Communicating open systems

Artificial Intelligence 186 (2012) 38–94

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Communicating open systems

Mark d'Inverno [a], Michael Luck [b], Pablo Noriega [c], Juan A. Rodriguez-Aguilar [c,*], Carles Sierra [c]

[a] Department of Computing, Goldsmiths, University of London, London SE14 6NW, United Kingdom
[b] Department of Informatics, King's College London, Strand, London WC2R 2LS, United Kingdom
[c] Artificial Intelligence Research Institute, IIIA-CSIC, Spanish National Research Council, Campus de la Universitat Autònoma de Barcelona, 08193 Bellaterra, Catalonia, Spain

ARTICLE INFO

Article history:
Received 5 July 2011
Received in revised form 8 March 2012
Accepted 11 March 2012
Available online 13 March 2012

We dedicate this work to the memory of our friend and colleague, Marc Esteva, who sadly passed away in December 2011[1]

Keywords:
Open systems
Agent communication
Formal languages
Electronic institutions

ABSTRACT

Just as conventional institutions are organisational structures for coordinating the activities of multiple interacting individuals, electronic institutions provide a computational analogue for coordinating the activities of multiple interacting software agents. In this paper, we argue that open multi-agent systems can be effectively designed and implemented as electronic institutions, for which we provide a comprehensive computational model. More specifically, the paper provides an operational semantics for electronic institutions, specifying the essential data structures, the state representation and the key operations necessary to implement them. We specify the agent workflow structure that is the core component of such electronic institutions and particular instantiations of knowledge representation languages that support the institutional model. In so doing, we provide the first formal account of the electronic institution concept in a rigorous and unambiguous way.

1. Introduction

Open systems [61], in which the various constituent components are unknown in advance and can change over time, are increasingly becoming a *de facto* model for computing. Not only do they reflect the need for interconnection and interaction that are required by modern information systems, they also underpin several visions of future computing systems that span grid computing [52], ambient intelligence [92] and the semantic web [12], as well as many others. Such systems are characterised by: decentralised control, avoiding the bottleneck of a centralised decision-maker; concurrency, by which the different components operate simultaneously with others; and loose coupling, with no component having access to the internal state or structure of others.

At the same time, multi-agent systems have emerged as a promising approach for the development of agile information systems, and are well suited to addressing problems that have multiple problem-solving methods, multiple perspectives or multiple problem-solving entities [66]. In addition to inheriting the traditional advantages of distributed problem-solving, multi-agent systems are based on the exploitation of numerous varieties of sophisticated patterns of interaction, enabling agents to engage in many distinct forms of behaviour. For example, agents may cooperate to achieve a common goal, they

* Corresponding author.
E-mail addresses: dinverno@gold.ac.uk (M. d'Inverno), michael.luck@kcl.ac.uk (M. Luck), pablo@iiia.csic.es (P. Noriega), jar@iiia.csic.es (J.A. Rodriguez-Aguilar), sierra@iiia.csic.es (C. Sierra).

[1] Many of the ideas found in this paper stem from a close and intense collaboration with Marc over the last decade. We have valued his insight and his intellectual contributions to our joint work, to the research community, and to the research domain of multi-agent systems. Most of all, however, we have valued his friendship. The wealth of joyful moments we have shared with Marc over the years will remain constant in our memories. We miss him and will continue to do so.

0004-3702/$ – see front matter
http://dx.doi.org/10.1016/j.artint.2012.03.004

may coordinate their activities in order either to avoid harmful interactions or to exploit beneficial interactions, or they may negotiate agreements. Such varieties of interaction provide the means for multi-agent systems to be highly flexible, in a very different fashion to other forms of software. However, the design and development of such multi-agent systems suffer from all the problems associated with the development of distributed concurrent systems and the additional problems that arise from the kinds of flexible and complex interactions envisaged among autonomous entities [66]. Moreover, the complexity of designing multi-agent systems increases when they are also open systems.

Such open multi-agent systems are populated by heterogeneous agents, and can be considered to be developed by different people using different languages and architectures, representing different parties, and acting to achieve individual goals. Since they are highly complex, costly and may sustain critical applications, it is vital to adopt principled methodologies that support their specification, analysis and validation [66,11]. Indeed, there has been a surge of interest in agent-oriented methodologies and modelling techniques in recent years, motivated and driven both by work on the development of first generation agent systems, which have informed subsequent efforts, and by the need to address the concerns raised in seeking to deliver the visions of future computing systems, as suggested above.

While much work has focused on the micro-level (agent-centred) view, in which the control architecture of individual agents is the key concern, the macro-level (organisation-centred) view of multi-agent systems requires equal attention, particularly in light of the demands for interconnection and interaction. Indeed, there has recently been increasing interest in incorporating organisational concepts into multi-agent systems as well as in shifting from agent-centred to organisation-centred designs [10,34,40,46,78,85,95,100] that treat the organisation as a first-class citizen, similar to the views articulated in pioneering work by Gasser [58] and Pattison et al. [87].

Here, organisations structure the activities of the entities involved, or control the actions of a system as a corporate entity. In this view, a shared organisational structure provides agents with descriptions of 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 set of conventions, or rules of behaviour, that agents are required to follow. Of course, many different types of organisational structure (that specify the roles played by the various agents in the system, their activities, relationships among these activities, and so on) are possible, and providing a means by which these can be specified and constructed is important. One way of providing such organisational structure for open systems is through electronic institutions that provide a computational analogue of conventional institutions.

According to North [80], human interactions themselves are guided by institutions, which represent the rules of the game in a society, including any (formal or informal) kind of constraint that humans devise to shape their interactions. Thus, institutions are the framework within which human interactions take place, specifying what individuals are forbidden from doing and permitted to do, the sanctions that may be imposed if they do not comply, and under what conditions. Human organisations and individuals conform to these institutional rules in order to receive legitimacy and support.[2] In this view, establishing a stable organisational structure for human interactions provides the *raison d'être* of institutions.

The successful adoption of institutions by human societies as a means of structuring and regulating interactions suggests that we might also use institutional notions for structuring and regulating computational interactions, in particular to cope with the complexity of deploying open multi-agent systems. Importantly, this inspiration from human institutions, for use in institutions of humans and agents or even entirely computational agent institutions, in the management and regulation of traditional information systems, for example, demands that we incorporate several key principles. First, the context of any institution must be explicitly taken into consideration as a means of constraining interactions; interactions between individuals are constrained by the prior history that has led them to their current position, and actions can only be situated in that context. Moreover, institutional interactions are persistent in the sense that they are not one-shot computations, but provide the overall institutional context in which other actions can be taken. Actions have such social persistence so that even if an individual agent does not remember what has happened previously, there are consequences for the institution as a whole, and for the individuals within it. In addition, the interactions that underpin the operation of an institution are not necessarily just between two agents, as has often been the restricted case considered in the literature, but instead are collective, involving multiple agents in different roles. Finally, subject to these constraints, each agent is autonomous, and makes individual decisions about actions in light of the context and persistence of the institution. Such decision making impacts on the decisions and actions of others subsequently, and such decision making must therefore be appropriately signalled to others.

In this paper, we argue that open multi-agent systems can be effectively designed and implemented as electronic institutions, for which we provide a comprehensive computational model. The paper makes several distinct contributions, as follows.

First, the paper provides a *reference ontology* for agents engaging in interaction as envisaged here. The paper thus establishes a framework of terminology to describe a range of open systems.

Second, we the paper gives data structures and operations that constitute core *governance mechanisms* to regulate autonomy of agents.

Third, the paper provides a *formal operational specification* for electronic institutions in support of two distinct aims: first, to provide a clear and unambiguous exposition of the concepts underlying electronic institutions; and second, to enable

[2] For instance, *Robert's Rules of Order* (of parliamentary procedure) is a well-known example of an institution.

practical instantiation by those seeking to develop open multi-agent systems. In order to serve these two aims we are careful to separate what is essential to the model – and therefore should be respected in any instantiation – from those aspects that may be implemented in different ways.

Fourth, the paper provides a worked example to show how the formal specification can be used to *design electronic institutions*. More specifically, it provides the apparatus to copy an existing human organisation, to modify one, or to create an entirely new organisation, and to design it in terms of an electronic institution that can meaningfully structure the interactions within open agent systems.

The rest of the paper is organised as follows. Section 2 gives an overview of the concept of electronic institution. Section 3 introduces the languages that are needed to specify any electronic institution and includes illustrative examples of these languages. Section 4 introduces the structural components of an institution, essentially describing what any design must specify before an electronic institution can be executed. All these aspects of electronic institutions are then illustrate in a detailed example of a fish market in Section 5, before proceeding to consider its state and operations on that state. Section 6 specifies the state of an operating electronic institution in terms of its variables during execution. Section 7 provides the corresponding operational semantics in terms of how this state changes over time. Then, Section 8 presents a particular computational architecture that implements the model specified in the paper along with a set of tools for achieving such an implementation. Finally, Section 9 provides an evaluation of the work in this paper, Section 10 reviews related work, and Section 11 concludes.

2. Background

2.1. Institutions

As indicated previously, an institution is an organisational structure for coordinating the activities of multiple interacting agents; it typically embodies some rules that govern these interactions. In seeking to explicate the nature and components of institutions, we use two examples as illustration throughout the paper. The first is the oft-cited Spanish fish market [78,97], which provides a good illustration of the nature of a human institution, with buyers and sellers seeking to engage in interactions aimed at realising the goals of buying and selling fish. Here, there are strict trading conventions by which particular types of fish are traded subject to explicit time and location constraints, and under strict negotiation protocols. More specifically, the fish market is an auction house that enforces certain conditions on: the eligibility of traders (both buyers and sellers); the availability, presentation and delivery of goods; the acceptable behaviour of participants; and the satisfaction of their public commitments. While the actual trading makes up the critical part of the fish market, there are other aspects that must also be considered, and are governed by the market's rules. For example, before any trading can be undertaken, sellers must deliver fish to the market, and buyers must register with the market. Then, once a deal has been agreed, the sellers must pay for and collect the fish, and the buyers must collect payment. The second example relates to a group of friends who meet regularly for particular social activities, such as going to see a movie together. In this group movie scenario, friends coordinate to choose a particular movie, arrange travel to the cinema and a time and place to meet, delegate the task of booking tickets, and so on. Beyond these two specific examples, other institutions have similar sets of distinct activities that can be identified: in hotels, there are activities of making reservations, checking-in and checking-out; in universities, there are activities of lecture and laboratory sessions, registration, examination, and so on.

This suggests that institutions involve a number of distinct activities, or meetings, which are connected together in some form appropriate to the institution. For each of these meetings, agents enter and leave to carry out their respective roles or responsibilities. For example, in the fish market example, there is a clear order to the different activities in which agents may engage, with agents completing some activities before participating in others. In the group movie example, friends agree on which movie to go see before delegating the task of booking tickets. Similarly, in universities, students must register before they have lectures, and only later are examined; and in hotels, reservation precedes check-in, which must be done before checking-out. Now, within a particular meeting, there must be interactions between agents. Typically, these interactions are achieved by means of communication, in making a bid in the fish market, in requesting check-in at the hotel, and so on. In fact, all interactions can be considered as being achieved via communication between agents, in the context of meetings.

2.2. Electronic institutions

Given this view of human institutions we can proceed to examine how computational organisational structures can be understood as analogous electronic institutions. Importantly, our model focuses on the macro-level (societal) aspects of such electronic institutions rather than the micro-level (internal) aspects of agents. In this way we adopt a societal view in accordance with early work in the domain of distributed artificial intelligence (DAI), which followed a societal perspective of agent systems to ease their design and development. In the remainder of this section, we identify the core notions on which our conception of such electronic institutions is founded.

Just as there are meetings in human institutions in which different people interact, electronic institutions have similar structures, known as *scenes*, to facilitate interactions between agents. These scenes are essentially group meetings, with well-defined communication protocols that specify the possible dialogues between agents within these scenes. Importantly, such dialogues are the only means of interaction between agents, and comprise sequences or combinations of individual

illocutions, messages or *utterances*. For example, an electronic fish market might include an auction scene in which buyers compete to purchase fish, with interaction dialogues being specified by auction protocols that involve utterances expressing bids. In this example, it should be clear that there might be many simultaneous instances of such auctions within a fish market. Consequently, rather than specify the communication protocol defining the possible interactions within a scene in terms of specific agents, the protocol is instead *role*-based. In other words, a scene provides a role-based framework of interaction for agents.

As this suggests, scenes within an institution are connected, leading to a *scene network*, with agents moving from one scene to another. In the fish market example, scenes for sellers delivering fish to the market, buyers registering, the auction, and payment are thus connected to form such a scene network. Now, since agents must clearly be able to move between scenes, this scene network establishes how agents can legally move from scene to scene.

It is clear from this discussion that the participants in an electronic institution are agents that interact by the exchange of utterances, similar to humans. However, as indicated above, communication protocols are specified in terms of roles rather than agents. Here, roles can be understood as standardised patterns of behaviour that agents, when instantiating a role, must respect. The identification and regulation of these roles is thus considered as part of the formalisation process of any organisation, and any agent within an electronic institution is required to adopt some roles.

2.3. Electronic institutions for open systems

Electronic institutions are intended to do for open computational systems what conventional institutions do for open human interactions. They therefore provide a particular framework for open systems to support the social activities of agents[3] that are social, decomposable, scalable, local, and dialogical, as we consider below.

Since activities are performed by groups of agents that must coordinate with each other, agents that cannot achieve their goals independently need to interact. In this sense, activities in electronic institutions are social, because agents that interact need to be aware, first, of others and their roles, and second, that certain capacities (or roles) may be required to achieve a particular goal within a common activity.

Activities are decomposable in the sense that the goals of an agent, and the overall goals of a group of agents, can be decomposed into simple activities, or scenes, whose performance achieves the individual and collective goals. This (de)composition requires the interconnection of scenes so that individual and social goals can be achieved through particular combinations of atomic interactions (called utterances) within scenes and the movement of agents between scenes.

Openness means that agents may enter and leave a system at any time, potentially causing a large number of agents to interact. To cope with the scalability that is required as a result of this, not only is problem decomposition needed, but so is the possibility of replicating the enactment of simple activities so that different groups of agents can be allowed to perform the same activity, concurrently, over an enlarging infrastructure. Openness and dynamism cause knowledge about others necessarily to be limited. In consequence, interactions in electronic institutions are naturally local within scenes (and apply to subgroups of agents). This locality of interaction supports scalability by establishing bounds on the interactions of agents, and also supports security and privacy.

Finally, activities are dialogical as they are achieved via agent interactions composed of non-divisible units, utterances, that occur at discrete instants of time. These units can be modelled as point-to-point messages that conform to an interaction mechanism established by the institution, and physical actions are represented by appropriate messages of this form. Consider the game of chess as an institution in this view. The actions involved are typically physical actions of the players to move the pieces around the board. However, they need not be physical actions: in the case of correspondence chess, which is played by various forms of long-distance communication, all that is needed is a means of communicating a move from one player to another. This can be by post, by email, or even by homing pigeon (which has been used!). The key point is that the actions taken are simply articulations of the moves of the players that have an effect on the institution itself just as in our notion of utterances in electronic institutions. Likewise, in an auction, a buyer commits to buy a box of fish at a certain price by claiming so, at an appropriate time, for the auctioneer and all present buyers to hear, while the actual physical action of transferring money from the buyer's bank account to that of the auction house is triggered when the auctioneer declares to everyone that the box is sold to the buyer.[4]

Since the purpose of an electronic institution is to enable this type of social, decomposable, scalable, local and dialogical activity, the electronic institution ought to provide a framework that supports not only atomic interactions between agents (the exchange of utterances), but also those social actions that are needed for the coordination of such social activities, like joining or leaving the institution, creating and terminating scenes, or moving from one scene to another.

[3] Notice that since we are dealing with open systems, we will not be concerned at all with supporting an agent's internal decision making. Thus, decisions regarding when to speak, what to say, which activities to join, or what role to adopt, to name a few, are considered individual decisions that are left to each agent.

[4] Strictly speaking, domain language terms need to be anchored to real world entities, and the electronic institution ought to comply with certain constitutive conventions in order to establish a proper correspondence between brute and institutional facts (see, for example, Searle [98] or Jones and Sergot [67]). Under this consideration, the raising of a hand may amount to a buyer's claim, and the auctioneer's declaration becomes a procedure that activates the appropriate electronic payment.

In summary, an electronic institution is a computational analogue of a conventional institution, with the aim of structuring and facilitating the implementation of the conventions needed for agent interaction to achieve a specific collective endeavour. The formal framework presented in this paper thus includes all the elements needed for an appropriate computational architecture to specify and run any electronic institution. Based on this framework, an engineer can implement an infrastructure that supports social interactions among agents, by choosing the specific languages needed and, using these languages, implementing the data structures that support the relevant institutional constructs – scene, transition, scene network, electronic institution and institutional state. Building on this, our engineer can then provide the utterance-based interaction mechanism to handle atomic agent interactions and, finally, all the operations required for social interaction – those operations that need to be performed by the infrastructure itself to support collective interactions (such as initiating an enactment of an institution), together with those operations whose performance is the outcome of an agent's decision (like speaking or changing roles).

The framework thus provides an institutional environment that is beyond the interpretation of agents, enables actual interactions, enforces conventions and supports awareness. In the remainder of this paper, therefore, we focus on specifying this formal framework. The specification is formal in order to make explicit all details; our adoption of the particular formal notation of Z is simply so that the passage from formal specification to implementation is straightforward. The formalism itself is discussed next.

2.4. Electronic institutions in context

The previous discussion suggests four characterisations of electronic institutions. These can be understood as different ways in which we can interpret what electronic institutions actually are. Each of these characterisations has affinities with work of different communities or disciplines that we outline here, and in some instances elaborate further in Section 10. The characterisations have inspired the development of the model of electronic institutions for which we provide a formal specification.

1. *Mimetic perspective*. Electronic institutions can be characterised as computational environments that mimic the coordination support provided by conventional human institutions.[5]

In this view, the model we formalise in this paper is a type of metamodel (in the sense used to describe UML) to represent the relevant aspects of activities that are supported by the organisation (institutional activities) in which several individuals participate, collectively. The focus of our model is on the social aspects of human institutions such as the roles that individuals play, the relationships among roles, the ontology that pertains to the activity, and to collective activities. Hence, the model is concerned with the social context in which the particular activity takes place, and therefore makes explicit how actions of agents are conditioned and change that context, and how interactions among individuals may affect future interactions. The focus is thus on role flow rather than information flow. Consequently, the abstractions that our model uses as building blocks have a coarser granularity than those addressed in ambient [22] and π-calculus [77], and formalisms like petri-nets and interaction diagrams need to be complemented with other concepts and data structures that are closer to those abstractions of our model. As a metamodel for representing business processes, our work is closer to the view of commitment-based protocols championed by Yolum, Chopra and Singh [24], because of the salience of social constructs and the need for flexibility, standardisation and contextualisation of interactions.

We do not seek to model aspects of a more comprehensive dimension like the strategic organisation of a conventional institution – adequacy to the business context, the efficiency of its departmental structure, incentives or employment policies – nor its performance with respect to global and individual goals. Such issues are more relevant to the work mentioned above [46,85] that takes an organisational view of multi-agent systems. Our work as a descriptive grammar is, in this end of the spectrum, closer to the *activities* & *artefacts* meta-model [81] that describes an interaction environment populated by agents performing activities with the help of passive devices called artefacts. We are even closer to the OMNI model [36] that permits a high level description of organisation activities and the conventions that govern them, and maps the high-level description into an operational version. Likewise, our model shares many features with the proposal of Lopes-Cardoso, Oliveira and Rocha [72,71] as a means to organise the types of virtual organisations inherent in contracting among firms.

The work described in this paper is not just a metamodel but also provides a description that can be used to build systems directly. In particular, one may specify the institution and then use that specification to generate a system that behaves according to the specification. In this respect, our framework is similar to MOISE+ [63], which has a conceptual framework to model organisations that is accompanied by a modelling language (OML in their case, ISLANDER in ours) and an organisational architecture that supports agent interactions (implemented by them as ORA4MAS [62] and as AMELI in our case (see Section 8)). An analogous relation is that of Colombetti and Fornara's Artificial Institutions and their OCeAN metamodel [50].

[5] By conventional institutions, we mean auction houses, hospitals, college fraternities, professional sport leagues, etc. We are aware that we are abusing language when we identify "conventional institution" (conventions) and "organisation" (the entity that incarnates those conventions) and to a further extent with "electronic institution", whose meaning is further merged with the specification of a collective interaction, the code that implements it and the system that ends up running it. Context, and the discussion that follows, should make these distinctions evident.

In this paper, we present a formal definition of the metamodel only, and do not address the means of implementing it. Nevertheless, in Section 8 we give an indication – not a full description – of how the metamodel may be implemented in a particular architecture, and describe an application-independent suite of tools (EIDE) that allow specification of a particular electronic institution, and generate a runtime version of the specification. In the final section of the paper, we also summarise the types of systems that have been implemented with our model and its associated tools. In relation to methodologies, our model does not include agent goals in its conceptual ontology but goal-oriented methodologies for MAS like Prometheus [86] and ours are complementary [101]. This situation is analogous to that of service [3] and organisational oriented methodologies [5].

2. *Governance perspective*. Electronic institutions can be characterised as artefacts that organise collective activities by establishing a restricted virtual environment in which all interactions take place according to some established conventions.

In abstract terms, this is the classical conception of economists and political scientists [80,83,88,98] and arguably also that of Legal Theory [67]. We advocate, as they and others do, a clear distinction between the real and the institutional realities with some connection between the two. We also claim, as they do, that the institutional reality is created by (constitutive) conventions and that inside that institutional reality, behaviour is also subject to conventions (functional and procedural). The essential difference is that we intend those conventions to hold in computational environments. In this respect, our work is akin to the work of Artikis et al. [8].

In more operational terms, this understanding of electronic institutions makes them a form of regulated multi-agent system, and in particular we may see them as "normative multi-agent systems" [14]. What is distinctive about our model is that an electronic institution creates the institutional reality (a virtual world in which only certain things exist and only certain things can happen) and the model provides the means to specify what that reality is, and how the activity of participating agents is articulated (within it). The languages, data structures and operations that we include in our model are the formal tools to express and enforce the conventions that articulate agent interactions. In this paper we do not use a normative language (formal, deontic or otherwise) but we discuss briefly (in Section 10, p. 85) how our formal elements correspond to what the normative multi-agent systems research community understands as norms, and how the model we present here may be extended to explicitly include more normative concepts. Section 10 provides further details and references for the discussion summarised here.

3. *Artefact perspective*. Electronic institutions can be characterised as the operational interface between the subjective decision-making processes of participants and the social task that is achieved through their interactions.

In the second chapter of Sciences of the Artificial [102], Simon refers to the market as such an interface. Here, the market makes it possible for buyers and sellers to exchange goods; the market is not concerned with how individuals decide, and is just there, outside the decision making (mind) of each trader, but enabling the goal of buying or selling Brussels sprouts, by providing each trader with the means to interact in an orderly fashion with other traders. Like Simon's example, our model deals with standard messages, organises them in protocols that support the shared view of interactions, and binds the space of outcomes of each interaction. Again, like Simon, the electronic institution is indifferent to the goals of the individuals and the way in which these individuals reason about their actions. As a pure interface, the electronic institution only establishes some procedural and functional conditions on the interactions of agents and ensures that these conditions hold. What distinguishes our work is that we provide a fully operational model with well-defined computational semantics that permits the implementation of such interfaces.

We share this non-mentalistic view of agent interactions with Singh [104] and Colombetti [27], and in this sense our treatment of scenes is consistent with their commitment-based protocols approach (see Section 10, p. 85). However, we do not profit from the implicit meaning of speech acts that they use to simplify the operationalisation of protocols, combine them and reason about them. Our model does not preclude that form of specification of the interaction conventions, but we have not explored the possibility of extending our model to include operations that handle commitments.

