for processes in different ambients to interact. Thus communication is a localized activity and it is only via movement that two processes in different ambients can arrive in the same ambient and hence interact. The unit of mobility is the ambient—not individual processes, but rather, a collection of processes and the messages that may be in transit between them. A process within an ambient may execute a capability to (make the whole ambient) enter a sibling ambient or move out of its enclosing (parent) ambient—these are called objective moves, because it is the ambient that moves itself—or an ambient may be dissolved, unleashing its constituents into the enclosing ambient—a subjective operation, because a process executes a capability in one ambient to carry out the operation on another. The effect of each of these operations is conveniently imagined as reorganizations of a tree (see Figure 15), where `in` detaches the sub-tree rooted at the moving ambient and re-attaches it as a child of the target ambient, `out` detaches as for enter and re-attaches it as a child of the parent of the parent and `open` attaches all the children of the subject ambient as children of the ambient performing the open. A sequence of mobility operations is called a path or a capability and may be passed as a message from one process to another. Attempting a move, when the target ambient is not present (i.e. as sibling or parent) causes the process executing that operation to block until the named ambient appears. However, other processes may continue to execute and the ambient itself may still undergo other objective or subjective moves.

From the above brief sketch of (untyped) AC, there is an attractive mapping of scenes to (immobile) ambients, where the conversation can take place via messages, and of agents to (mobile) ambients, which move from scene to scene given the appropriate capability. The situation improves further when we see what work has been started on type systems for AC, since these provide the basis for verifying some forms of norm. Exchange types [8], permit us to specify what may be read and written within an ambient, allowing us to specify which illocutions may be exchanged. An extension of this system [1], which as well as formalizing the idea of polymorphic exchange types, develops the notion of orderly communication being a sequence of types where the sequence relates to the passage of time. In our context, this may correspond to the progression of the conversation. Building on exchange types, Cardelli *et al.* [7], tackled the issue of describing the mobility of ambients, which for us means we can declare, for instance, that the various scenes are immobile (not necessarily always desirable: it may be preferable for an instance of, say, a bilateral negotiation scene to go to the two participants and then permit them to enter) and which ambients (scenes) the various agents may cross (enter/exit). However, while types may be adequate for some relatively

P ::=	$\mathbf{0}$	inactivity	α ::=	x	variable (read)
|	$P \mid P$	composition	|	n	name (new)
|	$!P$	replication	|	$in\ \alpha$	enter α
|	$(\nu\ x_1, \ldots, x_n)\ P$	restriction	|	$\overline{in}\ \alpha$	allow enter α
|	$\alpha\ .\ P$	action	|	$out\ \alpha$	exit α
|	$n[P]$	ambient	|	$\overline{out}\ \alpha$	allow exit α
|	(x)	bind input	|	$open\ \alpha$	open α
|	$\langle\alpha\rangle$	output	|	$\overline{open}\ \alpha$	allow open α
			|	ϵ	empty path
			|	$\alpha\ .\ \alpha'$	path

Fig. 13. Ambient calculus syntax from [8] with co-actions from [14]

$$n[in\ m\ .\ P \mid Q] \mid m[\overline{in}\ m\ .\ R] \longrightarrow m[n[P \mid Q] \mid R] \quad \text{(in)}$$
$$m[n[out\ m\ .\ P \mid Q] \mid \overline{out}\ m\ .\ R] \longrightarrow n[P \mid Q] \mid m[R] \quad \text{(out)}$$
$$open\ n\ .\ P \mid n[\overline{out}\ Q] \longrightarrow P \mid Q \quad \text{(open)}$$
$$(x)\ .\ P \mid \langle\alpha\rangle \longrightarrow P\{x, \alpha\} \quad \text{(communication)}$$

Fig. 14. Safe Ambient calculus reduction rules

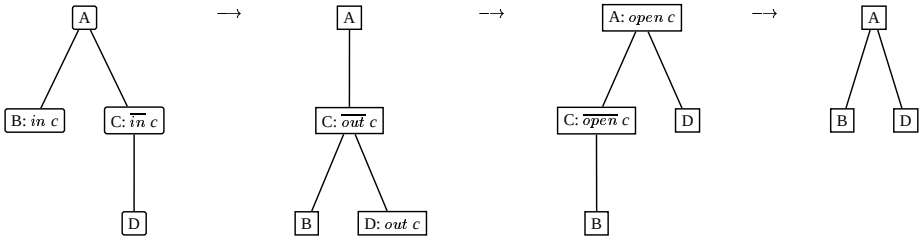

Fig. 15. Ambient calculus mobility operations

simple security norms—who goes where, says what—they cannot capture the more complex norms, such as the examples given above for the FishMarket, which sit in logic and may have modal, temporal or deontic aspects to them. There has been an initial attempt to develop a modal logic for AC, which would permit properties such as “there is at most one staff agent in each scene”, “every governor eventually reaches the exit scene” to be specified. Also of relevance is the temporal logic model checker developed for PoliS [10]. A fuller discussion of these issues, as well as a consideration of the Seal Calculus [9] appears in [21, 20]. A summary of the syntax of AC appears in Figure 13, from which it will be observed that we have adopted the extension by co-actions, introduced in Safe Ambients [14]. Under this variant of AC, for every action (*in*, *out*, *open*) there is a corresponding co-action ($\overline{in}$, $\overline{out}$, $\overline{open}$) and for any action to succeed, the collaborator in the action (a sibling, child or parent ambient, respectively) must engage in the corresponding co-action, that is both parties to the ambient operation must synchronize.

$$\text{MARKET} \triangleq \text{BUYERS-ADMISSION} \mid \text{SELLERS-ADMISSION} \mid \text{AUCTION-SCENE} \mid \text{BUYERS-SETTLEMENT} \mid \text{BUYERS-SETTLEMENT}$$

Fig. 16. High-level market specification

4.1 Agent communication

Our consideration of communications begins with the following two assumptions: that the external agent will be immobile (or leave an immobile representative/proxy) within the market for the duration of all its transactions and that the market representative—the governor—will move between the scenes of the market, interacting with the staff agents and communicating with the external agent it represents. Thus a high level specification of the market is the composition of the different scenes (see Figure 16)

For each scene, a scene manager (SM) provides the following services:

1. mediation of local one-to-one and broadcast message traffic
2. management of agent transport
3. relay of messages between governors and associated external agents

A complete specification of such a SM is too long for the space available, so instead we restrict ourselves to examining the interface provided in each case and a sketch of the specification, looking at the infrastructure for scenes and their interaction with governors.

As noted above each ambient contains a single anonymous channel, communication is asynchronous and with the use of disjoint sums, that single channel becomes, in effect, multiple named channels, or rather a pool which uses type information to match messages with read requests (a similar but more detailed observation on polymorphism and ambients appears in [1]) and starts to look rather similar to the ideas outlined in SecureSpaces [4]. In spirit however, it is much closer to a classical AI component: the blackboard. Thus, our solution to communication takes that principle and implements a simple blackboard at scene ambient level, through which the sub-ambients communicate using *get* and *put* operations.

Thus, in order to send a message, a governor needs to eject an ambient which can post a message on the SM's blackboard, but since for the purposes of this experiment, we are not using objective moves, it is a matter of creating an ambient named *put* which will subsequently move out of the governor, and be dissolved by the SM, so unleashing the *inform* message for posting on the blackboard. Meanwhile, the SM reads the messages posted on the blackboard. In the case of an *inform* message, it checks to see if the intended recipient is a sub-ambient (i.e. another governor), and if so, constructs a *get* ambient containing the message, which enters the target ambient. The target ambient dissolves the *get*, unleashing the *inform* message, which gets posted on the governor's internal blackboard and is subsequently processed. If the target is not a sub-ambient, the message will somehow be routed to the recipient, assuming it is known within the institution. The *broadcast* message is simply broken into multiple *inform* messages and the *enter* and *exit* messages perform book-keeping for message routing. Thus, the governor engages in the following activities:

SCENEMANAGER $\triangleq$
repeat {
open put
| (inform(m, a, b)) . if $b \in$ *subambients*
then *get*[$\langle$inform(m, a, b)$\rangle$. *in* b]
else route message m from a to b
| (enter(a)) . add a to routing tables
| (exit(a)) . delete a from routing tables . *get*[$\langle$leave(*out self*)$\rangle$. *in* a]
| (broadcast(m, a)) . $\forall b : (b \in$ *subambients*$)$ & $(a \neq b), \langle$inform(m, a, b)$\rangle$
}

GOVERNOR $\triangleq$
repeat {
open get
| (inform(m, a)) . process inform
| (leave(*move*)) . prepare to move . *move*
| other agent services
}

msgs = *inform* + *enter* + *exit* + *leave* + *broadcast*

put/get : *messages*$^\frown\varnothing[^\frown\{$*governors*, *staff*$\}, ^\circ\varnothing,$ *msgs*$]$

GOVERNOR : *governors*$^\frown\varnothing[^\frown\{$*scenes*$\}, ^\circ$*messages*, *inform* + *leave* + others$]$

ADMISSION : *scene*$^\frown\varnothing[^\frown\varnothing, ^\circ$*messages*, *msgs*$]$