Simon's consideration of a market as an interface also resonates with the connection between electronic institutions and mechanism design. In our case, the electronic institution may be used to specify the mechanism within an experimental setting in which a design is tested or refined through the use of human or software agents [3,90,96]. The purpose of electronic institutions in this context is not the calculation of optimality or equilibria, nor testing a mechanism against, say, market observation [68], but rather the more or less systematic exploration of the space of outcomes to detect unexpected singularities or test alternative definitions [107].

4. *Coordination support perspective*. Electronic institutions can be characterised as a way of providing structure and governance to open multi-agent systems.

Conceptually, the formal model we present in this paper describes the blueprint of an application-independent platform to support the coordinated interactions of open systems. This platform is used – when enabling a specific application – to specify an *institutional environment* in the sense that: (i) it creates the virtual environment in which interactions are enacted;

(ii) it establishes a common set of interaction standards and conditions (scene specifications, transitions between scenes, common domain language, common state variables, etc.) that apply to every interaction that takes place in the environment; and (iii) it enforces those standards and conditions to the point that only what is admissible by the infrastructure may affect the environment. Agents exist independently of the environment, and the environment cannot control their internals, but only filters the actions they attempt within the environment. Because of the institutional character of the platform, only agents that are willing and capable of satisfying the standards and conditions established in the environment may interact within that environment. It is in this very specific sense of openness that electronic institutions support open systems. In real terms, an actual platform that conforms to the formal model needs to be implemented within a computational architecture that is implemented with actual software tools that enable actual interactions among actual agents. In this paper we provide a comprehensive specification of the formal model, discuss one particular architecture and review actual tools that implement it (in Section 8) as mentioned above.

Weyns et al. [110] discuss in depth the notion of environment in the context of a multi-agent system (MAS). They describe three levels of support that the environment may provide: (i) the *basic level* that links a MAS with the "deployment context" (that is, hardware, software and external resources that allow the MAS to function); (ii) the *abstraction level* that sits on top of the deployment context and allows agents to be functional; and (iii) the *interaction-mediation level* that "*offers support to*: (1) *regulate the access to shared resources, and* (2) *mediate interaction between agents*". Electronic institutions belong to the third of these levels. Note that an agent communication support environment, in the sense of FIPA's reference model [47], belongs to the abstraction level, while the institutional framework sits on top of it. In Section 8, we advocate a full separation so that the institution is concerned only with those utterances that reach the institution through some communication support environment (for example *JADE*) and making available to that communication environment those utterances meant to reach an agent beyond the institutional environment.

Our proposal shares elements with other models that could also belong to the interaction mediation level. So it is akin to the views on the articulation of software components proposed by Gasser and Pattison already mentioned [58,87]. Also mentioned above, the role of commitment machines [23] in coordination support is similar in abstraction and functionality to our work. Even closer is the proposal of Robertson and McGinnis of a Light Communication Calculus [76] (see Section 10, p. 86). The latter two approaches deal with agent coordination in a layer that is outside the agents themselves, and introduce languages, data structures and functionalities that are specific for joint activities. Both commitment machines and LCC have the notion of protocol as a primitive but both give particular shades to its meaning, a salient feature being that the space of possible agreements is potentially infinite.

2.5. Notation

There is a large and varied number of formal techniques and languages available to specify properties of software systems [37] including state-based languages such as VDM, Z and B, process-based languages such as CCS and CSP, temporal logics, modal logics and statecharts to name a few. While there are advocates for each of these approaches to modelling various aspects of computer programs, one important objection to the use of formal techniques is that they do not directly relate strongly enough to the construction of software. We therefore choose the Z specification language to deliberately adopt a technique that not only enables the formal specification of systems and languages, but allows for the systematic reduction of such specifications to implementations. Moreover, there is a range of tools available including a syntax and type checker, which we have used in this paper to ensure that our specification is consistent and well defined. For more information on the language, the reader should consult the authoritative Z Ref. [105] or consider examples of its use in specifying agent-based systems [39].

Z is a state-based specification language based on set theory and first order predicate calculus; its key syntactic element is the schema, which allows specifications to be structured into manageable modular components. Z schemas consist of two parts, the upper declarative part, which declares variables and their types, and the lower predicate part, which relates and constrains those variables. Modularity is facilitated in Z by allowing schemas to be included within other schemas, while operations are defined in terms of changes to the state. More specifically, an operation relates variables of the state after the operation (denoted by dashed variables) to the value of the variables before the operation (denoted by undashed variables). Operations may also have inputs (denoted by variables with question marks), outputs (exclamation marks) and a precondition. To introduce a type in Z when no information about the elements within that type are known, a *given set* is used. For example, [*TREE*] represents the set of all trees without saying anything about the nature of the individual elements within the type. Then, if we wish to state that a variable takes on a value, a set of values, or an ordered pair of values of this type, we write $x : TREE$, $x : \mathbb{P} TREE$ and $x : TREE \times TREE$, respectively. A summary of the Z notation used in this paper is provided in Table 1, with further details of the use of the language provided in Appendix A.

3. Languages

Open systems of many different flavours, such as service-oriented systems, multi-agent systems and peer-to-peer (P2P) systems, are predicated on the requirement for meaningful and effective interactions between their component entities. Many aspects of these interactions can be, and are, captured by means of languages representing the domain of interest, constraints that must hold in order for an action to take place, updates to the state when actions take place, and so on,

Table 1
Summary of Z notation used.

Definitions and declarations	
a, b	Identifiers
p, q	Predicates
s, t	Sequences
x, y	Expressions
A, B	Sets
R, S	Relations
$d; e$	Declarations
$a == x$	Abbreviated definition
$[A]$	Given set
$A ::= b\langle\langle B\rangle\rangle \mid c\langle\langle C\rangle\rangle$	Free type declaration
$\mu d \mid P$	Definite description
let $a == x$	Local variable definition
Logic	
$\neg p$	Logical negation
$p \land q$	Logical conjunction
$p \lor q$	Logical disjunction
$p \Rightarrow q$	Logical implication
$p \Leftrightarrow q$	Logical equivalence
$\forall X \bullet q$	Universal quantification
$\exists X \bullet q$	Existential quantification
Sets	
$x \in y$	Set membership
$\{\}$	Empty set
$\mathbb{N}$	Set of natural numbers
$A \subseteq B$	Set inclusion
$\{x, y, \ldots\}$	Set of elements
$(x, y, \ldots)$	Ordered tuple
$A \times B \times \cdots$	Cartesian product
$\mathbb{P}A$	Power set
$\mathbb{P}_1 A$	Non-empty power set
$A \cap B$	Set intersection
$A \cup B$	Set union
$A \setminus B$	Set difference
$\bigcup A$	Generalised union
$\#A$	Size of a finite set
$\{d; e \ldots \mid p \bullet x\}$	Set comprehension
Functions	
$A \nrightarrow B$	Partial function
$A \rightarrow B$	Total function
first	First element of ordered pair
second	Second element of ordered pair
Relations	
$A \leftrightarrow B$	Relation
$\mathrm{dom}\, R$	Relation domain
$\mathrm{ran}\, R$	Relation range
$R^{\sim}$	Relational inverse
$A \lhd R$	Domain restriction
$A \rhd R$	Range restriction
$A \lhd\!\!\!\!- R$	Anti-domain restriction
$A \rhd\!\!\!\!- R$	Anti-range restriction
$R \oplus S$	Relational overriding
$R\star$	Reflexive transitive closure
Sequences	
$\mathrm{seq}\, A$	Sequence
$\mathrm{seq}_1\, A$	Non-empty
$\langle\rangle$	Empty
$\langle x, y, \ldots\rangle$	Sequence
$s^\frown t$	Concatenation
Schema notation	
S / d / — / p	Schema
d / — / p	Axiomatic definition
T / S / d / — / p	Schema inclusion
$z.a$	
θS	
$z.a$	Component
θS	Instance of the schema S
ΔS / S / S'	Operation
$a?$	Input to an operation
a	State component before operation
a'	State component after operation
S	State schema before operation
S'	State schema after operation
ΞS	No change of state

with the differences between the types of system arising from particular choices of the representation languages used. In this section, we focus on the framework established by these languages, within which our electronic institution model is developed. These languages are necessary for the specification, but are not fundamental to the basic computational concept. That is, they are parameters to a core computational component in the same way as an algebraic functor is applied to different implementations of basic structures that satisfy certain input–output restrictions, and can be instantiated to suit the needs of a particular application. The rest of the paper builds on this underpinning by providing a means of specifying stateful communicative workflows that constitute the backbone of electronic institutions.

3.1. Electronic institution languages

Our concern is with specifying open systems in which agent interactions are meaningful, contextual, consequential and regulated. We handle these properties through different languages and constructs as follows, illustrated in Fig. 1, which shows the languages and their mapping to the particular institutional constructs used in the paper.

1. *Domain language.* When interacting within a common environment, agents need to communicate about their problem domain, so that an ontology that describes this domain is required. Such a domain ontology must be specified in terms of a domain language in which to express the purpose and means of the interaction; although this shared language might be the result of some process of semantic alignment, for the interaction to be successful it must be meaningful to all participants. For instance, *fish* is a concept that belongs to the ontology of a fish market, just as *movie* is a concept in the group movie example.

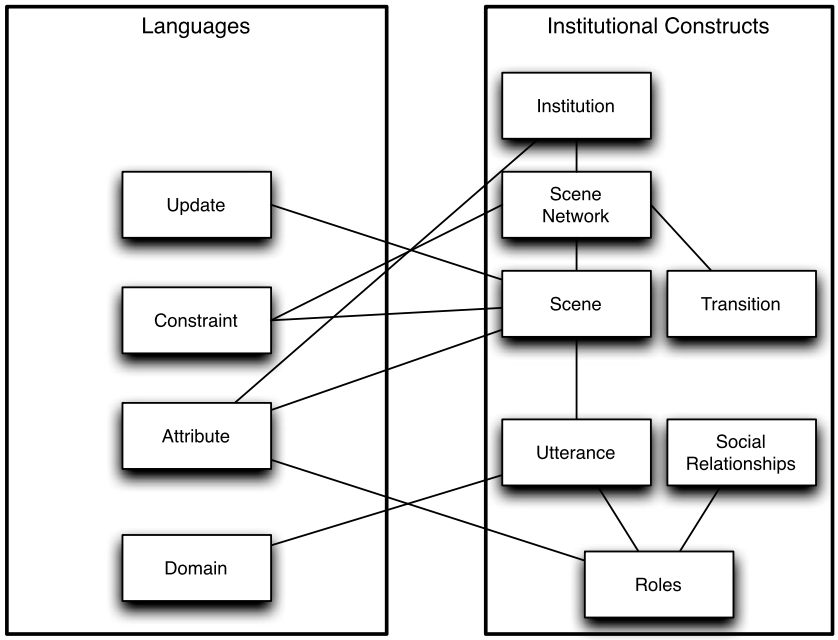

Fig. 1. Tower of languages. Lines show which institutional construct requires which language.

2. *Attribute language*. We need to represent the attributes of the components of the framework (such as for an agent, a role, the institution itself, and so on), for which an attribute language is required. An example here is the credit of an agent taking part in an interaction to buy goods, or the movie of the week selected by group members in the group movie scenario.
3. *Constraint language*. Interactions generate a history of information exchanges – a sequence of messages – that determines the particular context in which new interactions are to be interpreted. Interactions are thus regulated in being constrained by such a context. For this reason, a constraint language is needed to guarantee that every atomic interaction occurs in the right context. For example, any bid in an English auction must be higher in value than a preceding bid. In the group movie scenario, a movie cannot be chosen unless more than 50% of the group members agree on watching it.
4. *Update language*. Changes to the context of an interaction occur as actions are executed when agents speak. Interactions between agents must thus be contextual in the sense that their effects depend on the specific circumstances in which they take place (the history of interactions, which agents were involved in them, and what roles they were playing at the time). The representation of such evolving circumstances requires a stateful architecture in which this context is explicitly represented. Since interactions have consequences for the context in which the participants are engaged, an update language is needed to specify how to update attributes after an interaction. For example, if John wins an auction for a box of fish, his credit (a variable associated with any buyer) is decreased by his winning bid amount. In the group movie scenario, the movie of the week (an attribute associated with a group) is updated with the most preferred movie after all group members have voted for a movie.

To summarise, we need domain, attribute, update and constraint languages, which we outline and specify at the highest level of abstraction next. We eventually use these languages in Section 4.3 to define the notion of *utterance*, the atomic interaction mechanism of electronic institutions.

3.2. Domain language

While ontologies can be very complex data structures, here we use a simple framework that has been shown to cover many practical applications, but which can also be extended further to arbitrary levels of sophistication, as needed. We consider the domain ontology to comprise a set of concepts (such as auctions, fish, buyer, movie, restaurant, etc.), which may have some relationships between them. Thus, in order to allow for agents to exchange information in a domain, a language to structure an ontology that includes a repertoire of elements is needed. The syntactic details of each formula are not relevant for the purpose of this specification in Z since, as mentioned earlier, our specification does not commit to the choice of any particular language.

$[DLFormula]$

$DomainLanguage == \mathbb{P}\, DLFormula$

DLFormula ::= Concept | Concept **ISA** Concept | Relation({Concept}$^+$)
Concept ::= *ConConstant* | Basic
Relation ::= *RelConstant*
Basic ::= **Integer** | **Real** | **String** | **Bool**

Fig. 2. BNF for example domain language.

ALFormula ::= Attribute : Type
Attribute ::= AttributeName [Default] [Current]
Type ::= Concept

Fig. 3. BNF for example attribute language.

An example of a very simple domain language is shown in Fig. 2 as a set of concepts (for example, quantity, quality, material) organised in an *is-a* hierarchy (for example, cod *is-a* fish, Australian-dollar *is-a* currency, "The Godfather" *is-a* movie), and a set of relations over these concepts (such as *price(cod,5 AUD)* or *genre ("The Godfather", thriller)*). Note that terminal symbols are indicated in the figure in bold, and unspecified terminal symbols in italics. Importantly, this domain language provides the basic types that are required as a base for further development.

3.3. Attribute language

Building on the domain language types, the attribute language provides the means to represent attributes of the components in the framework (see Schema 1). Such attributes might indicate that buyers have credit, or that a seller owns something like a boat or boxes of fish. Formally, an attribute expression is a formula from the domain language, and an attribute consists of an *attribute name*, its *range* (which is the range of attribute expressions to which the attribute *value* can be mapped, drawn from the domain language), an optional *current value*, and an optional *default value*.

$[AttributeName]$

$AttributeExpression == DLFormula$

$AttributeExpression \in DomainLanguage$

$$
\begin{array}{|l}
\hline
Attribute \\
label : AttributeName \\
range : \mathbb{P}\, AttributeExpression \\
value : \mathsf{optional}\ [AttributeExpression] \\
default : \mathsf{optional}\ [AttributeExpression] \\
\hline
\end{array}
$$

Schema 1. The definition of an attribute.

A subset of attributes, *required attributes*, must have defined values at all times. In particular, those required attributes that do not have a default value need some value to be provided for them before an electronic institution can be initiated. We call these *user defined attributes*.

$RequiredAttribute == \{a : Attribute \mid \mathsf{defined}\ a.value\}$

$UserDefinedAttribute == \{a : RequiredAttribute \mid \mathsf{undefined}\ a.default\}$

Attributes are employed to model the institutional state of the components of an institution, with a set of attributes being associated with every element of an institution. For consistency of representation across languages, we define an *ALFormula* to be an *Attribute*.

$ALFormula == Attribute$

$AttributeLanguage == \mathbb{P}\, ALFormula$

As an example of a very simple attribute language, Fig. 3 shows the BNF that allows us to instantiate the specification and define sets of $\langle attribute, type \rangle$ pairs, where this type of attribute is any domain type as defined in the domain language. This is illustrated through the use of Concept, inherited from the exemplar domain language. An attribute formula (ALFormula)

$$
\begin{aligned}
\text{CLFormula} &::= \text{Expression BinaryOp Expression}\\
\text{Expression} &::= \text{CLTerm MathOp CLTerm}\\
\text{CLTerm} &::= \mathit{clconstant} \mid \text{CLVarTerm} \mid \text{PropertyName}\\
\text{MathOp} &::= + \mid - \mid / \mid *\\
\text{BinaryOp} &::= == \mid \mathbf{!}= \mid < \mid <= \mid >= \mid >\\
\text{CLVarTerm} &::= \mathbf{!}\mathit{varid} \mid \mathbf{?}\mathit{varid}
\end{aligned}
$$

Fig. 4. BNF for example constraint language.

is then any correctly typed expression recognised by the grammar of Fig. 3, giving a language that can express that credit is of type Integer, and item is of type Fish, for example.

$$\mathit{credit} : \mathit{Integer}$$
$$\mathit{item} : \mathit{Fish}$$

3.4. Constraint language

One of the most powerful representation elements of an electronic institution is the notion of constraints, which can be understood as restrictions on the utterances of agents, on what they can say. For example, any bid in an English auction must be higher in value than a preceding bid. The constraint language is thus a set of formulae, *CLFormula* which, for instance, may consist of logical expressions over the primitive terms specified earlier.

$$[\mathit{CLFormula}]$$
$$\mathit{ConstraintLanguage} == \mathbb{P}\ \mathit{CLFormula}$$

Again, while the syntactic details of each formula are not relevant for the purpose of this specification, one possible syntactic representation of constraints could be the widely used language OCL.[6] A more specific illustrative example of a simple constraint language is given in Fig. 4. This is based on a set of simple relational operators (e.g. $=$, $<$) to access the bindings of variables, where $\mathbf{?}x$ indicates the value matching variable x, for example in an utterance, and $\mathbf{!}x$ indicates the most recent binding of variable x.[7] (A more complete account of variable bindings in electronic institution is provided in Section 6.) A formula in the constraint language can thus express the restriction that a new bid must be at least 5 Euros higher than the highest previous bid, or that the balance of a bidder must be equal to the credit multiplied by the interest. These examples are expressed in the constraint language as follows.

$$\mathbf{?}\mathit{bid} \geqslant \mathbf{!}\mathit{highestbid} + 5\mathit{EUR}$$
$$\mathbf{?}\mathit{Balance} := \mathit{Credit} * \mathit{Interest}$$

In general we can understand a constraint language as providing predicates over attributes that either hold or do not at any specific time.

3.5. Update language

All components of an institution (such as agents, scenes, etc.) are stateful and thus require a way to capture the changing values of attributes. The values of these attributes at any moment in time capture the state of the component; changes to attributes are implemented as *updates* that are executed when agents speak. For example, if *JohnSmith* wins an auction for a box of fish, his credit (a variable in the information model associated with any buyer) is decreased by the amount of his winning bid. An electronic institution thus requires an update language that determines how to update the values of attributes after illocutions occur. For example, this update language can be a set of update language formulae, which consist of constructors on the primitive terms specified in previous examples. As before, the set of update language formulae are

[6] See http://www-st.inf.tu-dresden.de/ocl/.

[7] We recognise that question marks and exclamation marks are also part of the standard Z notation but, because there is a history of work in Z and in electronic institutions, it would not be appropriate to change the notation. Nevertheless, we clarify the distinction further here. We do not use exclamation marks in the Z specification in this paper so there is no conflict. With respect to question marks, we distinguish them as follows: first, while in the Z specification they are used as postfix for an expression, in the electronic institution language they are used as prefix; second, in the electronic institution language, both **!** and **?** are written in bold, but in the Z specification they are not; and third, we present our work in a way that clearly distinguishes the Z notation from the electronic institution language.

ULFormula ::= *Assignment*
Assignment ::= Attribute := *Expression*

Fig. 5. BNF for example update language.

defined as a given set.[8] In general we can understand an update language as being a mapping from attributes to attributes, with new information overwriting the old information.

$$[ULFormula]$$
$$UpdateLanguage == \mathbb{P}\, ULFormula$$

Most commonly, updates assign the result of the evaluation of an expression, defined over the terms of the constraint language, to an attribute (from the attribute language). In the example grammar of Fig. 5, an update formula, ULFormula, is any correctly typed expression allowing the following, for example.

$$Loan := 5000$$
$$Credit := Loan + 2000$$
$$Item := cod$$
$$Fee := !bid * 0.1$$

3.6. The language model

An electronic institution is parameterised by all the languages specified above. Thus, for any institution, we must specify the domain language, attribute language, update language and constraint language. We also specify subsets of the domain and update languages that are ground. This is captured by the *Languages* schema (Schema 2) in which we include explicit line numbering (that is not part of the standard Z notation) to aid the reader. (We adopt such line numbering throughout the rest of the paper.)

Languages

$$domainontology, grounddomainontology : DomainLanguage \quad [1]$$
$$attributelanguage : AttributeLanguage \quad [2]$$
$$updatelanguage, groundupdatelanguage : UpdateLanguage \quad [3]$$
$$constraintlanguage : ConstraintLanguage \quad [4]$$

$$grounddomainontology \subseteq domainontology \quad [5]$$
$$groundupdatelanguage \subseteq updatelanguage \quad [6]$$

Schema 2. The language definitions required in an electronic institution.

Thus, when an electronic institution is defined (in the next section) a specific set of languages is provided to that institution, and we refer to these using the global constant *languages*.

$$| \quad languages : Languages$$

4. Scenes, transitions and institutions

As has been stated previously, an electronic institution is the computational analogue of a human institution in which agents work together in some structured fashion while retaining their autonomous, independent nature. Institutions here are not simple organisations capturing a single set of one-shot interactions, but instead must facilitate more sophisticated combinations of interactions. This is because, for anything more than trivial problems, agents need to engage in multiple activities that are woven together to form a network of connected interactions, with each activity possibly involving different groups of agents. As was illustrated earlier by the fish market auctions, an institution comprises a set of connected activities, known as *scenes*, in which agents playing particular roles interact in support of some particular institutional objective.

The choice of terminology here in drawing inspiration from theatre is deliberate, and is not dissimilar in spirit to the scripts knowledge representation schema. In both cases, the theatre analogy involves scenes that are the activities in which

[8] A possible syntactic representation of updates could be the widely used language for planning PDDL (see http://ipc.informatik.uni-freiburg.de/PddlResources).

agents participate, and roles that agents play within these scenes (like actors in a play). Thus, in the fish market example, agents play the roles of bidders, sellers, auctioneer and auction manager, and various combinations of these roles participate in scenes in which fish are delivered to the market, buyers register with the market, buyers pay for bought fish, buyers collect fish, and sellers collect payment. In each such scene, agents interact with others by speaking their *lines*,[9] or making utterances, just as in plays. Scenes regulate agent interaction, depending on the role each agent plays, so that specifying a scene amounts to specifying the regulation for agent interactions.

An electronic institution is thus a scene network connected together, called a *scene network*, with agents playing their roles within scenes, and moving from one scene to another, possibly to play different roles, at appropriate moments, like a play-script in theatre that brings together the overall play. In addition, there is an order in which scenes must unfold, so that buyers collect their goods only after they have bought them and paid for them, for example. Importantly, since agents in this model are autonomous and decide for themselves whether and when to move from one scene to another, we need some means of coordination so that an agent does not arrive at a scene too early or too late (in theatrical terms so that it *hits its cue for action*), and so that it is able to change role. In order to do this, we introduce the concept of *transitions*, corresponding to time backstage preparing for, and waiting for, the relevant cue.

On this basis, we can set up the fundamental constructs necessary for a formal computational specification of electronic institutions. We begin in this section by introducing the basic building blocks for what follows in relation to the operational semantics of electronic institutions, first with an execution semantics in Section 5, and then with a specification of the operations in Section 6. Since designing an electronic institution amounts to specifying its roles and relationships, scenes and transitions, the section starts by considering these roles, relationships, scenes and transitions, together with utterances.

4.1. Primitives

In order to build up a specification of any system, we need to be able to refer to (i) the entities involved (the agents), (ii) the roles they need to adopt in order to take part in an electronic institution, and (iii) time. For each of these we introduce constants, variables and terms to provide a first order language. Constants are needed to refer to actual entities in the domain (such as a concrete box of fish) and to actual agents participating in the institution (such as a buyer called *JohnSmith*), and variables are needed to write generic expressions throughout an electronic institution specification (for example, any buyer can bid for a concrete box of fish). Special consideration must be given to the modelling of time instants; any communication within an institution takes place at a particular time, and the time of an action is crucial for verifying its correctness (for example, a bid can only take place before an auctioneer concludes an auction) or for certification purposes (for example, my bid was sent before yours).

We can define the set of identifiers for agents and roles as given sets of constants, which means that we simply parachute in their types. In addition, we define time to be a natural number.[10] So, for example, elements of each of these sets in turn might be *JohnSmith*, *Auctioneer*, and *1234567*.

$$[AgentConst, RoleConst]$$

$$TimeConst == \mathbb{N}$$

We can define the set of all constants for agents, roles and time, using a one-to-one mapping from the base types.

$$\begin{aligned} Const ::= \; & aconst\langle\langle AgentConst\rangle\rangle \\ | \; & rconst\langle\langle RoleConst\rangle\rangle \\ | \; & tconst\langle\langle TimeConst\rangle\rangle \end{aligned}$$

In addition, there is one special constant signifying the set of all agents.

$$| \; agentall : AgentConst$$

We also introduce given sets for the symbols used to represent variables.

$$[AgentSymbol, RoleSymbol, TimeSymbol]$$

Now, the entire electronic institution framework requires a distinction between free and bound variables. The intuition is that free variables allow us to describe future events, while bound variables allow us to refer to past events. When one of the potential future events happens, free variables become bound. For instance, when we specify a protocol for an auction, we need to identify generic utterances that: (i) refer to any bidder, so that the variable representing the bidder must be free and only becomes bound when the actual illocution made by a concrete bidder is uttered; and (ii) refer to a concrete bidder that has already uttered previous illocutions. We therefore introduce a prefix signifying bound (concrete) variables, **!**,

[9] Note that to distinguish the *lines* spoken by agents from the lines of a schema, we always italicise the former in this paper.

[10] Without loss of generality, we assume that actions are tagged with Unix time, a system for describing points in time as the number of seconds elapsed since January 1, 1970.

or free (unbound) variables, **?**, followed by the variable symbol. The set of bound variables is then defined simply as those pairs whose first element is the bound symbol, and free variables are those whose first element is the free symbol.

$$Prefix ::= \mathbf{!} | \mathbf{?}$$

$$\begin{aligned}
AgentVar &== Prefix \times AgentSymbol \\
RoleVar &== Prefix \times RoleSymbol \\
TimeVar &== Prefix \times TimeSymbol \\
BoundAgentVar &== \{a : AgentVar \mid first\ a = \mathbf{!}\} \\
BoundRoleVar &== \{a : RoleVar \mid first\ a = \mathbf{!}\} \\
BoundTimeVar &== \{a : TimeVar \mid first\ a = \mathbf{!}\} \\
FreeAgentVar &== \{a : AgentVar \mid first\ a = \mathbf{?}\} \\
FreeRoleVar &== \{a : RoleVar \mid first\ a = \mathbf{?}\} \\
FreeTimeVar &== \{a : TimeVar \mid first\ a = \mathbf{?}\}
\end{aligned}$$

Given this, we can define the type of all bound and free variables introduced so far.

$$\begin{aligned}
BoundVar ::= & \ boundavar\langle\langle BoundAgentVar\rangle\rangle \\
| & \ boundrvar\langle\langle BoundRoleVar\rangle\rangle \\
| & \ boundtvar\langle\langle BoundTimeVar\rangle\rangle \\
FreeVar ::= & \ freeavar\langle\langle FreeAgentVar\rangle\rangle \\
| & \ freervar\langle\langle FreeRoleVar\rangle\rangle \\
| & \ freetvar\langle\langle FreeTimeVar\rangle\rangle
\end{aligned}$$

The set of all variables may then also be defined.

$$\begin{aligned}
Var ::= & \ free\langle\langle FreeVar\rangle\rangle \\
| & \ bound\langle\langle BoundVar\rangle\rangle
\end{aligned}$$

Now that we have the set of variables and constants, we can define the set of primitive terms. Terms are either constants or variables, or include other terms. We also include basic terms for integers, reals and strings.

$$\begin{aligned}
AgentTerm &::= agentconst\langle\langle AgentConst\rangle\rangle \mid agentvar\langle\langle AgentVar\rangle\rangle \\
RoleTerm &::= roleconst\langle\langle RoleConst\rangle\rangle \mid rolevar\langle\langle RoleVar\rangle\rangle \\
TimeTerm &::= timeconst\langle\langle TimeConst\rangle\rangle \mid timevar\langle\langle TimeVar\rangle\rangle
\end{aligned}$$

In turn, we can define the set of all terms that are used in the specification.

$$\begin{aligned}
Term ::= & \ agentterm\langle\langle AgentTerm\rangle\rangle \\
| & \ roleterm\langle\langle RoleTerm\rangle\rangle \\
| & \ timeterm\langle\langle TimeTerm\rangle\rangle
\end{aligned}$$

4.2. Roles and social relationships

Electronic institutions provide a generic and reusable means of structuring sophisticated interactions of multiple agents that can be applied to multiple distinct instances of such interactions. In this respect, we want to avoid specifying institutions in terms of individual agents, and indeed it would not be practical to do so. We therefore require an abstraction of an agent in the context of a social setting; that is, as in human institutions, agents manifest certain patterns of behaviour and interaction. In consequence, we introduce the notion of *role*, which enables us to describe, design and understand interactions in an abstract and re-usable sense. Roles are concrete patterns of individual behaviour: what can be said and to whom. In order to deal with the problem of conflict between agents, and to determine whether agents incarnating a role could take on another role, it is useful to be able to specify a set of relationships between roles. These relationships between roles can either further restrict behaviour (for example, a committee member cannot perform the actions of the chair) or

determine subsumption policies (for example, the chair may take on the responsibilities of any committee member). Thus, our framework contains roles and any relationships that we may wish to specify between them.

Formally, we define the set of system roles (line 1 in Schema 3) as a set of role constants (such as auctioneer, buyer, or auction manager). In addition, a set of properties is associated with each of the system roles (line 2) (for example, any buyer is associated with a credit rating, which means that any agent playing that role must have a specific credit rating). These attributes represent the public view to the institution of any agent incarnating a particular role, where the institution is authorised to consult and modify the values of these attributes in response to the actions undertaken by the agents. The predicate (line 3) ensures that only defined roles may have properties.

Roles

$roles : \mathbb{P}\, RoleConst$ [1]

$roleproperties : RoleConst \pfun \mathbb{P}_1\, Attribute$ [2]

$\mathrm{dom}\, roleproperties \subseteq roles$ [3]

Schema 3. Roles that agents can play in an electronic institution.

An agent that joins an institution is given a special role, called *guest*, until it states the roles it wants to play and its credentials for doing so are checked. This is because agents are autonomous, and decide for themselves what to do. Our institutional structure says nothing about the internal decision making of agents but instead specifies what must be done only after they have chosen particular roles in line with their objectives. Formally, we declare a global *guest* variable by using an axiomatic definition in Z.

| *guest* : *RoleConst*

Next we specify the social relationships between the roles described above. In Schema 4, we first include the previous *Roles* schema (line 1). Then we define the set of all binary relationships (line 2) (such as authority, power, and so on), and in the predicate (line 3) ensure that the relationships and roles are well defined: any role in a social relationship with another must be one of the previously identified roles of the institution.

SocialRelationships

$Roles$ [1]

$socialrelationships : \mathbb{P}(RoleConst \rel RoleConst)$ [2]

$\forall\, sr : socialrelationships \bullet ((\mathrm{dom}\, sr) \cup (\mathrm{ran}\, sr)) \subseteq roles$ [3]

Schema 4. Relationships between roles.

4.3. Interaction mechanism

Although humans interact in a variety of forms (visual, phonetic, linguistic), for societies involving computational agents of the kind we are addressing it would seem that anything other than linguistic interaction is impossible. Agents playing roles must thus interact by means of a linguistic interaction mechanism, and in this sense electronic institutions are dialogical: agents interact via the sending and receiving of utterances using the classical approach of Austin and Searle [103]. The basic template for any such exchange must include, as a bare minimum, the illocutionary force of the communication (such as asserting), a formula from the domain language, a time term, a sender agent and a receiver agent, both with associated roles. The syntax is thus similar to that of the agent communication languages, FIPA or KQML [47,70], but here we provide the core components that must be part of an electronic institution communication model. For example, we might have an inform force, with a domain language formula *bid (cod, EUR15)*, at time *36487*, from a *buyer* role to an *auctioneer* role (which may then become instantiated at some later point with the agents *Alice* and *Bob* as sender and receiver). In our group movie example we may have a request force, with domain language *see ("Shawshank Redemption")* from a *friend* role to a *friend* role.

Now, when defining an interaction mechanism, some elements must be abstracted away. In particular, it cannot be known at design time which concrete formulae will be exchanged by agents at execution time, who will talk to whom or when. However, it is known which roles those unknown agents will instantiate in a particular utterance and its illocutionary force. That is, only the roles will be known.

Formally, we define the schema *UtteranceTemplate* (Schema 5), which first requires introduction of a given set for *IllocutionaryForce*. The schema has a force (line 1), a formula for the content (line 2), a term to time-stamp the utterance (line 3), the participants (line 4) and their roles (line 5). The predicates assert that the sender and receiver agents must be distinct (line 6), that the formula must belong to the domain ontology from the *Languages* schema (line 7), that the agents are variables (line 8), and that time is a variable (line 9) that is necessarily free (line 10). The reason for this last predicate for time is that we cannot determine at design time when an utterance must take place, even though it may be possible, at design time, to relate one agent variable to the agent variable of a previous utterance template.

[*IllocutionaryForce*]

UtteranceTemplate

$force : IllocutionaryForce$ [1]
$dlformula : DLFormula$ [2]
$time : TimeTerm$ [3]
$sender, receiver : AgentTerm$ [4]
$sendrole, receiverole : RoleTerm$ [5]

$sender \neq receiver$ [6]
$dlformula \in languages.domainontology$ [7]
$\{sender, receiver\} \subseteq (\text{ran } agentvar)$ [8]
$time \in (\text{ran } timevar)$ [9]
$(timevar^{\sim} time) \in FreeTimeVar$ [10]

Schema 5. Utterance template – the fundamental data structure for communication between two agents playing roles.

4.4. Scenes and transitions

Agents interact by exchanging utterances within a group. More precisely, utterances always occur in a scene in a particular environment that involves the goals of the participating agents, the roles they are playing, and a particular shared set of variables modelling the attributes of the scene. If these utterances are not grouped into scenes (in which agents adopt specific roles), then it is not possible to interpret them, since any previous utterances provide agents with the necessary context for understanding them. In fact, the only way to understand the significance of what one agent utters to a group of other agents is to understand the role it is playing in that scene, the roles the others play, and the history of what has been said up to that point.

For example, consider the *group movie* scenario in which a group of agents decide which movie to go and see together. This is just one of several situations that agents may encounter in the larger set of activities involved in actually going to see a movie, including setting a time and place to meet, making travel arrangements, and actually going to the cinema. Each of these separate situations can be regarded as a scene in which agents exchange utterances that move a conversation forward in discrete steps. Thus, an agent makes an utterance in a particular state of a scene, which we call a *conversation place*, or just a *place*, and moves the conversation from one such place to another. At any point in its lifetime, a scene is in one of these conversation places, transitions between which are achieved either by agents making utterances, or eventually by agents not uttering anything at all after some time.

From an operational perspective, an electronic institution is thus a network of activities performed by agents. We refer to the combination of the different utterance activities and their interconnections as a *scene network* which, in essence, amounts to a multi-agent workflow. The two basic components on which this is based are thus *scenes* to model activities, and *transitions* to model the flow of agents between these scenes, as discussed next.

4.4.1. Scenes

As we have seen, an electronic institution comprises several scenes through which agents pass, where a scene is a directed graph of *places*, in which the links between places correspond to actions. In this context, an action is either a *line* (extending the theatre metaphor), which is an utterance with preconditions and postconditions, or a *pause*, which is a specific amount of time in which nothing is uttered by any agent. We choose to use preconditions because the electronic institution must verify that the agent actions (utterances) are performed in the right context, and postconditions because electronic institutions must guarantee that the consequences of agent actions are enacted. We use time-outs as a way of protecting the institution from the inactivity of agents. Both *lines* and pauses move a scene from one conversation place to another and are thus links between them. For instance, an auctioneer with an opening *line* to start an auction moves the scene to a place at which bids can be made by buyers, while a pause of five seconds without a bid being heard moves the scene to a place at which the auctioneer can close the auction round.

To specify this, we define the two data structures, *Line* and *Pause* (see Schemas 6 and 7). A *Line* contains an utterance template (line 1), a set of constraints (line 2) that are the precondition that must be satisfied, and a sequence of postcondition actions (line 3) that are executed in order. A *Pause* is defined as an amount of time (line 1) plus constraints and actions as before. Finally, we define the *Action* type as the aggregation of *Line* and *Pause*.

$$Action ::= line\langle\langle Line\rangle\rangle \mid pause\langle\langle Pause\rangle\rangle$$

Line

$pattern : UtteranceTemplate$ [1]
$constraints : \mathbb{P}\, CLFormula$ [2]
$updates : \text{seq}\, ULFormula$ [3]

Schema 6. A *Line* is an utterance template labelled with a set of constraints (that must be satisfied for the utterance to occur) and a sequence of updates (that take place once the utterance has been executed).

Pause

$time : \mathbb{N}$	[1]
$constraints : \mathbb{P}\ CLFormula$	[2]
$actions : \text{seq}\ ULFormula$	[3]

Schema 7. A *Pause* takes place if no utterance has occurred within a pre-defined time period.

Having specified these data structures, we can now specify scenes in the *Scene* schema (see Schema 8), which contains: a name for the scene that enables us to identify it (line 1), such as *DutchAuction*; the set of role identifiers in the scene (line 2); the limits on the number of agents allowed to instantiate each role (line 3; for example, there must be a minimum of 3 and a maximum of 20 bidders in a particular auction); a set of places in the conversation graph (line 4) and the moves between them (line 5) that are traversed when an action is made; the actions labelling each link that arises as a result of such moves (line 6); the initial place (line 7) and the set of possible final places (line 8); for each scene role, the set of *access* and *leaving* conversation places (line 9); and a flag to indicate whether scenes can be multiply instantiated to accommodate the possibility of repeating the activity for different groups of agents (line 10).

The predicates of the schema are as follows. First, we specify that all states are reachable from the starting place, by stating that any non-starting place is in the reflexive transitive closure of *moves* domain-restricted to the *start* place (line 11). Then, two integrity constraints require the starting place and the closing places (line 12), and access and leaving places (line 13), to be legitimate places within the scene. Clearly, only roles from the scene can be allowed to join or leave (line 14) and, for any agent accessing the scene, there must always be a path leading to a closing place (line 15). An action labels every link between places (line 16), links are defined only over places (line 17) and must not lead to the initial conversation place (line 18) nor lead out of any closing place (line 19). In all this, the directed graph of conversation places and utterances must be connected (line 20). Moreover, for any place where an agent can join (access) there must exist at least one reachable place where the agent can leave (line 21). Finally, the *limits* function is defined for all scene roles (line 22).

[*ConvPlace*, *SceneName*]

An illustration of the basic elements of the *Scene* schema can be seen in Fig. 6. The scene shown contains three places: $p0$, $p1$, and $p2$. Agents playing role $R1$ can joint at place $p0$ and leave at place $p2$, for example.

Scene

$sname : SceneName$	[1]
$sceneroles : \mathbb{P}\ RoleConst$	[2]
$limits : RoleConst \pfun \mathbb{P}_1(\mathbb{N})$	[3]
$places : \mathbb{P}\ ConvPlace$	[4]
$moves : ConvPlace \rel ConvPlace$	[5]
$link : (ConvPlace \times ConvPlace) \pfun Action$	[6]
$start : ConvPlace$	[7]
$closing : \mathbb{P}\ ConvPlace$	[8]
$access, leaving : RoleConst \pfun (\mathbb{P}\ ConvPlace)$	[9]
$multiple : Bool$	[10]
$\forall\, cs : ConvPlace \mid cs \neq start \bullet cs \in (\text{ran}(\{start\} \lhd (moves\star)))$	[11]
$(\{start\} \cup closing) \subseteq places$	[12]
$\bigcup(\text{ran}\ access \cup \text{ran}\ leaving) \subseteq places$	[13]
$(\text{dom}\ access \cup \text{dom}\ leaving) \subseteq sceneroles$	[14]
$\forall\, r : RoleConst;\ c_1 : ConvPlace \mid c_1 \in (access\ r) \bullet \exists\, c_2 : closing \bullet (c_1, c_2) \in moves\star$	[15]
$\text{dom}\ link = moves$	[16]
$(\text{dom}\ moves \cup \text{ran}\ moves) = places$	[17]
$start \notin (\text{ran}\ moves)$	[18]
$closing \cap (\text{dom}\ moves) = \{\}$	[19]
$\forall\, s_1, s_2 : places \bullet (s_1, s_2) \in (moves \cup moves^{\sim})\star$	[20]
$\forall\, r : RoleConst;\ c_1 : ConvPlace \mid c_1 \in (access\ r) \bullet (\exists\, c_2 \in (leaving\ r) \bullet (c_1, c_2) \in moves\star)$	[21]
$\text{dom}\ limits = sceneroles$	[22]

Schema 8. Scenes are key data structures in an electronic institution, and include the structure of possible dialogues, the roles agents can play in the scene, at what points agents playing roles may join or leave, the scene's unique start state, and its set of closing states.

We use finite state machines (FSMs) to represent dialogues within a scene, for two reasons: first, because FSMs are a well-known formalism with which most engineers feel comfortable; and second, because they have a clear declarative

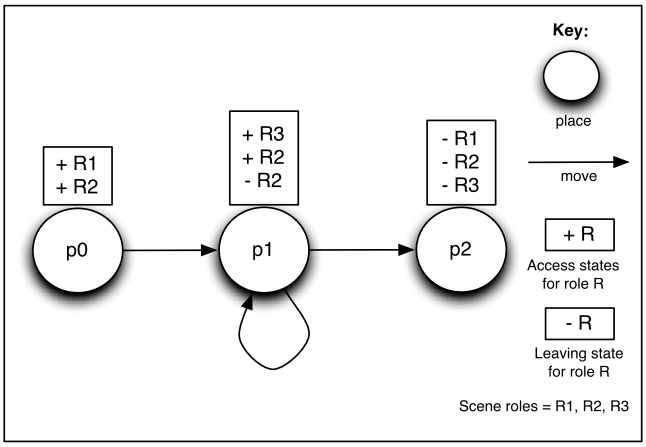

Fig. 6. An illustration of the mechanics of a scene.

semantics. Within a scene specification, the FSM indicates what can be said by agents in particular dialogical contexts (the FSM states) without imposing a particular control flow on agents.

While process workflow languages like BPMN[11] or BPEL [17] have a notion of both choreography (how to model the interaction between agents) and orchestration (programming of the internals of agents), electronic institutions are only concerned with choreography and not orchestration. Indeed, the notion of workflow is very general and can be applied to any type of process and agent [65,99] in widely used tools like BPMN or BPEL. However, choreography is limited to defining messages that interconnect the internals of the processes and does not provide the rich structure that electronic institutions bring, with scenes and scene networks that specify the role flow.

As we are neutral with respect to the internals of agents, in electronic institutions no link is made between a dialogue and the internal workflow of agents, so that the FSM must be understood as a pure declaration of restrictions on agent communication and not as a program to be run by the agents.

4.4.2. Transitions and arcs

Recall that electronic institutions are neutral with respect to the architecture of agents. The institution has no sense of the internals of participating agents and certainly no understanding of their goals and motives within the institution. However, the institution needs to connect all the possible suites of activities of agents. This is often referred to as choreographing the activities of agents in workflow terminology. In order to do this we need to introduce locations where agents can wait, regroup and synchronise themselves so that the future activity of the institution can be properly choreographed. We call these locations *transitions* and, along with arcs that connect transitions with scenes, we can build the scene network of an institution (see Schemas 9 and 10). The resulting scene network allows groups of agents to jointly decide whether to start a new scene, join a scene, leave a scene, or close a scene. This is where the theatrical metaphor breaks down: in contrast to traditional plays in which scenes are sequential, our model permits a network of interconnected scenes in which agents can play multiple roles, even concurrently.

As indicated, arcs link scenes to transitions and transitions to scenes, with each arc associated with a set of actions from the action language and constraints from the constraint language, corresponding to preconditions that govern the ability of an agent, playing a particular role, to traverse an arc. For example, in the *group movie* scenario, an agent seeking to go to the group movie must express a preference for a particular movie, and must pay part of a group rate fee in advance. In such a case, there may be an arc to the movie decision scene with the constraint that the agent has voted for a movie, and an action that decreases the agent's balance by the movie fee. Networking scenes in this way is necessary to capture the causal dependencies between them, including order, synchronisation, parallelism, choice points, creation, change of roles between scenes, and so on.

Those arcs that connect scenes to transitions are *outarcs* (since they go out from a scene), while those that connect transitions to scenes are *inarcs* (since they go in to a scene). Fig. 7 illustrates this: transition $T1$ has an inarc ($inarc1$) to scene $S1$. There is then $outarc2$ from $S1$ to $T2$, and $inarc2$ from $T2$ to $S2$. The mode of a transition can be *and*, or *or*. If the mode is *and*, agents will participate in all scenes linked from that transition. If the mode is *or*, agents select only one. Formally, a transition is defined by its name and its mode.

$$\mathit{TransitionMode} ::= \mathit{and} \mid \mathit{or}$$

$$[\mathit{TransName}]$$

[11] http://www.bpmn.org/.

Transition

$tname : TransName$
$mode : TransitionMode$

Schema 9. Transitions link scenes.

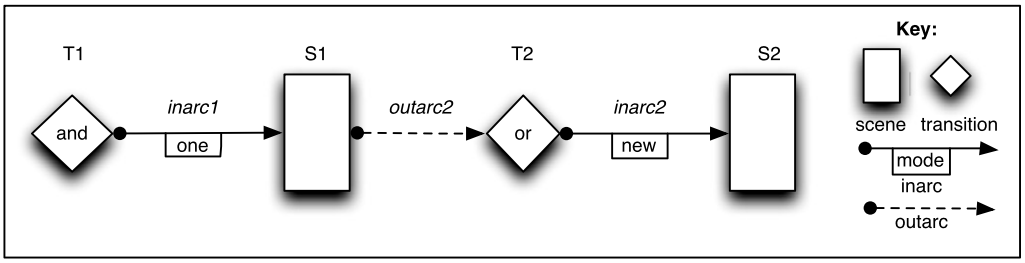

Fig. 7. Scenes, transitions, inarcs and outarcs.

Formally, outarcs have no mode, but inarcs do have a mode. The mode of an inarc represents the way in which the agents join the target scene. For instance, the mode *new* indicates that a new target scene must be created.

$[ArcMode]$

$InArc == \{a : Arc \mid \text{defined } a.mode\}$
$OutArc == \{a : Arc \mid \text{undefined } a.mode\}$

Arc

$transition : Transition$
$scene : Scene$
$mode : \text{optional } [ArcMode]$

Schema 10. Arcs link transitions and scenes.