Fig. 17. Ambient scene manager and governor

TRANSPORT(*route passenger*) $\triangleq$ *go*$[\overline{in}$ *go* . *route* . $\langle$enter(*passenger*)$\rangle$. $\overline{open}$ *go*$]$

TRANSPORT : *transporters*$^\frown\varnothing[^\frown\{$*scenes*$\}, ^\circ\varnothing, \varnothing]$

Fig. 18. Transport agents

- *put*[$\langle$inform(m, TO, *self*)$\rangle$. *out self*] puts a message on to the blackboard for the agent TO. The SM will forward the message by wrapping it in a *get* ambient.
- *open get* is used by an agent to open ambients from the SM's blackboard in order subsequently to be able to receive inform and exit messages.
- *put*[$\langle$enter(*self*)$\rangle$. *out self*] informs the SM of an agent's presence (since moves are subjective in AC, the SM must be informed explicitly of an agent's presence, otherwise it will not receive any messages, but see also the next section).
- *put*[$\langle$exit(*self*)$\rangle$. *out self*] informs the SM that an agent would like to depart and the reply will include a capability to exit the scene.

Following the typing conventions presented in [7], where a type is effectively a quadruple, defining the group of ambients that can be crossed (meaning can be entered or exited) objectively (see earlier discussion of objective versus subjective), those that can be crossed subjectively, those that can be opened and the types of messages that can be exchanged, we can assert types (see Figure 17) for (i) the blackboard messages (*put* and *get*), to say they do not undergo objective moves, but do cross governors and staff, open nothing and exchange messages of type *msgs*, where we have additionally defined *msgs* as the sum of *inform*, *enter*, *exit*, *leave* and *broadcast* (ii) for the governor

to say they do not undergo objective moves, but do cross scenes, open messages and exchange messages of type *msgs* and other (unspecified) internal types and (iii) for the scenes to say they do not undergo objective moves, do not cross anything (i.e. they are immobile), but do open messages and exchange messages of type *msgs*. Thus, some very basic norms can be verified by use of a type system.

4.2 Agent mobility

In the scene manager specification, a governor is only able to move after it has sent an `exit` message to the scene manager, in reply to which it should receive a `leave` message, containing a capability to move out of the scene. Although this enforces some coordination between scene manager and governor, it is generally undesirable to issue capabilities freely, since they can be communicated and re-used. In consequence we have adopted a novel solution involving specialized transport ambients which are created on demand with fixed destinations, issue a capability for the agent that wants to move to enter it, and then dissolve the transport ambient upon arrival at its destination, after also generating a registration message for the new scene manager.

Synchronization is necessary at both ends of the journey, because the transporter should only start to move once the passenger ambient has entered it, furthermore, it should only be dissolved once it has arrived at the destination. This is elegantly resolved by using co-actions, and so we can specify synchronization on the arrival of the passenger ($\overline{in}$ *go*), then follow the path specified in the *route* capability, and finally synchronize on the dissolving of the ambient on arrival at the destination ($\overline{open}$ *go*)—see Figure 18.

5 Related work

Recently there is a growing interest in incorporating organizational concepts into multi-agent systems, with the purpose of considering organization-centered designs of multi-agent systems. For instance, in [25] we find a methodology for agent-oriented design and analysis founded on the view of a system as a computational organization consisting of various interacting roles, although the notion of organization structure is only implicitly defined. Due to the complexity of the design and development of multi-agent systems some languages have been proposed for modelling agent based organizations [3] and [15]. The first presents a language covering the definition of organizations, roles, agents, obligations, goals, and conversations as well as a system for inferring and executing coordinated agent behaviours in multi-agent applications. Although supporting the definition of organizations, this approach exhibits a definite agent-centred view and, for example, negotiations among agents assume that an agent can let others know about its list of obligations and inter-dictions. Moss et al. [15] developed SDML (Strictly Declarative Modeling Language), a multi-agent object-oriented language for modelling organizations which is theory-neutral with respect to the capabilities of agents and, furthermore, includes a library for alternate agent architectures. One of its most distinctive features, from our point of view, is that within organizations there are predefined

linkages among agents and predefined roles in which knowledge is embedded with the purpose of constraining behaviour.

In respect of process algebra, we believe there is no related work on its application to the kind of modelling we have presented here. However, in relation to the work cited above on type systems and static analysis for process algebra, the main omission is an assessment of [16] on the use of Flow Logic to discover or establish properties of process algebraic specifications from static analysis. We hope to remedy this in the near future. In terms of a practical realization of process algebraic models, there are three candidates, namely Ambients on top of JOCAML [12], the JavaSeal kernel [5] and Nomadic Pict [24], all of which are relatively experimental in nature. Tools for type systems and logics for process algebra are also few, largely due to the diversity of approaches currently under exploration, although the temporal logic system of [10] seems promising.