4.5. Scene network

Given the discussion of scenes, transitions and arcs, we have the basic components necessary to characterise and represent the entire institutional framework. In an auction house, for example, we need to connect together the scenes for registration, admission, auction, payment, and so on. In particular, the scene network just outlined enables us to capture the causal dependencies between scenes indicating order (which scenes must follow others), synchronisation of scenes (which scenes must finish before new ones start), parallelism (which scenes start after others finish), or choice points (which particular scene to move to when there are several options). This relates not just to the scenes themselves, but also to the transitions that enable agents playing particular roles to move between scenes. In this way, each scene may be connected to multiple transitions, and each transition to multiple scenes.

Formally, this is captured in the *SceneNetwork* schema (Schema 11), which brings together these different components (scenes, transitions, and arcs) into an institutional model. Clearly, this contains the finite, non-empty set of all scenes (line 1), of which one is the entry scene and one is the exit scene (lines 2, 3, and 9), that must be distinct (line 10). It also contains finite non-empty sets of transitions (line 4), and arcs that are links from either scenes to transitions (outarcs) or transitions to scenes (inarcs) (line 5), where inarcs are defined to be those arcs with a defined type (line 11), and outarcs those with an undefined type (line 12).

A labelling function (*disjnorm*) (line 6) maps each arc to a disjunctive normal form of agent variables and role identifiers, defining the possible (non-empty) set of agents and associated (non-empty) roles that can simultaneously traverse the arc and that are specified for all arcs (line 13). Similarly, a second labelling function (*constraints*) maps arcs to constraints (in our constraint language) (line 7), which individual agents must fulfil in order to traverse the arc and again that are specified for all arcs (line 14). A third labelling function (*updates*) maps arcs to actions (in our update language) (line 8) that are triggered when individual agents traverse the arc and that are specified for all arcs (line 15). In this way, each arc is labelled with (possibly empty) sets of constraints and actions.

Finally, to enforce the restrictions discussed above on entry and exit scenes, it is not possible to return to the entry scene (line 16), nor is it possible to leave the exit scene and return to another scene in the institutional scene network (line 17).

SceneNetwork

$$allscenes : \mathbb{P}_1\ Scene \quad [1]$$
$$entryscene : Scene \quad [2]$$
$$exitscene : Scene \quad [3]$$
$$transitions : \mathbb{P}_1\ Transition \quad [4]$$
$$arcs, inarcs, outarcs : \mathbb{P}_1\ Arc \quad [5]$$
$$disjnorm : Arc \nrightarrow \mathbb{P}_1(\mathbb{P}_1(AgentVar \times RoleConst)) \quad [6]$$
$$constraints : Arc \nrightarrow (\mathbb{P}\ CLFormula) \quad [7]$$
$$updates : Arc \nrightarrow (\mathbb{P}\ ULFormula) \quad [8]$$

$$\{entryscene, exitscene\} \subseteq allscenes \quad [9]$$
$$entryscene \neq exitscene \quad [10]$$
$$inarcs = \{a : arcs \mid \text{undefined } a.mode\} \quad [11]$$
$$outarcs = \{a : arcs \mid \text{defined } a.mode\} \quad [12]$$
$$\text{dom } disjnorm = arcs \quad [13]$$
$$\text{dom } constraints = arcs \quad [14]$$
$$\text{dom } updates = arcs \quad [15]$$
$$\neg\ (\exists\ a : InArc \bullet a.scene = entryscene) \quad [16]$$
$$\neg\ (\exists\ a : OutArc \bullet a.scene = exitscene) \quad [17]$$

Schema 11. A scene network using arcs and transitions where *inarcs* link transitions to scenes and *outarcs* link scenes to transitions; each arc is labelled with constraints, updates and a set of disjunctions of agent variables and role constants.

4.6. Standard constraints in a scene network

While this specification and description of a scene network captures the basic components, it omits several important aspects that are vital to the design and definition of electronic institutions. In order to elaborate these aspects elegantly, we define the *SceneNetworkSystem* schema containing six auxiliary functions: the first two (lines 1, 2, 7, and 8) return the set of agent variables and role identifiers that label an arc; the next two (lines 3, 4, 9 and 10) return the inarcs and outarcs (respectively) of a scene within the network; and the final two (lines 5, 6, 11 and 12) return the inarcs and outarcs (respectively) of a transition. Given these functions, we specify the standard constraints of an electronic institution in the *SceneNetworkStructure* schema (Schema 12).

As we have seen, the movement of agents between scenes is mediated by transitions and arcs. Within a scene, the roles that agents play are bound to particular agents, but when agents leave a scene, these bindings are removed, since agents must be free to play different roles in subsequent scenes. In fact, this is one of the key tasks of transitions, which bind the agent variables to specific agents. Thus, on any outarc from a scene, all the agent variables are free (line 1), and on any inarc to a scene, all the agent variables are bound (line 2).

A consequence of this view of transitions as mediating the movement of agents is that we must preserve agents at transitions; in particular we cannot lose agents at a transition, nor can we introduce new agents. Thus, the union of agent variables labelling outarcs of any transition must be the same as the union of all agent variables labelling inarcs from that transition (line 3). (Note that we take the range of the agent variables in order to strip away the variable prefix before testing for equality.) The same is true for roles as well as agents: the union of the roles labelling the outarcs of every scene (line 5), and the union of all roles labelling the inarcs (line 6), must be equal to the scene's roles, so that any role can leave a scene, and there is no scene role that cannot join that scene, respectively.

Then, for each scene in the network, there must be a path from the entry scene to the exit scene that passes through that scene, so that each scene must be reachable. Line 4 uses the reflexive transitive closure of the *move* relation. It relates one scene to another scene if there is a series of intermediate transitions and scenes that can be traversed to arrive from the first scene to the second scene. The predicate states that any scene can always be reached from the entry scene, and the exit scene can always be reached from it. In essence, this ensures that a scene network is well defined and that all scenes can potentially be reached.

We can also elaborate the ways in which roles join scenes through access conversation places. In particular, for every role labelling an inarc of a scene, there must be at least one access conversation place in that scene for that role (line 7). For every conjunction in the disjunction of agent roles labelling an inarc, there must be an access state in the scene that contains all the roles that are contained in the conjunction. Intuitively, this means that all agents with their associated roles can join together (line 8). Finally, the access states for every role labelling an inarc of type *new* must include the initial conversation place (line 9). We explore this mode in Section 7 on operations, but anticipate here that an inarc of type *new* specifies that the agents following the arc are allowed to start a scene. In other words, this arc mode specifies when agents can bootstrap a scene at run-time.

| *new* : *ArcMode*

SceneNetworkSystem

SceneNetwork

$$arcagentvars : Arc \pfun (\mathbb{P}\, AgentVar) \quad [1]$$
$$arcroles : Arc \pfun (\mathbb{P}\, RoleConst) \quad [2]$$
$$sceneinarcs : Scene \pfun (\mathbb{P}\, Arc) \quad [3]$$
$$sceneoutarcs : Scene \pfun (\mathbb{P}\, Arc) \quad [4]$$
$$transitioninarcs : Transition \pfun (\mathbb{P}\, Arc) \quad [5]$$
$$transitionoutarcs : Transition \pfun (\mathbb{P}\, Arc) \quad [6]$$

$$\forall\, arc : arcs \bullet$$
$$arcagentvars(arc) = \{a : AgentVar;\ r : RoleConst \mid (a, r) \in (\bigcup(disjnorm(arc))) \bullet a\} \wedge \quad [7]$$
$$arcroles(arc) = \{a : AgentVar;\ r : RoleConst \mid (a, r) \in (\bigcup(disjnorm(arc))) \bullet r\} \quad [8]$$
$$\forall\, s : allscenes \bullet$$
$$sceneinarcs(s) = \{i : InArc \mid i.scene = s\} \wedge \quad [9]$$
$$sceneoutarcs(s) = \{o : OutArc \mid o.scene = s\} \quad [10]$$
$$\forall\, t : transitions \bullet$$
$$transitioninarcs(t) = \{i : InArc \mid i.transition = t\} \wedge \quad [11]$$
$$transitionoutarcs(t) = \{o : OutArc \mid o.transition = t\} \quad [12]$$

Schema 12. Six functions that return the set of agent variables and role constants for an arc, and the set of *inarcs* and *outarcs* for any scene or transition.

SceneNetworkStructure

SceneNetworkSystem

$$\forall\, in : inarcs \bullet \mathrm{dom}(arcagentvars(in)) = \{?\} \quad [1]$$
$$\forall\, out : outarcs \bullet \mathrm{dom}(arcagentvars(out)) = \{!\} \quad [2]$$
$$\forall\, t : Transition \bullet \bigcup\{out : transitionoutarcs(t) \bullet \mathrm{ran}(arcagentvars(out))\} = \bigcup\{in : transitioninarcs(t) \bullet \mathrm{ran}(arcagentvars(in))\} \quad [3]$$
$$\forall\, s : allscenes \bullet \{(entryscene, s), (s, exitscene)\} \subseteq \{out : OutArc;\ in : InArc \mid out.transition = in.transition \bullet (out.scene, in.scene)\}\star \quad [4]$$
$$\forall\, s : allscenes \bullet \bigcup\{out : sceneoutarcs(s) \bullet arcroles(out)\} = s.sceneroles \quad [5]$$
$$\forall\, s : allscenes \bullet \bigcup\{in : sceneinarcs(s) \bullet arcroles(in)\} = s.sceneroles \quad [6]$$
$$\forall\, r : RoleConst;\ in : inarcs;\ s : allscenes \mid (r \in arcroles(in)) \wedge in.scene = s \bullet r \in (\mathrm{dom}\, s.access) \quad [7]$$
$$\forall\, conjunction : \mathbb{P}(AgentVar \times RoleConst);\ in : inarcs \mid (conjunction \in disjnorm(in)) \bullet (\mathrm{ran}\, conjunction) \subseteq (\mathrm{dom}\, in.scene.access) \quad [8]$$
$$\forall\, r : RoleConst;\ in : inarcs \mid in.mode = \{new\} \wedge r \in arcroles(in) \bullet (in.scene).start \in (in.scene).access(r) \quad [9]$$

Schema 13. Constraints that must be satisfied for any well-defined electronic institution.

4.7. The electronic institution

As a result of all that has gone before, it is trivial now to define an electronic institution as a data structure that consists of the *SocialRelationships* and the *SceneNetworkStructure* components. Once this is defined, along with the set of languages (at the end of Section 3), it then becomes possible to run an institution (see Schemas 13 and 14).

ElectronicInstitution

SocialRelationships

SceneNetworkStructure

Schema 14. An electronic institution.

5. An illustration of electronic institutions

In this section we show how the formal model described so far can be used to specify the well-known and understood institution of a fish market. The design produced can then be used as the blueprint for the implementation of the modelled fish market as an electronic instantiation.

In the Mediterranean regions fresh fish has traditionally been sold through downward bidding auctions, operating in auction houses close to harbours. Here, fish is grouped into sets of boxes, called lots, and sold in auctions that follow the Dutch protocol. The fish market can be described as a place where several activities occur simultaneously, at different locations, but with some causal connection [78]. Fishermen deliver their goods to a seller-administrator, an expert responsible for

weighing and pricing fish boxes according to quality. The principal scene is the auction itself, in which buyers bid for tagged fish boxes that are presented by an auctioneer, who calls prices in descending order, following an open cry, downward bidding protocol, a variation of the traditional Dutch auction protocol that proceeds as described below.

1. The auctioneer chooses an item from a batch of goods that are sorted according to the order in which they were delivered to the administrator.
2. For a specific item, the auctioneer opens a bidding round by quoting offers downward from the starting price of the item as previously fixed by the administrator. This figure must be higher than the reserve price previously defined by the seller.
3. For each successive price the auctioneer calls, several situations might arise:
   - Several buyers submit their bids at the current price. In this case the item is not sold to any buyer and the auctioneer restarts the round at a higher price.
   - Only one buyer submits a bid at the current price. The item is sold to this buyer as long as his credit can support his bid. If the credit is not sufficient the round is restarted by the auctioneer at a higher price, and the unsuccessful bidder is sanctioned.
   - No buyer submits a bid at the current price. If the reserve price has not yet been reached, the auctioneer quotes a new price which is obtained by decreasing the current price, according to a price step defined by the auctioneer. Otherwise, the auctioneer declares the item as withdrawn and closes the round.
4. The first three steps are repeated until there are no more goods left.

Once an item is sold, its cost is immediately charged to the purchaser's account, and the item must be taken away by the purchaser immediately, since the institution is responsible neither for warehousing fish nor for its delivery. As buyers may run out of credit during a market session, they may update their accounts at any time. In the case of a buyer exceeding their credit when submitting a bid, the bid is declared invalid and the buyer cannot return until their credit is reactivated by the accounting office. Finally, a supervisor decides when to open an auction and when to close it.

Now, suppose that we wish to capture the operation of a fish market in the formal model of an electronic institution. For the sake of simplicity, we concentrate on specifying a Dutch auction where no collisions occur (that is, there are no instances of more than one agent bidding at the same price). In what follows, we show how to connect the scene specifying the Dutch auction to other scenes in the scene network required by the electronic institution. Before specifying the structure, recall that for any institution we must first define the domain, attribute, update and constraint languages (as stated in Schema 2). We will assume that the languages are already defined to be consistent with the examples of Section 3 (see Figs. 2, 3, 4, 5). Note that these languages, and the example we have chosen, are simple and used for the purpose of exposition and would all be more complex in reality. We focus our attention on the structural components defined in the previous section.

5.1. Social relationships

First, recall from Schema 14 that an electronic institution is composed of social relationships along with a scene network structure. In addition, Schema 4 contains the roles in the institution along with the social relationships between them. In the fish market, these roles are: buyers, sellers, auctioneer, seller-administrator, buyer-administrator, accountant, and supervisor. Now, according to Schema 3 we must identify the properties of each role. In the Dutch auction, a buyer requires *Credit* as an attribute; more specifically, this credit is an integer and is part of our attribute language.

$$\mathit{Credit} : \mathit{Integer} \subseteq \mathit{AttributeLanguage}$$

Moreover, since the auctioneer charges a fee per transaction as a percentage of the sale, we require another attribute for the auctioneer role: $\mathit{FeePercentage} : \mathit{Integer}$. Finally, by charging a percentage of every sale, the auctioneer accumulates profit during a market session, requiring a further attribute $\mathit{Benefit} : \mathit{Integer}$ for the auctioneer role. Then, following Schema 4, we must identify the social relationships between roles. For instance, fish markets do not allow an auctioneer to participate as a buyer in the auctions they are facilitating. To address this, we define a separation of duties (ssd) relationship that prevents roles from being adopted simultaneously (for example $\mathit{ssd}(\mathit{auctioneer}, \mathit{buyer})$ and $\mathit{ssd}(\mathit{auctioneer}, \mathit{seller})$).

5.2. Scene network structure

In this section we show how to put together the specification of a simplified version of the fish market's Dutch auction and how to connect the Dutch auction scene to another scene.

5.2.1. Languages

First, we use the languages we have previously defined to specify the domain ontology required for a Dutch auction in which only buyers and auctioneers participate. In addition, we define attributes, constraints and updates.

Recall from Section 3 that the domain language helps us identify the concepts in the domain ontology (such as fish, kgs, buyer, etc.). Table 2 lists the concepts and relationships required in a Dutch auction. From Section 5.1, recall that

Table 2
Concepts in the Dutch auction domain ontology along with their relationships.

Concepts	Semantics	Relationships	Semantics
boxid **ISA** Integer	Fish box identifier	fishbox(boxid,fishtype,kgs)	Features of a fish box
fishtype **ISA** String	Type of fish	offer(boxid,fishtype,kgs,price)	fish box offered at starting price
price **ISA** Integer	Price value	bid(boxid,price)	Bid price for a fish box
seller **ISA** String	Seller name	sale(boxid,price)	Sale price of a fish box
kgs **ISA** Integer	Weight of fish	withdrawal(boxid,price)	Withdrawal price of a fish box

Table 3
The Dutch auction scene in terms of the specification definition of a *Scene* in Schema 8.

sname:	**Dutch Auction**
sceneroles:	auctioneer, buyer
limits:	auctioneer → 1, buyer → 100
places:	start auction, bidding, call price, winner declaration, end auction
moves:	(start auction, bidding), (bidding, call price), (call price, bidding), (call price, end auction) (bidding, bidding), (bidding, winner declaration), (winner declaration, end auction)
link:	(start auction, bidding) → *Line*1, (bidding, call price) → *Pause*1, (call price, bidding) → *Line*2 (call price, end auction) → *Line*3, (bidding, bidding) → *Line*4, (bidding, winner declaration) → *Line*5 (winner declaration, end auction) → Line_6
start:	start auction
closing:	end auction
access:	auctioneer → start auction, buyer → start auction
leaving:	auctioneer → end auction, buyer → end auction
multiple:	False

both the buyer and the auctioneer roles have attributes (*Credit*, *Fee*, *Benefit*). Now, we assume that every fish box placed at auction has a reserve price, hence $ReservePrice : Integer \subseteq AttributeLanguage$. In order to call prices downwards, the auctioneer requires a further attribute, $PriceStep : Integer$, to account for the value at which the previous price called will be decreased. Finally, note that we must verify four constraints from the description of the operation of the Dutch auction in the fish market ($C1$ to $C4$) and there are three updates to perform during a Dutch auction ($U1$ to $U3$). (The notation we use is that **?***agentX*.*Credit* refers to the credit attribute of the role of the agent bound to **?***agentX*.)

- ($C1$) The bidder's credit is higher than the current price of the item:
  ?agentX.Credit >= **!**current_price
- ($C2$) The credit of a bidder is less than the current price called by the auctioneer:
  ?agentX.Credit < **!**current_price
- ($C3$) A new price called by the auctioneer must be higher than the item's reserve price:
  ?current_price > ReservePrice
- ($C4$) An item can be withdrawn when the new offer price to be called by the auctioneer is less or equal than the reserve price:
  !current_price <= ReservePrice
- ($U1$) The cost of a fish box is immediately charged to the purcharser's account:
  ?agentX.Credit := **?**agentX.Credit − **!**current_price
- ($U2$) If no bids are received, the next price called by the auctioneer is assessed by decreasing the last price called by some amount:
  !current_price := **!**current_price − PriceStep
- ($U3$) The auctioneer gets a fee after every sale:
  Benefit := Benefit + (FeePercentage × **!**current_price)

We must also identify the illocutionary forces for agents to utter illocutions during the auction as required by Schema 5. In essence, the auctioneer informs bidders about the events occurring during the auction and bidders commit to the bids that they submit. Therefore, we can choose as illocutionary forces *inform* and *commit*.

5.2.2. The Dutch auction scene

Schema 8 details how to construct a scene and Table 3 details how the Dutch auction is instantiated in this data structure. Note that the specifications of *Lines* and *Pauses*, based on Schemas 6 and 7, are included in Table 4. Fig. 8 offers a graphical representation of the scene using the same representation as Fig. 6 of the previous section. More precisely, the scene limits participation in the auction to a single auctioneer and 100 bidders. The choice of places within the scene structure is related to the states through which the auction evolves: *start auction*, *bidding* (where bidders are allowed to bid), *call price* (from where the auctioneer calls a new price), *winner declaration* (from where the auctioneer can declare the winner), *end auction* (from where the auctioneer can close the auction).

Table 4
Lines and *Pauses* in the Dutch auction scene.

*Line*1	
pattern = {	force = inform;
	dlformula = offer(**?**boxid, **?**fishtype, **?**kgs, **?**current_price);
	sender = **?**agentY; receiver = agentall;
	sendrole = auctioneer; receiverole = bidder}
*Pause*1	
time =	200 msec.
actions =	**!**current_price := **!**current_price − PriceStep
*Line*2	
pattern = {	force = inform;
	dlformula = offer(**!**boxid, **!**fishtype, **!**kgs, **?**current_price);
	sender = **!**agentY; receiver = agentall;
	sendrole = auctioneer; receiverole = bidder}
constraints =	**?**current_price > ReservePrice
*Line*3	
pattern = {	force = inform;
	dlformula = withdrawal(**!**boxid, **?**current_price);
	sender = **!**agentY; receiver = agentall;
	sendrole = auctioneer; receiverole = bidder}
constraints =	**?**current_price == ReservePrice
*Line*4	
pattern = {	force = commit;
	dlformula = bid(**!**boxid, **!**current_price);
	sender = **?**agentX; receiver = **!**agentY;
	sendrole = bidder; receiverole = auctioneer}
constraints =	**?**agentX.Credit < **!**current_price
*Line*5	
pattern = {	force = commit;
	dlformula = bid(**!**boxid, **!**current_price);
	sender = **?**agentX; receiver = **!**agentY;
	sendrole = bidder; receiverole = auctioneer}
constraints =	**?**agentX.Credit >= **!**current_price
*Line*6	
pattern = {	force = inform;
	dlformula = sale(**!**boxid, **!**current_price);
	sender = **!**agentY; receiver = agentall;
	sendrole = auctioneer; receiverole = bidder}
constraints =	**!**current_price <= ReservePrice
updates =	**?**agentX.Credit := **?**agentX.Credit − **!**current_price
	!agentY.Benefit := **!**agentY.Benefit + (FeePercentage × **!**current_price)

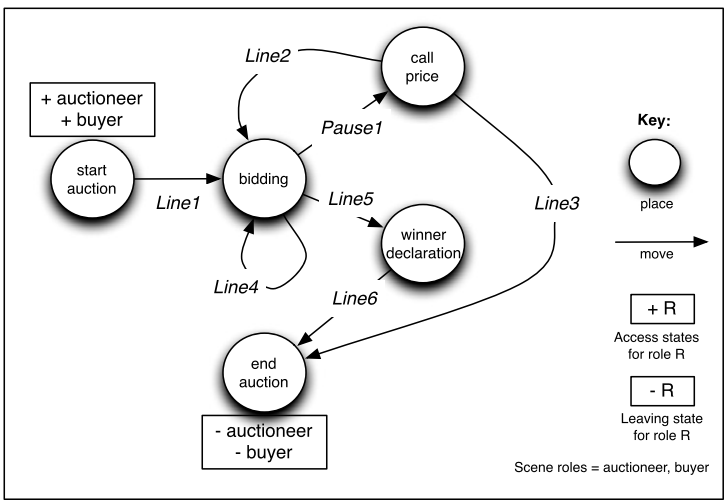

Fig. 8. An illustration of the mechanics of the Dutch auction scene.

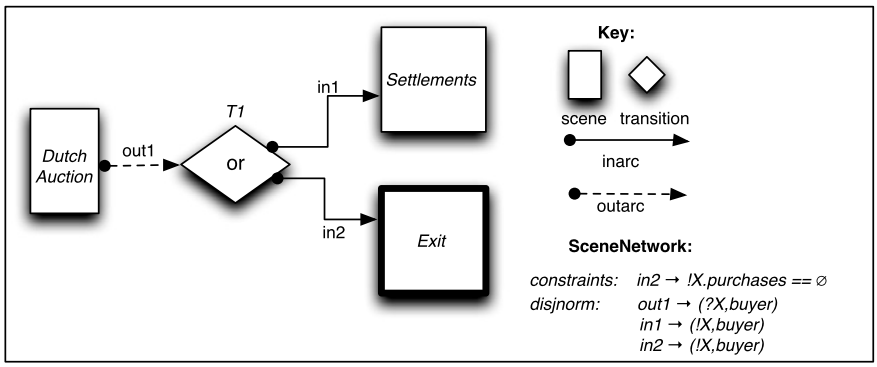

Fig. 9. Scene network encoding the choices for a buyer leaving the auction scene.

It should be clear that the scene starts at the *start auction* place and finishes at the *end auction* place. Moreover, these are the only access and leaving states, respectively, for both the auctioneer and the buyers (so that, in this simplified example, buyers thus cannot leave until an auction has finished). The moves in the specification detail the legal moves from scene place to scene place. For instance, the auction starts by moving the scene from place *start auction* to place *bidding*. This occurs after the auctioneer utters an illocution matching the utterance template linking these two places, namely the utterance template in *Line*1 (of Table 4), putting an item on sale at some starting price.

Thereafter, the scene is designed to proceed as follows. Once at the *bidding* place, three distinct events may occur: either no bids are received in a pre-defined time limit, a potential buyer bids but does not have enough credit, or a potential buyer with enough credit makes a bid. These three cases cause transitions to the *call price*, *bidding*, and *winner declaration* places respectively, and are captured by the moves connecting these places along with their links, namely *Pause*1, *Line*4, and *Line*5. Notice that *Pause*1 uses the update expression $U2$ above (from Section 5.2.1) so that the auction price is updated, *Line*4 uses constraint $C2$ to verify that a buyer has insufficient credit, and *Line*5 uses constraint $C1$ to verify that a buyer does has enough credit. If a buyer does have enough credit, the scene specifies a move to place *winner declaration*, otherwise there is a move that returns the scene to the same place, discarding the invalid bid. From *winner declaration* there is a further move specified by *Line*6, which occurs when the auctioneer declares the winner of the auction and charges the cost of the purchase to the winning buyer, as specified in the update expression $U1$. Finally, if the time specified by *Pause*1 elapses, there is a move that takes the scene from place *bidding* to *call price*. From here, the auctioneer may call a new price, according to *Line*2, as long as the new offer price is strictly higher than the reservation price, as expressed by constraint $C3$. If the new offer price is equal to or less than the reservation price, the scene moves to place *end auction* from where the auctioneer is able to withdraw the item, as specified by *Line*3. The last line of Table 3 indicates that the institution will not host multiple auctions simultaneously (that is, multiple executions of the Dutch auction scene are not possible).

5.2.3. Scene network

Given these key components, we can consider the way in which scenes are connected to compose a scene network structure based in Schema 13. To illustrate, we describe two example scenarios. The first is concerned with how individual buyer agents can leave an auction scene and the second with how to close the market (or institution). As before, in order to demonstrate how the model can be used to design electronic institutions, both examples involve some simplifying assumptions.

In the first example, when a buyer agent leaves the auction they have two choices. They can either go to the accountant's office to collect an invoice and the goods they purchased, or they simply leave the market if they have not made any purchases. Fig. 9 shows a sample of a scene network encoding the choices for any buyer leaving the auction scene along with the *constraints* and *disjnorm* functions that are required in designing an institution as specified by the *SceneNetwork* schema (Schema 11). Here, the Dutch Auction scene is linked to transition (Schema 9) $T1$ through outarc (Schema 10) *out*1. In turn, transition $T1$ is connected by means of inarcs *in*1 and *in*2 to scenes Settlements (the accountant's office) and Exit (leave the market). Since the transition mode of $T1$ is *or*, it is intended to offer buyers the choice to either move into the Settlements scene or the Exit scene. At this point, recall that according to Schema 11, the Exit scene (*exitscene*) is a special scene representing the exit of the institution. The labels on the arcs (corresponding to the specification of the function *disjnorms*) prescribe that the agent that leaves the Dutch Auction scene either joins the Settlements or the Exit scenes (notice the **!** prefix before agent variable X on arcs *in*1 and *in*2). Finally, to ensure that a buyer does not leave the institution without settling their invoices, we specify a constraint (as part of the definition of function *constraints*) to check that the buyer has made no purchases: $!X.purchases == \emptyset$.

In the second example, we must design part of the institution to specify what happens when the supervisor of the market decides to close the fish market. Fig. 10 shows a scene network encoding how the agents playing the market roles auctioneer, seller-administrator, buyer-administrator, and accountant synchronise to close the market. The figure

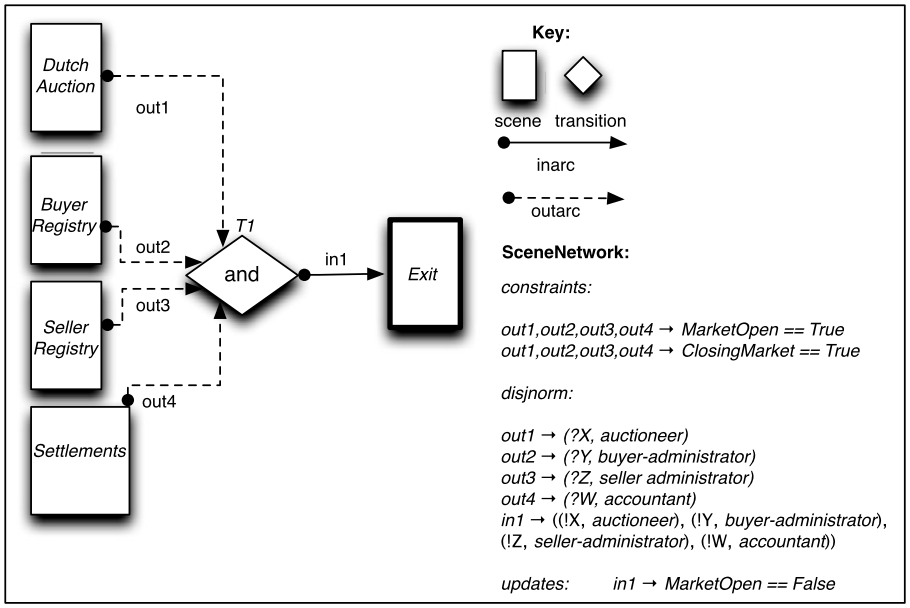

Fig. 10. Scene network encoding the synchronisation of the market's staff members to close the market.

also contains the *constraints*, *disjnorm*, and *updates* functions required by the *SceneNetwork* schema (Schema 11). Here, the Dutch Auction, Buyer Registry, Seller Registry, and Settlements scenes are linked to transition $T1$ through outarcs $out1$, $out2$, $out3$, and $out4$. In turn, transition $T1$ is connected by means of the inarc $in1$ to the *exitscene* of the institution, Exit. Since the transition mode of $T1$ is *and*, it is intended to synchronise the buyer-administrator, the seller-administrator, the auctioneer, and the accountant to leave the market together. Therefore, unlike the previous example, here the transition does not offer a choice to agents. Instead, the transition only allows an agent through to the exit scene when all the agents arrive at the transition, which is when they are all waiting agents requesting to move towards the exit scene. The labels on the arcs (corresponding to the specification of function *disjnorms*) indicate that the agents that leave the Auction, Buyer Registry, Seller Registry, and Settlements scenes are those that together join the *exitscene* Exit (notice the **!** prefix before agent variables X, Y, Z, and W on arc $in1$). Finally, the constraints labelling the outarcs $out1$, $out2$, $out3$, and $out4$ ensure that an agent can only move towards this transition while the market is open (the *MarketOpen* attribute is set to True) and the supervisor has decided to close the market (the *ClosingMarket* attribute is set to True). When the auctioneer, buyer-administrator, seller-administrator, and accountant go through transition $T1$ towards the exit scene, the update expression on in arc $in1$ toggles the *MarketOpen* attribute from True to False.

6. The state of an electronic institution

The previous sections identified and elaborated the key data structures needed to define an *electronic institution*. In this section we build on that base to provide an account of the execution state of such an institution, where the state of a system is a snapshot of the current values of all observable variables that change over time. Importantly, the definition of the state of a system is required in order to give the precise operational semantics that can be found in the Section 7, through operations such as agents joining a scene or making an utterance, which change the execution state of an institution.

An electronic institution is enacted by agents that interact within, and move between, *instances* of the scenes specified above. Scenes themselves provide blueprints for agent interaction, while *scene instances* refer to the execution of the scene via a set of agents that instantiate the scene's roles and interact. In terms of our theatre metaphor, the scene is the text of the play and the scene instance is a performance of it.

More specifically, a scene instance can be regarded as an extension of the notion of scene, adding variables that represent the state of the interaction of the group of agents participating in that instantiation, together with their history. The state of a scene instance thus requires data structures to record the agent utterances that have taken place in that scene instance. Transition instances similarly capture the state of transitions on execution, and are where agents wait to join other scenes, either because they are waiting for other agents to join them, or because they are waiting for a target scene to reach a place where the agents can join.

In electronic institutions, the same agent may play different roles in different scenes. To represent the possible participation of an agent in multiple scene instances, therefore, we model agents as sets of *agent processes* so that each such

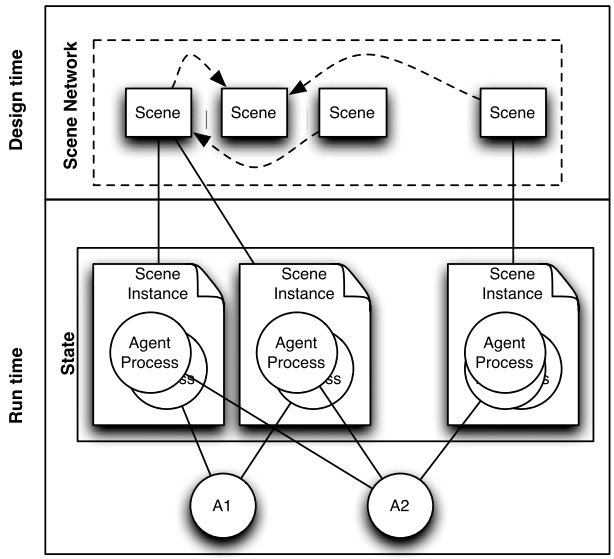

Fig. 11. Agents parenting agent processes within scene instances.

process participates in the execution of a particular scene or transition instance, and maintains information associated with the participation of the agent in that particular scene instance (for example, the role played within it). The state of an agent is thus the collection of its agent processes, as illustrated in Fig. 11. Since we are dealing with open systems, the internal state of an agent is private but the agent's utterances have institutional meaning (for example, a winning bid in an auction implies an obligation to pay) that the institution must maintain.

In this section we account for the three dynamic aspects of institution execution: scene instances (which are created by groups of agents, and disappear when all agents leave them), transition instances (which exist permanently throughout an execution, as we will see) and agent processes (representing the participation of agents in different scene and transition instances). For example, as mentioned above, a scene instance state must record the history of utterances so far in that scene instance, so that a new utterance by an agent participating in that scene instance has a well-defined context in which to be interpreted. Thus, the first element needed is a set of scene and transition instance identifiers to uniquely identity each scene and transition instance. We specify the identifier for the unique entry scene instance and for the exit scene instance for any executing electronic institution.

$$[STId]$$

$$| \quad entryId, exitId : STId$$

In what follows, we specify the data structures needed to support the three previously mentioned dynamic components of the enactment of an institution.

6.1. Agent processes

At run time, agents can concurrently take part in multiple scenes; that is, they may perform several activities at the same time. For example, a human agent may participate in a *Skype* call, in a meeting, and may also send and receive text messages at the same time; similarly, a software agent in an auction house may simultaneously bid in several auctions that run in parallel. Certainly these various activities can be inter-related in the sense that a text message may influence the agent's behaviour in the meeting and Skype call, and the selling price in one auction may influence what an agent chooses to bid in another. In this respect, our model must reflect a set of different processes so that performance in one activity may influence performance in others. From an institutional perspective, this inter-relation occurs within the agent model and has no institutional relevance. However, the institution must account for the actions within each of the activities of the agent, so that an agent process represents an agent playing a single role within an instance of a scene, but the number of such processes may change over time as different instances of scenes are created and terminated.

Formally, in the *AgentProcess* schema (Schema 15), we specify the name of the agent, which we sometimes call the parent agent of the process (line 1), participating in a scene instance (line 2), and instantiating a particular role (line 3) from the set of allowed roles (line 4). The predicate guarantees that the role being played is allowed (line 5). An agent thus comprises the set of processes spawned from it at any specific time.

AgentProcess

$name : AgentConst$	[1]
$instance : STId$	[2]
$role : RoleConst$	[3]
$allowedrole : \mathbb{P}\, RoleConst$	[4]
$role \in allowedrole$	[5]

Schema 15. An agent process consists of the name of the parent agent, an identifier, the role that it is currently playing, and the set of roles it can potentially play.

6.2. Transition instance

While a scene instance is the most natural data structure to introduce next, this relies on the definition of a transition instance first. As discussed earlier, transitions are a fundamental construct of *electronic institutions*. At run time, they *host* agent processes moving from scenes they have just left and before potentially joining other scenes. They allow for the synchronisation of such agent processes of different parent agents that must move *together* into a scene instance, and also allow agent processes to *split* again into more agent processes that then move on to several other scenes. The *TransitionInstance* schema (see Schema 18 below) includes the transition from which it is instantiated (line 1) with a unique identifier (line 2), and has two specific types of agents: *waiting* agents and *exiting* agents which are defined separately in schemas (lines 3 and 4). We define each of these *incrementally* below, because transition instances are conceptually the most sophisticated data structure, and because we seek to clearly identify and specify the distinct but related subsets of agents involved at transition instances.

In the *WaitingAgents* schema (Schema 16), waiting agents are those agents (line 1) that have expressed decisions about the future roles that they want to play in scenes (for example, I may want to be a buyer in a fish market and sell my boat in another market). (Note that we use the term *want* to apply equally to agents with any kind of architecture, but referring to an *expressed decision* that is completely independent of the agent's underlying architecture.) These waiting agents are identified via a mapping, *processwantsnext*, from agent processes to one or more *target* scene instances to play particular roles (line 2). The transition instance must also maintain an ordered list (*queue*) of the bindings of agent processes to variables for each agent process arriving from a scene instance via an arc, realised as a mapping from agent variables to a queue (in order of arrival) of agent constants (line 3). Here, variables are used to mark pathways through the transition; that is, an agent binding an agent variable in an outarc to get into a transition instance can only leave the transition instance through inarcs labelled with that variable. In this sense, transitions act as connectors to synchronise agents following different pathways in the electronic institution. In the predicate part of the schema, we assert that all waiting agents have elected to move to a target scene (line 4), that all such agents play the neutral *guest* role (line 5), and that combining all the agents that appear anywhere within the queue gives the set of waiting processes (line 6).

WaitingAgents

$waitingprocesses : \mathbb{P}\, AgentProcess$	[1]
$processwantsnext : AgentProcess \rightarrow (\mathbb{P}(STId \times RoleConst))$	[2]
$queue : AgentVar \rightarrow \text{seq}\, AgentProcess$	[3]
$\text{dom}\, processwantsnext = waitingprocesses$	[4]
$\forall\, a : waitingprocesses \bullet a.role = guest$	[5]
$waitingprocesses = \{a : AgentProcess \mid (\exists\, as : \text{seq}\, AgentProcess \bullet (as \in (\text{ran}\, queue) \wedge (a \in (\text{ran}\, as)))) \bullet a\}$	[6]

Schema 16. The waiting agent processes at a transition instance.

The *ExitingAgents* schema (Schema 17) describes a further set of distinct agents. It describes those agent processes that are ready to leave a scene after they have declared which specific target scenes they wish to join. In addition, it specifies the set of partner agents that will leave the transition instance together. Whenever a group of waiting agents at a transition instance satisfy the constraints on the outgoing arcs, they are queued into *exitingprocesses* (line 1) until the target scenes are ready for them to enter. The variable *watchtargetscenes* (line 2) keeps, for each target scene instance, a set of sets of processes that want to join that scene instance, through the inarc connecting the transition instance and the target scene instance. Every inarc is labelled with agent variables and roles in disjunctive normal form, as described previously, which explains why the range of this function is a set of sets of agents. The predicate simply states that the exiting processes are those that are *watching* target scenes (line 3). Initially, a transition instance has no processes as specified in Schema 19.

The key points here are that waiting processes are queued according to the variable they were bound to when joining the transitions, and that exiting processes monitor target scenes and wait for them to get to the right conversation place at which they are able to join. For instance, in Fig. 12, agents *a* and *b* came to the transition from scene instance *S*1, while agents *c*, *e* and *f* came from instance *S*2. Agent *a* came before agent *b*, and agent *c* came before *e*. Then, at the transition,

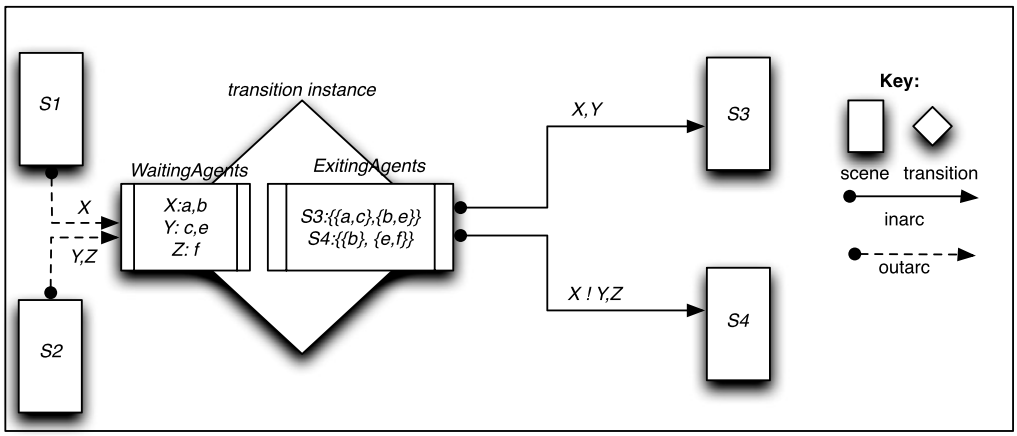

Fig. 12. Transition instance.

ExitingAgents

$$exitingprocesses : \mathbb{P}\, AgentProcess \quad [1]$$
$$watchtargetscenes : STId \pfun \mathbb{P}(\mathbb{P}\, AgentProcess) \quad [2]$$
$$exitingprocesses = \bigcup(\bigcup(\mathrm{ran}\ watchtargetscenes)) \quad [3]$$

Schema 17. The set of exiting agents at a transition instance.

TransitionInstance

$$transition : Transition \quad [1]$$
$$tid : STId \quad [2]$$
$$WaitingAgents \quad [3]$$
$$ExitingAgents \quad [4]$$

Schema 18. A transition instance instantiates a transition, and includes a unique identifier and sets of waiting and exiting agents.

InitTransitionInstance

$$TransitionInstance \quad [1]$$
$$waitingprocesses = \{\} \land exitingprocesses = \{\} \quad [2]$$

Schema 19. The initial state of a transition instance contains no agents.

the agents elected to join $S3$ and $S4$, forming groups of agents ready to be transferred to the scene instances as soon as they reach a state at which the roles (not represented in the figure) they aim to play are authorised to join. In this example, *b* and *e* are waiting together in one group for $S3$, as are *e* and *f* who are waiting together for $S4$.

6.3. Scene instance

The state of a *scene instance* is supported by a data structure that keeps track of the ongoing conversation of agents and, as already mentioned, and illustrated in Fig. 11, many instances of the same scene can be created. In order to define the state of a scene instance, we need a unique identifier, along with a record of the *conversation history*, the history of utterances, and the bindings that took place during each such utterance. This provides a *context* with which to validate every new utterance during the execution of a scene. We therefore define a conversational *history* as the history of *utterance bindings* that have occurred since the creation of the scene instance, where an utterance binding contains the variable *bindings* made during a transition from one conversation place to another such that there is only one time variable that necessarily becomes bound. This data structure supports the possibility of having many concurrent scene instances and managing the movement of agents from scene instances to other scene instances, avoiding deadlock. This is a key feature of the electronic institution architecture.

$$Bindings == \left\{ f : Var \pfun Const \mid \exists_1 t : FreeTimeVar \bullet free(freetvar\ t) \in (\mathrm{dom}\ f) \bullet f \right\}$$
$$UtteranceBindings == ConvPlace \times Bindings \times ConvPlace$$
$$History == \mathrm{seq}\ UtteranceBindings$$

Now, the status of a scene can be identified as being either *open*, when agents are authorised to produce utterances, or *closed*, when agents reach the end of the conversation and are ready to leave, and no more utterances can take place.

$$Status ::= Open \mid Closed$$

We can now move to a formal definition of a scene instance in the *SceneInstance* schema (Schema 20). This contains an identifier (line 1), captures the status of the scene instance (line 2), the conversation history (line 3), the scene of which it is an instance (line 4), the current conversation place (line 5), the set of agent processes involved in the scene instance (line 6), those agents waiting to leave (line 7), and the target transitions they intend to move to (line 8). Moreover, a time counter (which is necessary to support timeout operations in scenes as detailed later in Section 7) records how much time has elapsed since the last successful utterance (successful in the sense that it led to a conversation place transition) (line 9).

The predicates in the *SceneInstance* schema guarantee that the scene instance is consistent with the data structures of its scene and with our concept of electronic institutions in general. In particular, we require the current conversation state to be well defined (within the conversation states of the scene) (line 10), and the number of agents in the scene instance to be within the allowable range of the scene (line 11). All agents waiting to leave must have a transition instance to go to (line 12), when there are no more agent processes in the scene instance the scene status must be closed (line 13), and the exiting processes must be processes in the scene instance (line 14).

SceneInstance

$sid : STId$	[1]
$status : Status$	[2]
$history : History$	[3]
$scene : Scene$	[4]
$place : ConvPlace$	[5]
$sceneprocesses : \mathbb{P}\, AgentProcess$	[6]
$leavingprocesses : \mathbb{P}\, AgentProcess$	[7]
$nexttransitions : AgentProcess \pfun TransitionInstance$	[8]
$timesincelastchange : \mathbb{N}$	[9]
$place \in scene.places$	[10]
$\forall\, r : RoleConst;\ ags : \mathbb{P}\, AgentConst \bullet$ $\#(\{a : sceneprocesses \bullet (a.name, a.role)\} \rhd \{r\}) \in scene.limits\ r$	[11]
$\mathrm{dom}\, nexttransitions = leavingprocesses$	[12]
$sceneprocesses = \{\} \Rightarrow status = Closed$	[13]
$leavingprocesses \subseteq sceneprocesses$	[14]

Schema 20. A scene instance instantiates a scene and contains a status, a history of what has happened, a current position, conversing agents, agents waiting to leave to transitions, and the time since something last happened.

The initial state of a scene instance, represented in *InitSceneInstance*, is such that the status is open (line 2), the initial conversation place is the scene's initial conversation place (line 3), the time since the last utterance is set to 0 (line 4), there are no processes (line 5), and there is no conversational history (line 6) (see Schema 21).

InitSceneInstance

$SceneInstance$	[1]
$status = Open$	[2]
$place = scene.start$	[3]
$timesincelastchange = 0$	[4]
$sceneprocesses = \{\}$	[5]
$history = \langle\rangle$	[6]

Schema 21. An initial scene instance.

6.4. The state of an electronic institution

In consequence of the above, the state of an electronic institution is then given by the set of instances (of both scenes and transitions) and the set of agent processes. The predicates in the *SceneInstances* schema thus state that all scene instances are instantiations of scenes included in the *scene network* (line 6) and that there is a unique entry scene instance and exit scene instance (lines 5, 7 and 8). The *TransitionInstances* and *AgentProcesses* schemas are defined similarly. (See Schemas 22–25.)

SceneInstances

$SceneNetworkSystem$ [1]
$sceneinstances : \mathbb{P}\, SceneInstance$ [2]
$entryinstance : SceneInstance$ [3]
$exitinstance : SceneInstance$ [4]

$\{entryinstance, exitinstance\} \subseteq sceneinstances$ [5]
$\{s : sceneinstances \bullet s.scene\} \subseteq allscenes$ [6]
$entryinstance.sid = entryId$ [7]
$exitinstance.sid = exitId$ [8]

Schema 22. A set of scene instances in an electronic institution.

InitSceneInstances

$SceneInstances$ [1]

$\{entryinstance, exitinstance\} = sceneinstances$ [2]

Schema 23. Initially, a scene instance contains only instances of the exit scene and entry scene.

TransitionInstances

$transitioninstances : \mathbb{P}\, TransitionInstance$ [1]

Schema 24. A set of transition instances in an electronic institution.

InitTransitionInstances

$transitioninstances : \mathbb{P}\, TransitionInstance$ [1]

$\forall\, t : transitioninstances \bullet t \in InitTransitionInstance$ [2]

Schema 25. Initially, transition instances contain no agents.