Acknowledgements

Julian Padget was partially supported by a European Study Visit award from the Royal Society and the Consejo Superior de Investigaciones Cientificas (CSIC). Marc Esteva enjoys the CIRIT doctoral scholarship 1999FI-00012. The research reported in this paper is supported by the the SLIE project IST-1999-10948 and the Spanish CICYT project eINSTITUTOR (TIC2000-1414).

References

1. Torben Amtoft, Assaf Kfoury, and Santiago Pericas-Geertsen. What are polymorphically-typed ambients? In David Sands, editor, *Programming Languages and Systems, 10th European Symposium on Programming, ESOP 2001*, volume 2028 of *Lecture Notes in Computer Science*, pages 206–220. Springer Verlag, 2001.
2. J. L. Austin. *How to Do Things With Words*. Oxford University Press, 1962.
3. Mihai Barbuceanu, Tom Gray, and Serge Mankovski. Coordinating with obligations. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, pages 62–69, 1998.
4. C. Bryce, M. Oriol, and J. Vitek. A Coordination Model for Agents Based on Secure Spaces. In P. Ciancarini and A. Wolf, editors, *Proc. 3rd Int. Conf. on Coordination Models and Languages*, volume 1594 of *Lecture Notes in Computer Science*, pages 4–20, Amsterdam, Netherlands, April 1999. Springer-Verlag, Berlin. revised into Coordinating Processes with Secure Spaces and to appear in Science of Computer Programming (Autumn 2001).
5. Ciaran Bryce and Jan Vitek. The JavaSeal mobile agent kernel. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA, October 1999.
6. L. Cardelli. Mobile Ambient Synchronization. Technical Report SRC Tech Note 1997-013, Digital, July 1997.
7. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, volume 1872 of *LNCS*, pages 333–347. IFIP, Springer, August 2000.

8. Luca Cardelli and Andrew D. Gordon. Types for mobile ambients. In ACM, editor, *POPL '99. Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, ACM SIGPLAN Notices, pages 79–92, New York, NY, USA, 1999. ACM Press.
9. G. Castagna and J. Vitek. Seal: A framework for secure mobile computations. In H. Bal, B. Belkhouche, and L. Cardelli, editors, *Internet Programming Languages*, number 1686 in LNCS, pages 47–77. Springer, 1999.
10. P. Ciancarini, F. Franzè, and C. Mascolo. Using a Coordination Language to Specify and Analyze Systems Containing Mobile Components. *ACM Transactions on Software Engineering and Methodology*, 9(2):167–198, 2000.
11. M. Esteva and C. Sierra. Islander1.0 language definition. Technical report, IIIA-CSIC, 2001.
12. Cédric Fournet, Jean-Jacques Lévy, and Alain Schmitt. An asynchronous distributed implementation fo mobile ambients. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, volume 1872 of *LNCS*, pages 348–364. IFIP, Springer, August 2000.
13. Michael R. Genesereth and Richard E. Fikes. Knowldege interchange format version 3.0 reference manual. Technical Report Report Logic–92–1, Logic Group, Computer Science Department, Standford University, June 1992.
14. Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *Conference Record of POPL'00: The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 352–364, Boston, Massachusetts, January 19–21, 2000.
15. S. Moss and B. Edmonds. A formal preference-state model with qualitative market judgements. *Omega – the International Journal of Management Science*, 25(2):155–169, 1997.
16. Flemming Nielson and Hanne Riis Nielson. Shape analysis for mobile ambients. In *Conference Record of POPL'00: The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, Jan 2000. ACM, ACM Press.
17. P Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997.
18. Pablo Noriega and Carles Sierra. Towards layered dialogical agents. In *Third International Workshop on Agent Theories, Architectures, and Languages, ATAL-96*, 1996.
19. Douglass C. North. *Institutions, Institutional Change and Economic Performance*. Cambridge University Press, 1991.
20. Julian Padget. Modelling simple market structures in process algebras with locations. *Artificial Intelligence and Simulation of Behaviour Journal*, 1(1):87–108, 2001. to appear.
21. Julian Padget. Modelling simple market structures in process algebras with locations. In Luc Moreau, editor, *AISB'01 Symposium on Software Mobility and Adaptive Behaviour*, pages 1–9. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour, AISB, 2001. ISBN 1 902956 22 1.
22. J-A. Rodríguez. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona, July 2001.
23. J. R. Searle. *Speech acts*. Cambridge U.P., 1969.
24. Pawel Wojciechowski and Peter Sewell. Nomadic pict: Language and infrastructure design for mobile agents. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA, October 1999.
25. Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (AGENTS'99)*, May 1999.
26. G. H. Wright. Deontic logic. *Mind*, 60:1–15, 1951.