In the *AgentProcesses* schema, we identify the agent processes that are involved in the institution (line 1), along with their attributes, whose values are modified by the update language (line 2). It is worth recalling that *processes* here refers to all the processes in an institution from which we can define all the agents (*agents*) in an institution at any specific time as those spawning one or more of those processes (lines 3 and 4). (See Schema 26.)

AgentProcesses

$processes : \mathbb{P}\, AgentProcess$ [1]
$agentprocessattributes : AgentProcess \to \mathbb{P}\, Attribute$ [2]
$agents : \mathbb{P}\, AgentConst$ [3]

$agents = \{p : processes \bullet p.name\}$ [4]

Schema 26. A set of agent processes in an electronic institution.

All these elements (*SceneInstances*, *TransitionInstances* and *AgentProcesses*) can then be brought together to define the set of system instances (see Schema 27).

SystemInstances

$SceneInstances$ [1]
$TransitionInstances$ [2]
$AgentProcesses$ [3]

Schema 27. Combined agent, scene and transition instances of an electronic institution.

Finally, in the *ElectronicInstitutionState* schema (Schema 28), the system instances, together with the conversation *history* for each scene instance (line 2), the attributes, *eiattributes*, that relate to the institution as a whole (line 3), and a record of which agents are authorised to play which roles (line 4), determine the electronic institution state. The attributes are intended to store global information, such as the average market price in an auction house, obtained from successful communication between *agent processes*. There is also a consistency check in line 5 that the allowed roles of any process are

ElectronicInstitutionState

$SystemInstances$ [1]

$systemhistory : \mathbb{P}(STId \times History)$ [2]

$eiattributes : \mathbb{P}\, Attribute$ [3]

$authorised : AgentConst \rightarrow (\mathbb{P}\, RoleConst)$ [4]

$\forall\, a : AgentConst;\ p : AgentProcess \mid p.name = a \bullet p.allowedrole = authorised(a)$ [5]

Schema 28. The state of an electronic institution contains all instances, a system history, a set of system attributes, and a record of which agents are authorised to play which roles.

InitialElectronicInstitutionState

$ElectronicInstitutionState$

$InitSceneInstances$

$InitTransitionInstances$

$processes = \{\}$

$systemhistory = \{\}$

$authorised = \{\}$

$entryinstance.scene = entryscene$

$exitinstance.scene = exitscene$

$\forall\, t : transitions \bullet \exists_1\, ti : transitioninstances \bullet t = ti.transition$

Schema 29. Initially, an electronic institution is instantiated with a transition instance for every transition in the electronic institution design and an instance of the entry and exit scenes.

the authorised roles of the parent agent. Initially, the electronic institution state contains an instance of the entry and exit scenes only, together with a transition instance for every transition in the institution design.

Now, with the initial state of an institution additionally defined in *InitialElectronicInstitutionState* (Schema 29), we are ready to give semantics to agent utterances. In the next section, we thus consider the verification of the correctness of utterances, and the changes in the execution state they produce.

7. Operations for electronic institution execution

The specification of an electronic institution in Section 4 defines the interaction rules that shape agent interactions once an electronic institution is computationally enacted. In this section we specify the necessary operations to build any electronic institution and provide an operational semantics for electronic institutions in general.

The main responsibility of operational semantics is to give precise meaning to agent interactions. Thus, the computational model for such interactions is simple: as agents perform valid utterances – those complying with the interaction rules set by the specification – the state of the electronic institution (as described in Section 6) evolves.

This operational semantics must be satisfied by any software infrastructure that monitors agent interactions.

Such infrastructure is not part of the conceptual model but must guarantee that the integrity constraints defined in our model are satisfied during the electronic institution operation. This is to say that the infrastructure has only a programmatic rather than a conceptual meaning; after every operation, it checks that all system integrity constraints are satisfied and updates variable bindings and attribute values if necessary.

The execution of an electronic institution can be regarded as the concurrent execution of its different scenes, which become scene instances at run-time. In this setting, the activity of participating agents amounts to interacting with other agents within different scene instances and moving between them. Agent actions (utterances and moves) cause an electronic institution state to evolve. It is the responsibility of the infrastructure to control the institution execution by guaranteeing that all agent interactions abide by the interaction rules defined by the specification. Hence, the infrastructure must control: the flow of agents (when entering or leaving the institution and moving between scene instances), as well as the execution of scene and transition instances. With this purpose, the infrastructure must employ the electronic institution specification along with its execution state.

At the outset, any institution execution creates an entry scene instance and an exit scene instance, together with a transition instance for every transition specified. Thereafter, agents can enter, exit and participate within scene instances as they are created. The execution of a given scene instance is either a transition between conversation places, or involves agents joining or leaving. In the former case, a transition occurs after either a valid utterance or a pause, and an utterance is regarded as valid whenever it complies with the scene specification in a particular state of the scene instance. This happens whenever the utterance matches one of the labels of the outgoing arcs of the current conversation place and satisfies the constraints associated with the arc. If so, the scene instance changes by moving to a new conversation place and by updating its (conversation) history to incorporate the new bindings produced by the utterance. A transition between conversation places may also occur after a pause, when some specified time elapses. In the latter case, the infrastructure controls the movement of agents entering or leaving a scene instance at access and exit states respectively, without violating the restrictions on the minimum and maximum number of agents per role allowed within a scene instance at any one time.

Since the flow of agents between scene instances is mediated by transitions, agents are required to move to transition instances prior to joining any target scene instances. At this point, the infrastructure must guarantee that an agent within a scene instance can only move to a reachable transition instance for its current role. Moreover, the infrastructure must also control when to allow agents to move from transition instances into scene instances. This requires that the infrastructure considers the types of transitions (*and, or*), the types of arcs connecting scenes at the specification level, and the scene instances in the current electronic institution state. Importantly, when agents follow an arc of type *new* leading to a scene, the infrastructure must bootstrap a new scene instance of the target scene for the agents.

Given this, we are ready to formalise the operations that the infrastructure is required to implement. The structure of this section is thus as follows: Section 7.1 outlines the initialisation of an electronic institution, Section 7.2 describes how to creates instances of a scene, Section 7.3 details agents leaving and joining an institution, and Section 7.4 specifies moving from one conversation place to another within a scene instance. In Section 7.5 we describe agents moving from scenes to transitions, from transitions to scenes, and agents changing their targets while waiting at transitions. Finally, in Section 7.6 we consider how to close down scene instances once they have run their course, before finishing with a summary.

7.1. Initialising an electronic institution

The first operation an infrastructure must perform is the bootstrapping of an electronic institution. When bootstrapping, the running electronic institution is endowed with an initial state as described in Section 6 where an entry scene instance, an exit scene instance, and a transition instance for each transition specified at design, are created. Thereafter, agents can join in to start their interactions and hence cause the electronic institution state to evolve. We refer to the running electronic institution and its current electronic institution state with the following variables:

$$ei : ElectronicInstitution$$
$$eis : ElectronicInstitutionState$$

When an operation is performed, the state of the institution changes from that captured by the undashed variable to that captured by the dashed variable, but the institution itself does not change (see Schema 30).

$\Delta ElectronicInstitutionState$

$ElectronicInstitutionState$	[1]
$ElectronicInstitutionState'$	[2]
$\Xi ElectronicInstitution$	[3]

Schema 30. A change of state to an electronic institution does not impact on the design.

$StartElectronicInstitution$

$\Delta ElectronicInstitutionState$	[1]
$InitialElectronicInstitutionState'$	[2]
$attributes? : \mathbb{P}\, Attribute$	[3]
$eiattributes' = attributes?$	[4]

Schema 31. When an electronic institution is first instantiated, the user supplies a set of system attributes.

The *StartElectronicInstitution* schema (Schema 31) outlines how to start an electronic institution. This initialising operation bootstraps the electronic institution state from the specification of its structure (line 1), and amounts to creating an instance for the entry and exit scenes along with an instance for each transition (line 2). Moreover, the operation may receive as input the values for all of the attributes of the institution (line 3), in which case the values of the electronic institution attributes are set accordingly (line 4).

7.2. Creating a scene instance

In order to create a scene instance in the initial state as described previously, we simply need to provide the scene from which it must be instantiated along with a unique identifier. Schema 32 takes as input an identifier and a scene (in lines 1 and 2), and creates a scene instance in its initial state (line 3) with these input values (lines 4 and 5).

$CreateSceneInstance$

$sid? : STId$	[1]
$scene? : Scene$	[2]
$InitSceneInstance'$	[3]
$sid' = sid?$	[4]
$scene' = scene?$	[5]

Schema 32. Creation of a scene instance.

7.3. Joining and leaving an institution

Once an institution is up and running, agents can request access to begin their interactions. This service is provided by the operation specified in the *RequestAccess* schema (Schema 33). An agent (line 2) can request to access a running electronic institution (line 1) playing a given set of roles (line 3). If access is granted, the agent is authorised to play the requested roles (line 4). Note that more sophisticated role-based access control methods can be implemented by extending and refining this operation.

RequestAccess

$\Delta ElectronicInstitutionState$ [1]
$agent? : AgentConst$ [2]
$roles? : \mathbb{P}\, RoleConst$ [3]

$authorised' = authorised \oplus \{(agent?, roles?)\}$ [4]

Schema 33. An agent requests to be authorised to take on another role.

Once a set of roles is authorised in a running institution, an agent (line 1) can request to join an institution, as specified by the *JoinInstitution* schema (Schema 34), in which there is a change in the electronic institution state (line 3) and in a scene instance (line 2). The first predicate (line 4) states that it is the *entry* scene instance that is changing state. Next we check that there is no other process that has the joining agent as a parent, in which case the set of processes within the entry scene is updated to include a new process that has the input agent as the name of the agent, has an identifier equal to the entry scene instance, a set of allowed roles as specified in the previous schema and defined by the variable *authorised*, and has its role set to *guest*.[12]

JoinInstitution

$ag? : AgentConst$ [1]
$\Delta SceneInstance$ [2]
$\Delta ElectronicInstitutionState$ [3]

$sid = entryId$ [4]
$\neg\, (\exists\, a : processes \bullet a.name = ag?)$ [5]
$processes' = processes \cup$ [6]
$\{(\mu\, a : AgentProcess \mid a.role = guest;\ a.allowedroles = authorised(ag?);$
$a.instance = entryId;\ a.name = ag?)\}$
$agents' = agents \cup \{ag?\}$ [7]

Schema 34. An agent joins an institution and is initially assigned a *guest* role.

An agent can only leave an institution from the exit scene instance. The operation defined by the *LeaveInstitution* schema (Schema 35) checks whether the agent process requesting to leave (line 1) is not spawned by an agent that is also spawning another process taking part in any other scene instance (line 6). If so, then the agent process is removed from the set of agent processes (line 7) and agent names (line 8) in the institution.

LeaveInstitution

$ap? : AgentProcess$ [1]
$\Delta SceneInstance$ [2]
$\Delta ElectronicInstitutionState$ [3]

$sid = exitId$ [4]
$ap? \in sceneprocesses$ [5]
$\neg\, (\exists\, si : sceneinstances;\ ap : processes \mid si.sid \neq exitId \wedge ap.name = ap?.name \bullet ap? \in si.sceneprocesses)$ [6]
$processes' = processes \setminus \{ap?\}$ [7]
$agents' = agents \setminus \{ap?.name\}$ [8]

Schema 35. An agent can request to leave an institution as long as it is not involved in any scene instances.

[12] Recall from Section 4 that an agent that joins an institution is given the special role *guest* until it states the roles it wants to play and its credentials to do so are checked. (As before, note we use the term *want* to apply equally to agents with any kind of architecture, simply referring to an expressed decision.)

Table 5
An Example of a *Line* in a Dutch auction.

pattern = {	force = commit;
	dlformula = bid(**!**good, **!**current_price);
	sender = **?**agentX; receiver = **!**agentY;
	sendrole = bidder; receiverole = auctioneer}
constraints =	**?**agentX.credit $\geqslant$ **!**current_price
updates =	**?**agentX.credit = **?**agentX.credit − **!**current_price

7.4. Processing an utterance

Processing utterances is the central operation of an institution because it effectively processes agent actions (utterances). To understand this operation, recall from Section 4 that according to the specification of a scene, links between conversation places in a scene correspond to actions. An action in this context is either a *Line*, an utterance template with preconditions and postconditions, or a *Pause*, which models silence (no utterances) for a period of time. Thus, each *line* in a scene specifies the rules that an utterance performed by an agent must fulfill at run-time to enable a scene instance to move from one conversation place to another. Recall also from Section 6 that a distinguishing feature of a scene instance is that it keeps track of the conversation history – the context of the conversation that has taken place so far involving agents within the scene instance. Given these two elements – design (the structure of scenes as created at design time by the institution designer) and context (generated at run-time by the complete history of agents' conversing within a scene) – we can consider how to process an utterance.

At run-time, utterances are performed only by agents within a scene instance. Each time an agent performs an utterance at a given conversation place, the infrastructure checks whether the utterance is valid, and hence causes the scene instance to move from one conversation place to another. An utterance is valid if it unifies with the pattern (an utterance template) of an outgoing *line* from the conversation place and satisfies the constraints within that *line*. In order to test whether an utterance is valid – which amounts to checking unification and constraint satisfaction – the infrastructure employs the conversation history, providing the context to interpret the rules of the institution in order to test whether utterances are valid or not. In what follows we detail how to implement the processing of an utterance.

Building on our earlier example, we consider a scene that specifies a Dutch auction protocol where an auctioneer calls out prices downwards. Here, the auctioneer clears the auction when the price called is either accepted by a bidder or reaches the reservation price. The conversation place that the auctioneer reaches after calling out some price must be connected to another conversation place that gives agents the chance to bid at the current offer price. The *line* linking these two conversation places might look like that in Table 5, the intended semantics of which is that any bidder is allowed to bid the current price called by the auctioneer for the good at auction. This is what the pattern (comprising an *UtteranceTemplate*) specifies. However, only bidders whose credit is higher than the current offer price are allowed to bid (as specified by *constraints*). If a bid is accepted, the bidder's credit is decreased accordingly (as specified by *updates*).

Given this example, we now formalise the steps required to process an utterance. First, we define an *Utterance* as an *UtteranceTemplate* for which all variables are bound (see Schema 36). The *Utterance* schema has the same signature as an *UtteranceTemplate*, but here all the participants are constants (line 6) playing concrete roles (line 7) at a specific time (line 8) and uttering a grounded term (line 9). Recall that *agentconst* is a mapping from the type *AgentConst* to *AgentTerm*, so line 6 states that sender and receiver are of type *AgentConst*. Similarly, line 7 states that *sendrole* and *receiverole* are of type *RoleConst*. As an example of an *Utterance*, consider a bid issued by a participant in a Dutch auction as in Table 6.[13]

Utterance

$$force : IllocutionaryForce \quad [1]$$
$$dlformula : DLFormula \quad [2]$$
$$time : TimeTerm \quad [3]$$
$$sender, receiver : AgentTerm \quad [4]$$
$$sendrole, receiverole : RoleTerm \quad [5]$$

$$\{sender, receiver\} \subseteq (\text{ran } agentconst) \quad [6]$$
$$\{sendrole, receiverole\} \subseteq (\text{ran } roleconst) \quad [7]$$
$$time \in (\text{ran } timeconst) \quad [8]$$
$$dlformula \in languages.grounddomainontology \quad [9]$$

Schema 36. An utterance contains only ground terms.

As mentioned above, to check the validity of an utterance, the infrastructure employs the conversation history provided by *lines*. In general, the infrastructure considers all *lines* from the current conversation place and selects each of the utterance

[13] For the sake of simplicity we do not consider time in our example.

Table 6
Example of an *utterance* in a Dutch auction.

Utterance = {	force = commit;
	dlformula = bid(cod, 10 EUR);
	sender = JohnDoe; receiver = JohnSmith;
	sendrole = bidder; receiverole = auctioneer}

Table 7
Example conversation history of a Dutch auction.

good	= cod
current_price	= 10 EUR
agentY	= JohnSmith

Table 8
Example of a free utterance pattern in a Dutch auction.

FreeUtterancePattern = {	force = commit;
	dlformula = bid(cod, 10 EUR);
	sender = ?agentX; receiver = JohnSmith;
	sendrole = bidder; receiverole = auctioneer}

templates (patterns) contained in each of these *lines*. Next, it applies the history of bindings to the patterns by replacing bound variables in the patterns (variables preceded with **!**) with the most recent value from the history. As a result, it obtains a set of *free utterances* that only contain constants and free variables, and we call this a *FreeUtterancePattern* (see Schema 37).

$$
\begin{array}{|l|}
\hline
FreeUtterancePattern \\
\hline
UtteranceTemplate \qquad [1] \\
\hline
sender \in (\text{ran}\ agentvar) \Rightarrow (agentvar^{\sim})(sender) \in FreeAgentVar \qquad [2] \\
receiver \in (\text{ran}\ agentvar) \Rightarrow (agentvar^{\sim})(receiver) \in FreeAgentVar \qquad [3] \\
sendrole \in (\text{ran}\ rolevar) \Rightarrow (rolevar^{\sim})(sendrole) \in FreeRoleVar \qquad [4] \\
receiverole \in (\text{ran}\ rolevar) \Rightarrow (rolevar^{\sim})(receiverole) \in FreeRoleVar \qquad [5] \\
\hline
\end{array}
$$

Schema 37. A free utterance pattern contains agent and role terms which are all free (not bound).

To illustrate the generation of a *FreeUtterancePattern*, suppose that in our example the conversation history (the history of bindings) at the current conversation place is that of Table 7. After applying the conversation history to the *line* in Table 5, the infrastructure obtains the free utterance pattern of Table 8. Notice that this free utterance pattern states that any bidder is allowed to bid, but it is constrained to bid 10 EUR for *cod*, which is the current price and current good at auction according to the conversation history. Observe that the free utterance pattern in Table 8 and the utterance in Table 6 *can be unified*, thus generating the new binding **?**agentX = JohnDoe, which the infrastructure adds to the conversation history in Table 7. Now, given this matching between utterance and utterance template in the *line* in Table 5, if the constraints in the *line* hold (namely, if JohnDoe's credit is larger than 10 EUR), the infrastructure can deem the utterance as valid. This triggers the update of the current conversation place along with the updates specified by the *line* in Table 5. In our example, JohnDoe's credit is reduced by the 10 EUR the bidder committed to pay for the good.

To generalise this, the sequence of operations required of the infrastructure once an agent performs an utterance within a scene instance at the current conversation place is as follows.

1. Collect all lines from the current conversation place.
2. Select the utterance template (pattern) of each *line*.
3. Apply the conversation history (history of bindings) to the selected utterance templates by replacing bound variables in the schemas (variables preceded with **!**) with their most recent bound value. This operation obtains a set of *free utterances* that only contain constants and free variables (corresponding to the *FreeUtterancePattern* above).
4. Match and obtain the bindings unifying the agent's utterance with each of the free utterances found above. If there is a single free utterance matching the utterance, go to the next step, otherwise an error occurs.
5. Update the conversation history with the bindings obtained in the previous step.
6. If the constraints in the *line* are satisfied, then the action can take place, otherwise no conversation place change occurs, the history reverts to its initial value, and an error message is returned. If the action is successful, the updates of the *line* apply.

Given this, we introduce a collection of functions that help us to elaborate a schema for the operation that processes utterances, based on the sequence of events outlined above. The axiomatic schema below thus includes the following definitions: a function to apply a history to an utterance pattern to obtain a *FreeUtterancePattern* (line 1); a partial function to apply bindings to an *UtteranceTemplate* (line 2) that can only be applied if the result is a bound utterance; a predicate that holds between a *FreeUtterancePattern* and an *Utterance* when there exist bindings to unify them (line 3); a function that, given a history and a set of utterance patterns, replaces all bound variables in those patterns by tracing back through the history to find the last recorded binding of that bound variable, and returns a set of *FreeUtterancePatterns* (line 4); a predicate that holds when a set of constraints are satisfied by a conversation history (line 5); and a function that takes a conversation history and applies it to a sequence of updates (line 6).

$$
\begin{array}{ll}
applyhistory : History \to UtteranceTemplate \to FreeUtterancePattern & [1] \\
applybindings : Bindings \to UtteranceTemplate \to Utterance & [2] \\
matches\_ : \mathbb{P}(FreeUtterancePattern \times Utterance \times Bindings) & [3] \\
replaceboundvariables : History \nrightarrow (\mathbb{P}\ UtteranceTemplate) \nrightarrow (\mathbb{P}\ FreeUtterancePattern) & [4] \\
satisfied\_ : \mathbb{P}((\mathbb{P}\ CLFormula) \times History) & [5] \\
ApplyHistoryAL : History \nrightarrow (\text{seq}\ ULFormula) \nrightarrow (\text{seq}\ ULFormula) & [6] \\
\hline
\forall f : FreeUtterancePattern;\ u : Utterance;\ bs : Bindings \bullet & \\
\quad matches(f, u, bs) \Leftrightarrow (\exists_1\ bs : Bindings \bullet applybindings\ bs\ f = u) & [7]
\end{array}
$$

Using these definitions we can specify how to process an *Utterance* through the *Speak* schema (see Schema 38). In fact, we specify what happens when an *Utterance*, *u*?, is uttered by an agent, *agent*?. The *Speak* schema includes: a change of conversation place of the scene instance (line 1); the uttering agent (line 2); the utterance (line 3); a variable identifying the next potential actions (*lines* or *pauses*) from the current conversation place (line 4); the set of utterance patterns that label the *lines* (line 5); the set of free utterance patterns that arise once the conversation history has been applied to these patterns (line 6); the bindings that enable the utterance to be matched to one, and only one, of the available free utterance patterns (line 7); the specific free utterance pattern that can be matched (line 8); the chosen *line* that contains the open utterance pattern (line 9); the new history resulting after the utterance is made (line 10); the utterance pattern of the chosen *line* (line 11); the constraints of the *line* (line 12); the updates of the *line* (line 13); and the ground updates that result from applying the conversation history to the constraints (line 13).

In the predicate part of the schema: the potential next actions are defined by using a domain restriction applied to the set of all *lines* whose origin is in the current conversation place (line 14); the available utterance patterns can be retrieved

Speak

$$
\begin{array}{ll}
\Delta SceneInstance & [1] \\
agent? : AgentConst & [2] \\
i? : Utterance & [3] \\
nextmoves : ConvPlace \leftrightarrow ConvPlace & [4] \\
availableutterancepatterns : \mathbb{P}\ UtteranceTemplate & [5] \\
availableFreeUtterancePatterns : \mathbb{P}\ FreeUtterancePattern & [6] \\
bindings : Bindings & [7] \\
chosenFreeUtterancePattern : FreeUtterancePattern & [8] \\
chosenmove : ConvPlace \times ConvPlace & [9] \\
newhistory : History & [10] \\
chosenmovelabel : Action & [11] \\
constraints : \mathbb{P}\ CLFormula & [12] \\
updates, groundupdates : \text{seq}\ ULFormula & [13] \\
\hline
nextmoves = \{place\} \lhd scene.moves & [14] \\
availableutterancepatterns = & \\
\quad \{m : nextmoves;\ is : UtteranceTemplate \mid is = (line^{\sim}(scene.link(m))).pattern \bullet is\} & [15] \\
availableFreeUtterancePatterns = replaceboundvariables\ history\ availableutterancepatterns & [16] \\
\exists_1\ fi : availableFreeUtterancePatterns \bullet (\exists\ bs : Bindings \bullet matches(fi, i?, bs)) & [17] \\
matches(chosenFreeUtterancePattern, i?, bindings) & [18] \\
chosenmove = (\mu\ m : nextmoves \mid matches((line^{\sim}(scene.link(m))).pattern, i?, bindings)) & [19] \\
newhistory = history \frown \langle(first(chosenmove), bindings, second(chosenmove))\rangle & [20] \\
constraints = (line^{\sim}(scene.link(chosenmove))).constraints & [21] \\
satisfied(constraints, newhistory) & [22] \\
place' = second(chosenmove) & [23] \\
updates = (line^{\sim}(scene.link(chosenmove))).updates & [24] \\
groundupdates = ApplyHistoryAL\ history'\ updates & [25] \\
(\text{ran}\ groundupdates) \subseteq languages.groundupdatelanguage & [26]
\end{array}
$$

Schema 38. An agent speaks an utterance.

by applying the inverse of the injective function to each of the potential next *lines* (line 15); the set of available open utterance patterns is found by applying the conversation history to the available utterance patterns (line 16); there exists one, and only one, free utterance pattern in the available free utterance patterns for which there can be found bindings that match the utterance (line 17); we set the variable *bindings* to be equal to the bindings discovered in this process (line 18); we select the chosen *line* that contains the free utterance pattern that can be matched with the agent's utterance (line 19); we calculate the new conversation history by concatenating the chosen free utterance pattern and bindings (line 20); we make the variable *constraints* equal to the constraints of the chosen *line* (line 21); we check that the constraints of the *line* are satisfied (line 22); we update the current conversation place as defined by the *line* (line 23); we apply the new conversation history as calculated above to the sequence of update formulae that are contained in the *line* (line 24); and we check that these updates are ground (so that they can then be successfully applied) (lines 25, 26).

The ground updates then affect the properties of the roles of participating agents, of the scene instance and also, in general, of the electronic institution state.

Transitions between conversation places can also be caused by pauses. For example, suppose that there is a pause at the current conversation place. When the elapsed time, without any agent uttering a valid utterance, is greater than the pause time, the scene instance evolves towards a new conversation place as specified by the *Pause* schema (see Schema 39).

$Timeout$

$\Delta SceneInstance$	[1]
$\exists\, m : (ConvPlace \times ConvPlace) \mid$ $first(m) = place \wedge$ $timesincelastchange > (pause^{\sim}(scene.link(m))).time \bullet place' = second(m)$	[2]

Schema 39. A timeout takes place.

7.5. Moving between scenes and transitions

Now that we have specified how utterances are processed, we can detail the operations that allow *agent processes* to move between scene instances. At run-time an agent can either leave a scene instance to join a transition instance, or leave a transition instance to join a scene instance. Thus, agent processes travel between scene instances through transition instances. Importantly, agents can choose which scene instances and transition instances to join, with agent processes being able to change these choices in a way that preserves agent autonomy. Choices are conditioned by the institution itself, which specifies the connections between scenes and transitions, and the availability of running scene and transition instances. In this context, we must specify operations that allow groups of agent processes to move in and out of scene instances (that is, that allow group actions), but supporting agent autonomy and group actions in environments in which there are multiple activities (scene instances) is the main source of complexity.

7.5.1. From scene to transition

In general, an agent process intending to leave a scene instance must indicate (choose) a target transition instance to move to. With this aim, each scene instance keeps a record of the transition instances that each process requests to move to, through the *nexttransitions* function, as specified by the *SceneInstance* schema in Section 6. Building on this, the *MovingFromSceneInstanceToTransitionInstance* schema specifies how to move a group of agent processes from a scene instance to a transition instance, as follows. (See Schema 40.) The set of agent processes is moved from a scene instance into a transition instance provided that all agent processes: are in the scene instance (line 7); have requested to move to the transition instance (line 8); can leave at the current conversation place (line 9); can simultaneously follow an arc linking the scene instance with the transition instance (specified in the remainder of the schema). The final pre-condition deserves further explanation; it states that there is an arc, *i*, connecting the two instances (line 6), whose label contains a normal form of agent variables and role identifiers (line 10) that can be bound to the agent processes intending to exit the scene instance (line 11). If these preconditions hold, then the agent processes: are added to the queue of agent processes in the transition instance (line 14); and are removed from the set of agent processes in the scene instance (line 15).

Though not specified here (for reasons of simplicity of exposition), this new state of the scene instance must satisfy any constraints specified in the design of the parent scene, such as the minimum number of agents allowed to play various roles within the scene.

7.5.2. From transition to scene

Now we consider an agent moving to a scene from a transition. First, we specify an operation that enables a group of agents to leave a transition instance. The purpose of this operation is to assess, for a given transition instance, the group of agents with satisfiable targets (scene instances) that are enabled to join the scene instances they have requested. For both types of transitions (*and* and *or*), the enabling operation proceeds by: assessing the agent processes that are *waiting* to leave; removing them from the list of agent processes *waiting* to leave the transition instance; and adding them to the list of agents now *enabled* to leave the transition. Thus, the operation changes the state of a transition instance by moving

$MovingfromSceneInstanceToTransitionInstance$

$\Delta SceneInstance$ [1]
$\Delta TransitionInstance$ [2]
$\Delta ElectronicInstitutionState$ [3]

$\exists s : SceneInstance;\ t : TransitionInstance;\ i : OutArc;\ as : \mathbb{P}\, AgentProcess;$
$con : \mathbb{P}(AgentVar \times RoleConst) \mid$ [4]
$(\theta SceneInstance = s \land \theta TransitionInstance = t \land$ [5]
$i.transition = t.transition \land i.scene = s.scene \land$ [6]
$as \subseteq s.sceneprocesses \land$ [7]
$(\forall a : as \bullet nexttransitions(a) = t) \land$ [8]
$(\forall a : as \bullet place \in (i.scene.leaving(a.role))) \land$ [9]
$con \in disjnorm(i) \land$ [10]
$(\forall r : RoleConst \bullet \#\{a : as \mid a.role = r\} = \#(con \rhd \{r\}))) \bullet$ [11]
$(\forall ap : AgentProcess;\ a : AgentVar;\ r : RoleConst \mid$ [12]
$((a, r) \in con) \land (ap.role = r) \bullet$ [13]
$(queue' = queue \oplus \{a \mapsto (queue(a) \frown \langle ap \rangle)\})) \land$ [14]
$sceneprocesses' = sceneprocesses \setminus as$ [15]

Schema 40. A group of agent processes is moved from a scene instance to a transition instance by the electronic institution infrastructure.

some agent processes from the *waiting agents queue* to the *exiting agents queue*. Once a group of agent processes is enabled to leave a transition instance, and they are in the exiting agents queue, they are ready to join their target scene instances whenever these scene instances can let them in. In some sense, the transition instance has finished all the processing it needs to do.

The *EnableAgentsToLeaveOrTransition* and *EnableAgentsToLeaveAndTransition* schemas below specify the enabling operation, while the *MoveAgentFromTransitionToScene* schema details how to move agent processes from transition instances to scene instances. Prior to formalising these operations, however, we introduce some auxiliary functions. Recall that each transition instance keeps, for each variable, a queue of agents that came into the transition instance bound to that variable (the first to come in through that variable would be first in the queue, and so on). The *leadagent* function (line 1) takes a transition instance and a variable within this queue and returns the first agent in the queue. There are many ways in which this function can be implemented but, for now, we simply provide the signature of the function that describes this operation. Second, the *remove* function (line 2) eliminates an agent process from a sequence of agent processes. Finally, the *changerole* function (line 3) overrides the current role of an agent process with a new role.

leadagent : *TransitionInstance* → *AgentVar* → *AgentProcess* [1]
remove : seq *AgentProcess* → *AgentProcess* → seq *AgentProcess* [2]
changerole : *AgentProcess* → *RoleConst* → *AgentProcess* [3]

The next auxiliary function we require is the *agentsleave* function, specified below, which moves agents from the waiting queue to the exiting queue. The function looks for the group of agent processes that are allowed to leave a transition instance through a set of (in)arcs leaving that transition. This amounts to checking whether, on entering the transition, the agent processes were bound to all the variables in at least one conjunct of the disjunctions labelling all the outarcs that must be traversed (all for an *and*, one for an *or*) (lines 4, 5). If so, the predicate builds and returns a new transition instance (created in line 3) that removes from the waiting list the agent processes that are allowed to leave (line 6), in order to add them to the exit list (line 7). Moreover, the operation also updates the queue of agents bound to each variable in the transition instance (line 8) by removing those agent processes that have left the waiting list to join the exiting list. This function is instrumental in defining the semantics of the operation of agents switching queues at both *or* and *and* transitions which we specify next.

$agentsleave : TransitionInstance \to \mathbb{P}\, InArc \to \mathbb{P}\, AgentVar \to TransitionInstance$ [1]

$\forall t : TransitionInstance;\ arcs : \mathbb{P}\, InArc;\ agents : \mathbb{P}\, AgentVar \bullet$ [2]
$agentsleave\ t\ arcs\ agents = (\mu\, new : TransitionInstance \mid$ [3]
$(\exists con : \mathbb{P}(AgentVar \times RoleConst) \bullet (con \in (\bigcup \{arc : arcs \bullet (ei.disjnorm)(arc)\})) \land$ [4]
$(\forall a : AgentVar;\ r : RoleConst \mid (((a, r) \in con) \land (r \in ((leadagent\ t\ a).allowedroles))) \bullet$ [5]
$new.waitingprocesses = t.waitingprocesses \setminus$ [6]
$\{ap : AgentProcess;\ av : AgentVar \mid av \in agents \land ap = (leadagent\ t\ av) \bullet ap\} \land$
$new.exitingprocesses = t.exitingprocesses \cup$ [7]
$\{ap : AgentProcess;\ av : AgentVar \mid av \in agents \land ap = (leadagent\ t\ av) \bullet changerole\ ap\ r\} \land$
$new.queue = t.queue \oplus$ [8]
$\{a : AgentVar \mid a \in agents \bullet (a \mapsto (remove\ (t.queue(a))\ (leadagent\ t\ a)))\})) \bullet new)$

The *EnableAgentsToLeaveORTransition* schema (Schema 41) specifies how to enable a group of agent processes to leave an *or* transition instance. They can do so provided that they have chosen to leave through at least one of the inarcs of the

transition instance hosting them. Schema 41 checks that the transition instance is of type *or* (line 9), and assesses all the inarcs of the transition instance (line 7), along with the variables labelling them (line 8). Then, the schema employs the *agentsleave* function using, as arguments, the transition instance, the set of all inarcs and the set of all variables (line 10). This assesses a new transition instance whose exit list contains the group of agents allowed to leave and whose waiting list contains the agents that are pending to leave.

EnableAgentsToLeaveORTransition

$\Delta\, TransitionInstance$	[1]
$old, new : TransitionInstance$	[2]
$allagentvars : \mathbb{P}\, AgentVar$	[3]
$allinarcs : \mathbb{P}\, InArc$	[4]
$old = \theta\, TransitionInstance$	[5]
$new = \theta\, TransitionInstance'$	[6]
$allinarcs = ei.transitioninarcs\ old.transition$	[7]
$allagentvars = \bigcup\{a : allinarcs \bullet ei.arcagentvars(a)\}$	[8]
$old.transition.mode = or$	[9]
$new = agentsleave\ old\ allinarcs\ allagentvars$	[10]

Schema 41. A set of agents is enabled to leave an *or* transition instance.

The *EnableAgentsToLeaveAndTransition* schema specifies how to enable a group of agent processes to leave an *and* transition instance, which they are allowed to do if they have chosen to leave through each one of the inarcs of the transition instance hosting them. (See Schema 42)

EnableAgentsToLeaveAndTransition

$\Delta\, TransitionInstance$	[1]
$old, new : TransitionInstance$	[2]
$allagentvars : \mathbb{P}\, AgentVar$	[3]
$allarcs : \mathbb{P}\, Arc$	[4]
$old = \theta\, TransitionInstance$	[5]
$new = \theta\, TransitionInstance'$	[6]
$allarcs = ei.transitioninarcs\ old.transition$	[7]
$allagentvars = \bigcup\{a : allarcs \bullet ei.arcagentvars(a)\}$	[8]
$old.transition.mode = and$	[9]
$\exists\, try : TransitionInstance \bullet (\forall\, i : InArc \mid i \in allarcs \bullet agentsleave\ old\ \{i\}\ allagentvars = try)$	[10]

Schema 42. A set of agents is enabled to leave an *and* transition instance.

Now we can specify the operation that moves agent processes from transition instances to scene instances. This is given by the *MoveAgentFromTransitionToScene* schema (see Schema 43), which takes as input a transition instance (line 3) and a scene instance (line 2). Recall that after a valid utterance within a scene instance, the conversation state changes, and that

MoveAgentFromTransitionToScene

$\Delta ElectronicInstitutionState$	[1]
$sceneinstance? : SceneInstance$	[2]
$transitioninstance? : TransitionInstance$	[3]
$\Delta SceneInstance$	[4]
$\Delta\, TransitionInstance$	[5]
$accessibleroles : \mathbb{P}\, RoleConst$	[6]
$\theta SceneInstance = sceneinstance?$	[7]
$\theta\, TransitionInstance = transitioninstance?$	[8]
$accessibleroles = \{r : RoleConst \mid place \in (sceneinstance?.scene).access(r) \bullet r\}$	[9]
$(\exists\, ap : AgentProcess \mid ap \in transitioninstance?.exitingprocesses \bullet$	[10]
$\quad (ap.role \in accessibleroles) \wedge$	[11]
$\quad (\#\{a : AgentProcess \mid a \in sceneinstance?.sceneprocesses \wedge a.role = ap.role \bullet ap\} + 1) \in$	[12]
$\quad\quad (sceneinstance?.scene).limits(ap.role) \Rightarrow$	[13]
$\quad\quad ((exitingprocesses' = exitingprocesses \setminus \{ap\}) \wedge$	[14]
$\quad\quad (sceneprocesses' = sceneprocesses \cup \{ap\})))$	[15]

Schema 43. A set of agents moves from a transition instance to a scene instance.

each conversation state can only be accessed by some (and possibly no) roles. Therefore, after the performance of a valid utterance within a scene instance, the infrastructure must check whether the new conversation state can be accessed by some agent processes waiting to join the scene instance from some transition instance. With this aim, the operation first assesses the roles that can access the current conversation place (line 9). If there is a group of agent processes waiting at the exit agents list of the transition instance (line 10) to join the scene instance, and playing roles that can be let in at the current conversation state (line 11), then the agent processes: are removed from the transition instance (line 14); and are added to the scene instance's list of agent processes (line 15). Both steps can be performed whenever the limit on agent processes per role with the scene instance is not exceeded (lines 12, 13). In practise, in order to move all agents waiting at a transition instance, the infrastructure must repeat the operation below until it cannot find any agent process that is enabled to leave.

7.5.3. Changing targets at transitions

The final set of operations needed are concerned with an agent changing its target scenes at a transition instance. More specifically, an agent within a transition instance can request to leave it in order to join some new scene instance by playing a particular role. We refer to this combination of scene and role as a *target*, and an agent can request multiple targets simultaneously. However, the interpretation of such targets depends on the type of transition hosting the agent: within an *or* transition, this indicates that an agent wants to join *some* target scenes, whereas within an *and* transition, it indicates that it wants to join *all* target scenes. Importantly, an agent's targets must be consistent with the type of transition, as follows. They are consistent with an *or* transition if any target scene instance is reachable from the transition instance with the agent in the target role (via an arc labelled with the role). Conversely, they are consistent with an *and* transition if all targeted scene instances are reachable from the transition instance. Below we define general predicates, *consistentORtargets* and *consistentANDtargets*, to check whether the targets of a group of agents are consistent with an *or* transition and with an *and* transition, respectively.

$$consistentORtargets\_ : \mathbb{P}(TransitionInstance \times AgentProcess \times \mathbb{P}(SceneInstance \times RoleConst))$$

$$\begin{array}{ll}
\forall\, ti : TransitionInstance;\ ag : AgentProcess;\ targets : \mathbb{P}(SceneInstance \times RoleConst) \bullet & [1]\\
\quad consistentORtargets(ti, ag, targets) \Leftrightarrow & [2]\\
\qquad ti.transition.mode = or \wedge & [3]\\
\qquad (\exists\, i : InArc;\ target : targets \bullet & [4]\\
\qquad\quad i \in ei.transitioninarcs(ti.transition) \wedge & [5]\\
\qquad\quad (first(target)).scene = i.scene \wedge & [6]\\
\qquad\quad (first(target)).sid \in \{s : STId \mid s \in \mathrm{dom}(ti.processwantsnext(ag)) \bullet s\} \wedge & [7]\\
\qquad\quad (\exists\, con : \mathbb{P}_1(AgentVar \times RoleConst);\ a : AgentVar;\ r : RoleConst \bullet & [8]\\
\qquad\qquad (con \in ei.disjnorm(i)) \wedge ((a, r) \in con) \wedge & [9]\\
\qquad\qquad ag \in \mathrm{ran}(ti.queue(a)) \wedge & [10]\\
\qquad\qquad r \in ag.allowedroles \wedge & [11]\\
\qquad\qquad r = second(target))) & [12]
\end{array}$$

$$consistentANDtargets\_ : \mathbb{P}(TransitionInstance \times AgentProcess \times \mathbb{P}(SceneInstance \times RoleConst))$$

$$\begin{array}{ll}
\forall\, ti : TransitionInstance;\ ag : AgentProcess;\ targets : \mathbb{P}(SceneInstance \times RoleConst) \bullet & [1]\\
\quad consistentANDtargets(ti, ag, targets) \Leftrightarrow & [2]\\
\qquad ti.transition.mode = and \wedge & [3]\\
\qquad (\forall\, t : targets \bullet consistentORtargets(ti, ag, \{t\})) \wedge & [4]\\
\qquad (\forall\, i : InArc \bullet & [5]\\
\qquad\quad (\exists\, tar : \mathbb{P}(SceneInstance \times RoleConst) \bullet tar \subset targets \wedge tar \neq \varnothing \wedge & [6]\\
\qquad\quad (\forall\, target : tar \bullet & [7]\\
\qquad\qquad i \in ei.transitioninarcs(ti.transition) \wedge & [8]\\
\qquad\qquad (first(target)).scene = i.scene \wedge & [9]\\
\qquad\qquad (first(target)).sid \in \{s : STId \mid s \in \mathrm{dom}(ti.processwantsnext(ag)) \bullet s\} \wedge & [10]\\
\qquad\qquad (\exists\, con : \mathbb{P}_1(AgentVar \times RoleConst);\ a : AgentVar;\ r : RoleConst \bullet & [11]\\
\qquad\qquad\quad (con \in ei.disjnorm(i)) \wedge ((a, r) \in con) \wedge & [12]\\
\qquad\qquad\quad ag \in \mathrm{ran}(ti.queue(a)) \wedge & [13]\\
\qquad\qquad\quad r \in ag.allowedroles \wedge & [14]\\
\qquad\qquad\quad r = second(target))))) & [15]
\end{array}$$

The *SelectNewTargets* schema (see Schema 44) allows an agent (line 1) to specify new targets (line 2) at a given transition instance (line 3). If those targets are consistent, as defined above by the consistentORtargets (line 6) and consistentANDtargets (line 7) predicates, for the current transition instance, then the agent's targets are updated (line 8).

An agent waiting at a transition instance can change its targets, and may thus possibly need to remove some of its current targets. The *RemoveOldTargets* schema (see Schema 45) allows an agent (line 1) to remove old targets (line 2) at a given transition instance (line 3). Such removal of targets (line 5) is allowed, provided that the remaining targets are still consistent (lines 6, 7), depending on the type of transition hosting the agent. For this purpose, we rely on the

SelectNewTargets

$ag? : AgentProcess$ [1]
$newtargets? : \mathbb{P}(STId \times RoleConst)$ [2]
$\Delta TransitionInstance$ [3]
$potentialtargets : \mathbb{P}(STId \times RoleConst)$ [4]

$potentialtargets = processwantsnext(ag?) \cup newtargets?$ [5]
$(consistentORtargets(\theta TransitionInstance, ag?, \{tar : (SceneInstance \times RoleConst) \mid (first(tar)).sid \in (\text{dom}\, potentialtargets) \bullet tar\})) \vee$ [6]
$(consistentANDtargets(\theta TransitionInstance, ag?, \{tar : (SceneInstance \times RoleConst) \mid (first(tar)).sid \in (\text{dom}\, potentialtargets)\}))$ [7]
$\Rightarrow processwantsnext' = processwantsnext \oplus \{(ag? \mapsto potentialtargets)\}$ [8]

Schema 44. An agent defines a new set of targets in terms of which roles it seeks to play in which scene instances.

RemoveOldTargets

$ag? : AgentProcess$ [1]
$newtargets? : \mathbb{P}(STId \times RoleConst)$ [2]
$\Delta TransitionInstance$ [3]
$potentialtargets : \mathbb{P}(STId \times RoleConst)$ [4]

$potentialtargets = processwantsnext(ag?) \setminus newtargets?$ [5]
$(consistentORtargets(\theta TransitionInstance, ag?, \{tar : (SceneInstance \times RoleConst) \mid (first(tar)).sid \in (\text{dom}\, potentialtargets)\})) \vee$ [6]
$(consistentANDtargets(\theta TransitionInstance, ag?, \{tar : (SceneInstance \times RoleConst) \mid (first(tar)).sid \in (\text{dom}\, potentialtargets)\}))$ [7]
$\Rightarrow processwantsnext' = processwantsnext \oplus \{(ag? \mapsto potentialtargets)\}$ [8]

Schema 45. An agent removes a set of targets.

consistentORtargets and *consistentANDtargets* predicates defined above. Thus, if the remaining targets are consistent, the agent's targets are updated accordingly (line 8).

7.6. Closing down activities

Finally, we must consider how scene instances are closed down and the final (terminal) state of an electronic institution reached. According to the *CloseSceneInstance* operation (see Schema 46), when a scene instance reaches a closing conversation place (line 4), the instance can be closed down provided that all agent processes in the scene play roles that are allowed to leave the closing state (line 5). If so, all agent processes are removed from the scene instance (line 6), and this in turn changes its state to closed (line 7).

CloseSceneInstance

$\Delta SceneInstance$ [1]
$closing? : \mathbb{P}\, AgentProcess$ [2]

$sceneprocesses = closing?$ [3]
$place \in scene.closing$ [4]
$\forall a : closing? \bullet place \in scene.leaving(a.role)$ [5]
$sceneprocesses' = \{\}$ [6]
$status' = Closed$ [7]

Schema 46. All remaining agents leave a scene instance together and close it.

Once a scene instance is closed, the *RemoveClosedInstances* operation removes it from the list of scene instances kept in the state of the electronic institution (line 2). (See Schema 47.)

RemoveClosedInstances

$\Delta ElectronicInstitutionState$ [1]

$sceneinstances' = sceneinstances \setminus \{si : sceneinstances \mid si.status = Closed \wedge si.sceneprocesses = \{\}\}$ [2]

Schema 47. The closed scene is removed form the system by the electronic institution infrastructure.

Table 9
Electronic institution operations.

Operation	Called by	Instance
StartElectronicInstitution	Infrastructure	electronic institution
CreateSceneInstance	Infrastructure	scene institution
RequestAccess	Agent	electronic institution
JoinInstitution	Agent	electronic institution, scene
Speak	Agent	scene
Timeout	Infrastructure	scene
CloseSceneInstance	Infrastructure	scene
SelectNewTargets	Agent	transition
RemoveOldTargets	Agent	transition
EnableAgentsToLeaveOrTransition	Infrastructure	transition
EnableAgentsToLeaveAndTransition	Infrastructure	transition
MovingFromSceneInstanceToTransitionInstance	Infrastructure	scene, transition
MoveAgentFromTransitionToScene	Infrastructure	scene, transition
LeaveInstitution	Agent	electronic institution, scene
RemoveClosedInstances	Infrastructure	electronic institution

The final state of an electronic institution thus occurs when all scene instances have been closed down so that only the entry and exit instances remain. In other words, the final state is semantically equivalent to the initial state.

7.7. Summary

To summarise, Table 9 lists the operations that an electronic institution infrastructure is required to implement. For each operation, the *Called by* column identifies whether the operation is triggered by an agent's action, or instead whether it is internally triggered by the infrastructure. Moreover, the *Instance* column identifies the main instances of data structures that each operation updates. Observe that the operations leave it to each agent to decide:

- which roles to play in an institution;
- when to join and leave an institution;
- what to say, when and to whom; and
- where and when to move and what role to play there.

The operations specified here thus enable agent actions in an institutional context. In other words, the operations in Table 9 turn individual agent decisions into social actions with social effects. In summary, we have provided the key necessary operations for an electronic institution, but clearly different instantiations will require further and more refined operations. However, the specification gives the tools to add to the foundational system defined here for other purposes.

8. Computational architecture and tools

To this point, we have been concerned with introducing the key constructs that compose the formal specification of electronic institutions, and detailing the operations required to operate on such constructs. However, we have not yet considered how to run an electronic institution through a computational infrastructure capable of performing the operations specified in Section 7 on the structures specified in Sections 4 and 6. In this section, we address this issue, and also consider software tools for electronic institution engineers aimed at easing the intricate tasks of designing and running electronic institutions. We begin by addressing the issue of architecture, showing how a computational electronic institution is realised in one particular model, before reviewing several tools that have been developed to assist in the process of institution specification and implementation.

8.1. An agent-based architecture for electronic institutions

As we have seen, an electronic institution specification establishes the basis for agent interactions (scenes) and organises interactions into interconnected activities (scene networks). A computational infrastructure running an electronic institution provides a persistent computational environment in which the participating agents are required to abide by the rules of the specification. The computational model intended for such computational infrastructure is simple: agent actions (utterances and moves) cause an electronic institution state (as specified in Section 6) to evolve by performing the operations described in Section 7. For example, the agent-based architecture described in [43], and illustrated in Fig. 13, is one such infrastructure that realises this computational model. It is thus responsible for guaranteeing the correct evolution of each scene execution (by means of valid utterances), and guaranteeing that agent movements between scene executions comply with the specification.

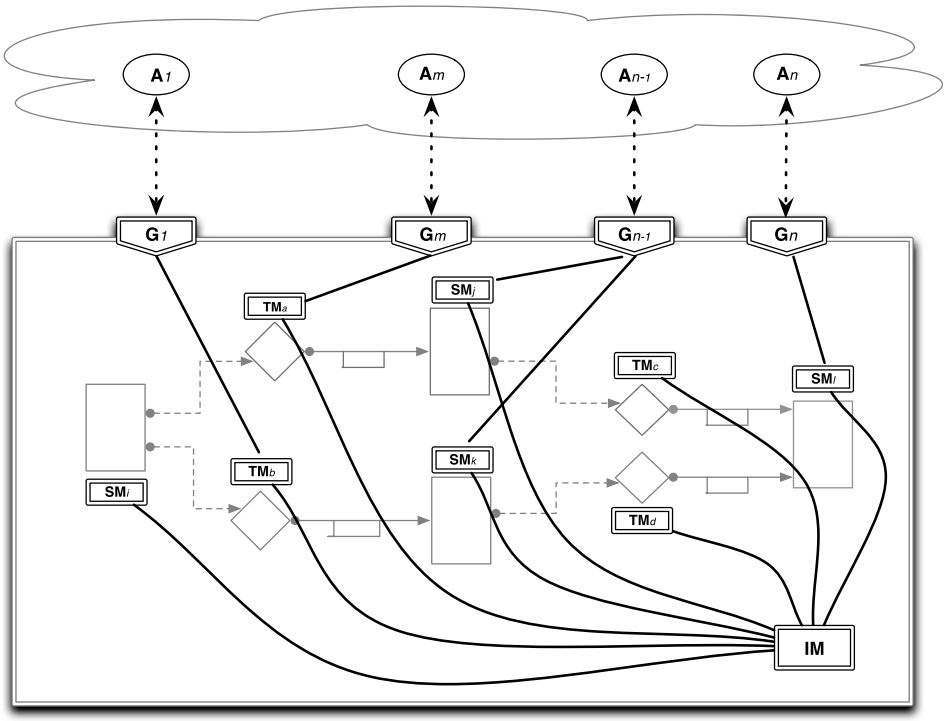

Fig. 13. An architecture for electronic institutions. Participating agents (A), communicate with (infrastructure) governor agents (G), which in turn coordinate with other infrastructure manager agents for each scene (SM) and each transition (TM) and with the institution manager agent (IM).

In more detail, the architecture in Fig. 13 is composed of several infrastructure agents. In what follows, we identify the agents involved in each aspect of the infrastructure, along with the operations in Table 9 that each agent contributes to implementing.

Institution management Each institution has one *institution manager* agent (IM), which activates the *StartElectronicInstitution* operation and terminates the institution. It also controls the entry (*RequestAccess*, *JoinInstitution*) and exit (*LeaveInstitution*) of agents, together with the creation of new scene instances and the closing of scene instances (*CloseSceneInstance*, *RemoveClosedInstances*). Finally, it keeps information about all participating agents and all scene and transition instances. In essence, it keeps track of the electronic institution state.

Transition management Each transition has a *transition manager* (TM) that controls the transit of agent processes between scene instances by checking that requested moves are allowed (*EnableAgentsToLeaveOrTransition, EnableAgentsToLeaveAndTransition*) and, if so, allowing agent processes to move (*MovingFromSceneInstanceToTransitionInstance, MoveAgentFromTransitionToScene*).

Scene management Each scene has an associated infrastructure agent, the *scene manager* (SM), which is in charge of: starting and closing the scene instance (in coordination with the institution manager); keeping track of agent processes that enter and leave the scene instance; updating the state of the scene instance by processing utterances (*Speak*) and time-outs (*Timeout*); and coordinating with transition managers to let agent processes in or out a scene instance (*MovingFromSceneInstanceToTransitionInstance, MoveAgentFromTransitionToScene*).

Mediation All interactions between a participating agent and the institution are mediated by an infrastructure agent, known as the *governor* (indicated as G in Fig. 13). Thus, participating agents cannot interact directly with one another; they have their interactions mediated by their *governors*, which mediate *all* interactions within the institution. There is one governor per participating agent. A governor offers a number of services to its agent so that it can enter and exit an institution, and its agent processes can perform utterances and move between scene instances. Note that in order to provide each of the above-mentioned services, a governor must coordinate with scene managers, transition managers, and the institution manager. In this realisation of the electronic institution framework, therefore, governors are involved in the implementation of most of the operations in Table 9.

In this architecture, the execution of an electronic institution begins with the creation of an *institution manager*. Once running, the institution manager activates the initial and final scenes by launching a scene manager for each one. Thereafter, new scene instances, as well as scene and transition managers, become active as required by agent interactions. In order to participate, external agents must ask the institution manager to join the institution. When an agent is authorised to join

the institution, it is connected to a governor and admitted into the initial scene. From there on, agents can move around the different scene instances or trigger new instances according to the specification and the current state of the electronic institution.

8.2. Software tools

In this section, we review several software tools available to ease development and implementation of electronic institutions. However, we do not provide excessive detail of these tools, since our aim is merely to illustrate the range of possibilities. A first, intricate step in the development of an electronic institution is to design it. This amounts to first specifying the electronic institution languages, and second specifying the institutional constructs. The result of this process is an electronic institution as a data structure, along the lines of that of Section 4. This task is supported by ISLANDER [41], a graphical software tool, which provides particular implementations of update, constraint, and attribute languages, and also offers graphical representations of institutional constructs to aid a designer in composing specifications of electronic institutions. Furthermore, ISLANDER supports the static verification of an electronic institution design by checking for *language integrity*, *structural integrity*, and *liveness* properties.

Now, any electronic institution description created with ISLANDER is a formal specification (in XML) that is a refinement of the general specification (in Z) contained in this paper. In this respect, all constructs formally described in this paper are supported by the ISLANDER tool. While the specifications generated by ISLANDER in XML are not automatically derived from the Z specification, it is possible to provide translations of the XML in Z so that they may be checked for consistency with the model specification in this paper. There are well-documented methodologies for deriving translations to and from Z specifications, and such automatic translation is under consideration for future releases.

Once an electronic institution specification is ready, it can be computationally enacted by means of a computational infrastructure. AMELI [43] provides such infrastructure as a particular implementation of the architecture outlined in Section 8.1 above. The infrastructure enacts an institution specified with ISLANDER [43] that is open to the participation of agents. Thus, AMELI activates infrastructure agents (institution manager, scene managers, transition managers, and governors) as needed, and controls the activation of scene and transition instances, the access of agents, and their actions (be they utterances or moves). In general, the coordination of infrastructure agents in AMELI guarantees the correct evolution of scenes and the correct movement of agents between scene instances.

Both ISLANDER and AMELI are part of the *electronic institutions development environment* (EIDE) [44], a collection of tools aimed at supporting the development of electronic institutions. Besides these core tools, EIDE offers others for the simulation of electronic institutions (SIMDEI) and for the development of participating agents. Moreover, recent developments have focused on providing support for humans to participate. First, HIHEREI [20] offers a web-based human interface to electronic institutions. Second, the work in [106] supports the interaction of humans in electronic institutions through virtual worlds. While much can be written about these aspects, we do not comment further in this paper, since they are not the focus of the current work.

9. Evaluation

As we have now described both the model and our tools for designing and implementing systems we are in a position to reflect upon and evaluate our work. As well as describing the precise nature of the contribution of our formal model we will also describe the limitations of the work in terms of how it relates to our tools, and outline our response to these limitations. First, we describe the precise contribution in terms of the formal model described in this paper.

1. The model provides, for the first time, a unified account of both the data structures and operation of electronic institutions.
2. All the concepts described in the model have a computational semantics. In addition, the operational semantics are provided in the same formal language.
3. The model provides a recipe for the design of electronic institutions as demonstrated in Section 5 with the illustrated example of designing an institution based on a human institution. (Of course, electronic institutions need not be inspired by human organisations and could be designed from entirely artificial metaphors.)
4. The model has been specified in the Z language, for which there are well-documented refinement mechanisms for producing concrete software. Our model therefore provides the blueprint not only for design (as in the previous point) but also for implementing concrete electronic institutions.
5. Because of the ubiquity of Z, there are many Z-related tools, including type and syntax checkers that guarantee internal consistency (which is not possible in general for formal descriptions of systems). In particular, with a specification of this size and complexity it would be impossible to produce a formal description that would not be littered with errors without such tools.
6. The model provides a structured framework for the development of further refinements. (For example, we can use these schemas to develop a more specific model with further constraints, to design additional data structures to model families or subtypes of institutions, or to implement a specific model of norms, for example.) This is in part a direct result of the property of allowing schema inclusion in Z, which allows more detailed and refined concepts to be introduced.

Table 10
Excerpts form the Dutch auction scene.

sname:	**Dutch Auction**
sceneroles:	auctioneer, buyer
places:	start auction, bidding, call price, winner declaration, end auction
moves:	(start auction, bidding), (bidding, call price), (call price, bidding), (call price, end auction) (bidding, bidding), (bidding, winner declaration), (winner declaration, end auction)
link:	(start auction, bidding) → Line_1, (bidding, call price) → Pause_1, (call price, bidding) → Line_ 2 (call price, end auction) → Line_3, (bidding, bidding) → Line_4, (bidding, winner declaration) → Line_ 5 (winner declaration, end auction) → Line_6
access:	auctioneer → start auction, buyer → start auction
leaving:	auctioneer → end auction, buyer → end auction

Table 11
Excerpt from an erroneous scene definition.

access:	auctioneer → start auction, buyer → start auction
leaving:	buyer → end auction

All of these aspects of our model enable future systematic, incremental and compositional development to take place. Even though there is much work on electronic institutions that has been published to date, it is not clear how these new ideas can always be brought together because they often have differing assumptions about the underlying semantics, for example. New work relating to electronic institutions as described in this model can now be integrated into the body of work that aims to describe how open systems can operate in theory and practise. Finally, though we provide no evaluation of it in this paper, we claim that our model provides a *reference ontology* for open multi-agent systems in general.

Turning to the limitations of the model, we note that the challenge of both automatically maintaining the integrity of attribute values and controlling the simultaneity and concurrency of agents playing multiple roles is outside the scope of this paper. Furthermore, there are two specific limitations to the formal model as it stands in relation to the tools we have described in the previous section, and that should be clarified here. First, we have no automatic technique for translating between XML code and Z representations which would make the kind of manual checking described below a more straightforward process. However, this should be relatively easy to overcome in future work. Second, and more problematically, there are no automatic techniques for proving that any system designed with our toolset is correct with respect to the properties of the model. While the ISLANDER tool checks some of the properties defined in this specification, as yet it is neither complete nor easily extensible, since the checking of every property must be hardwired in the tool. However, because the specification provides a very clear and explicit definition of what constitutes a well-defined system, it does make manual testing possible, as we demonstrate below.

Consider the worked example of Section 5 and the designed scene described in Table 3 on p. 60, from which we extract some excerpts below.

Now consider Schema 13 of p. 58 and within this schema the following property from line 21.

$$\forall r : RoleConst;\quad c_1 : ConvPlace \mid c_1 \in (access\ r) \bullet \left(\exists c_2 \in (leaving\ r) \bullet (c_1, c_2) \in moves\star\right)$$

This predicate states that if an agent incarnating a particular role can enter a scene, then there must be some future conversation place, which is reachable, where the agent can leave the scene. Let us evaluate this constraint with respect to the design provided in Table 10, instantiating the variable r first to the auctioneer and then to the buyer.

$$access(auctioneer) = start\_auction$$

$$access(buyer) = start\ auction$$

$$moves\star = \left\{(start\ auction, call\ price), (bidding, end\ auction), (start\ auction, end\ auction), \ldots\right\}$$

We must prove that, for the auctioneer, and for each of the accessible places that are available to it, that there is at least one leaving place that is reachable from it. The only accessible place for auctioneers is *start auction* and it is easy to see that there is an ordered pair $(start\ auction, end\ auction) \in move\star$ such that *end auction* is a leaving place for *auctioneer*.

If, by chance, we had erroneously designed a scene as described in Table 11, it should be clear that this property could not be satisfied and would force us to revise our design.

10. Related work

The electronic institution model presented in this paper builds upon the ideas originally proposed in [40,78,95] and developed subsequently. In this section we discuss how this new model relates to other work.

Modelling software as an organisation of components with clear interactions was pioneered by Gasser [58]. However, recent developments in process algebras (for example, [22,77,92]) and business processes specification languages like BPEL

[17] are not particularly well suited for this domain since, as we discuss in this paper, the most important aspect of any interaction is to represent joint activities of agents. The key difference is that these technologies are aimed at specifying hard-wired processes around individual agents and their individual interactions, and are not suited to defining group activities that allow agents to autonomously agree on how to interact and when.

More similar to our approach is the work that falls into the broad area of organisational views of multi-agent systems. In this line of work, as opposed to a classical agent-centric view of multi-agent systems, aspects such as roles and social structure (groups, communities), organisational goals and, to some extent, patterns of interaction among agents, become prominent [45,64,89]. Even closer to the notion of electronic institutions is the approach based on *activity theory* [81] that takes the notions of agent and of artefact as primitive, first class, elements in a multi-agent system. In particular, MOISE+ [62,63] allows the specification of organisations in which agents, playing particular roles, are involved in activities within a virtual environment populated with artefacts [81]. The model considers agent goals, role-based constraints and the distinction between regimented and non-regimented norms. Implementations are built with an application-independent programming framework (ORA4MAS) that is analogous to the electronic institutions framework. Through these tools it is possible to express the functionality of artefacts and their interface with agents, and use norms to define compatibility of tasks with roles. However, the ORA4MAS-MOISE+ framework does not create an infrastructure that centrally mediates agent communications, like our architecture does. Instead, it allows agents to interact directly with each other and with the environment through artefacts, but allows the implementation of only what amounts to a single scene in our model.

Some authors refer to multi-agent systems in which agent interactions are regulated as normative multi-agent systems [13–15]. The focus of this approach is not only how an agent is affected by norms [28,74] but also how norms are expressed (for example, [59,69,75]), enforced [60] or modified [18], and how they may be programmed and implemented [33]. As suggested below, when the normative interpretation of our proposal is made explicit, our work is consistent with this normative multi-agent system approach.

There are two salient issues when assuming a normative approach. First, the notions of compliance and enforcement become central. Second, it provides the possibility of having a convenient declarative specification of institutional conventions.

With respect to enforcement, it is often considered that there is a need to have regulated multi-agent systems in which agents are subject to norms that may be violated. The claim is backed by three arguments: (i) conventional legal frameworks work in this way and regulated multi-agent systems are likely to be called on to fulfil the role of complex legal systems; (ii) in several contexts, global goals may be in conflict with norms and an individual may contribute more to a global goal by violating a norm; (iii) agent autonomy is pointless if an agent may not choose to disobey a norm. This interest in potential non-compliance is reflected in the way norms are represented and used. When norms may be violated, implementations usually require that norms have a contrary-to-duty component that is triggered when a violation occurs. Thus if full observability of violations is assumed and enforcement is centralised, the operational semantics is straightforward: violations are acceptable but when they happen, they always trigger contrary-to-duty clauses and all the consequences of those clauses hold [32,62]. When there is only partial observability of misconduct in the system, then governance needs to be supported by explicit enforcement mechanisms as part of the system – e.g., agents with law enforcement roles – and even to support due processes for blame assignment and reparation. Our proposed model is not committed to any particular enforcement process, nor does it prevent them. We elaborate this notion below but state now that one way of adding flexible enforcement mechanisms to specific electronic institutions is discussed in [30].[14]

Declarative normative specification of regulations have the advantage – over procedural specifications – of expressing conventions in a more abstract fashion that, in principle, is suited to formal and automated reasoning and closer to the way in which conventional legal regulations are formulated, while still being machine readable.

The abstract model we propose here may be reified as a model for normative multi-agent systems. In fact, electronic institutions may be understood as a way of implementing a regulated environment where certain actions are permitted, some are forbidden and some others are obligatory. Furthermore, this deontic character is enforced by the electronic institution in such a way that actions may only happen if certain conditions are met while actions that do take place have the intended institutional effect. Hence, not surprisingly, many standard normative aspects are implicit in the model specification given in this paper. The domain language and related aspects may be understood as constitutive norms, while scene types and transitions correspond to procedural norms in the form of implicit permissions (available paths) and obligations (either unreachability or determinacy of places), and constraints and updates may also be seen to carry the same intent as the usual functional norms. However, one may prefer to have a more explicit use of norms in order to exploit their formal features, specifically the inferential ones, and therefore facilitate agents and designers to reason about the institutional conventions. The attribute, constraint and update languages defined in Section 3 may thus be included in a unique normative language that enables such explicit use of norms.

In order to have a fully declarative specification of an electronic institution, we would need to adapt the set of operations in Section 7.7 to the use of norms, and – in order to handle commitments – to add operations that account for the addition and removal of obligations, permissions and prohibitions. In essence, since any possible action within the institution is an utterance, these operations need to support the firing of norms provoked by agent utterances and how that firing may affect

[14] The approach to enforcement taken by Hubner et al. in [62] (p. 395) is similar, since they allow non-regimented norms that are detected by artefacts and evaluated and enforced by internal agents.

the institutional state. It should be noted that because explicit norms – expressed in the normative language – may refer to prohibitions, obligations and permissions, the state of the institution will need to include the normative positions of every agent. Note also that the update language described in Section 3 should now be able to refer to and update those normative positions.

It should be evident that when electronic institutions are reified as normative multi-agent systems, one may accomplish the same that is accomplished with the abstract model. Nevertheless, aspects like governance become more evident in this new guise and, without removing anything from the abstract model, one may extend it by adding more operations and constructs to exploit these aspects further. Hence, for particular implementations, one may want to include specific means for handling propagation of normative positions and, in particular, to deal with normative conflicts, norm violations and, depending on the normative language chosen, all the conditions that need to be observed for a given norm to be fired, and all the actions that need to be accounted for with the update language when norms are fired.

Some extensions to the model we propose here that make specific reference to normative aspects are presented in a systematic fashion by Garcia-Camino [55]. His proposal in [54] also illustrates how norms may be used to provide a declarative logic-based specification of institutional conventions that may be tested on-line against agent actions using formal tools. The possibility suggested there is to make all parametric languages part of a single rule-based normative language, represent scenes as a theory in that language and use an inference engine to test consistency of intended institutional facts before updating the theory with these facts. A more elaborate normative language and inference mechanism to expresses complex deontic formulas involving time and other constraints were presented in [56] and may be used to capture functional norms as well as procedural and constitutive ones that may be non-regimented. Another facet that has been explored is how scene protocols may be expressed as modal formulas thus enabling agents to reason about compliance, consequently lowering the entry programming cost for agent developers [38]. Finally, there is a proposal [53] to extend the computational architecture described in Section 8 to include a new type of infrastructure agent that manages norm activation and consistency in each scene at run time, plus an institutional normative manager that coordinates with all normative scene managers for the propagation of normative positions and the resolution of normative conflicts. This proposal would facilitate a distributed execution of institutions and the use of the coloured petri nets formalism to prove some formal properties of any given electronic institution.

A different declarative approach for agent coordination is based on the use of standardised speech acts. In this approach, interactions may be organised with commitment-based protocols that involve directed obligations among agents. The ground assumption is that instead of defining protocols with (flat) low-level messages (such as send or receive), one may express the protocol conventions through a dialogue game involving illocutionary particles (promise, request or declare) that have fixed *social* meaning and pragmatics, that are shared by the individuals involved in those games [7,27,29,48,103,104].[15]

Thus, the meaning of communications are explicit, common to all participants and therefore constitute signalling devises on which to build expectations and are amenable to compliance checking. A key feature is that commitment-based protocols semantics and pragmatics are public and do not depend on the mental dispositions of those agents. Hence these protocols only mediate public communication exchanges, not the mental decision-making processes that participants may need to make in order to engage properly in such dialogue games. The commitment-based protocols approach has four guiding concerns (i) flexibility in the specification of the protocol and its evolution; (ii) standardisation of means and meanings (ii) due awareness of context-dependence of meaning and practises; (iii) compliance effectiveness (to ascertain when an individual is not complying and to react appropriately, without over constraining agent autonomy); (iv) amenability to analysis and testing of properties like liveness, robustness, reusability, correctness and effectiveness. The ideas behind the commitment-based protocols approach are made operational as *commitment machines* [112,23] and in the works by Fornara and Colombetti [49], for example. The gist of the matter in these works is (i) that protocols are presented in a declarative way in an illocutionary language, (ii) that this language lends itself to formal analysis, and (iii) that a complex protocol may be assembled from simpler ones. While in a way analogous to our own proposal, commitment machines define an interaction protocol in terms of actions that change the value of state variables of the institution, in connection machines agent actions may explicitly also create, modify or discharge commitments that in fact include their own protocol. The crux is that in commitment-machines agents may reason about standard commitment-making patterns (an offer, or a promise), which is something that our proposal does not provide. Thus, commitment machines allow for a more compact expression of protocols, protocol composition and formal testing of protocol properties [24,111]. It is not clear to us at this point what would be the trade-offs of using commitment-based protocols to express the contents of our scenes and scene networks. There seem to be obvious advantages in conciseness, modularity, and formal testing and, more significant still, that commitment-based protocols may also provide an elegant way for the implementation of institutional dynamics (like changing a protocol, adding scenes); something that our current framework is not designed to accomplish. On the other hand, however, the model we present here may be expanded to accommodate the expression and handling of commitments, and thus maybe allow for a natural use of commitment-based protocols. Such expansion would be similar to the expansion for the full declarative implementation of normative notions mentioned above: first it would be necessary to include for each agent the set of its commitments, as a state variable that is updated like any other state variable; second, the set of oper-

[15] The presence of illocutionary particles in our illocution templates (Section 4) betrays a yet unaccomplished development along precisely these lines that was already suggested in [78].

ations in Section 7 needs to be expanded with operations that account for the life-cycle of commitments, and the update language described in Section 3 correspondingly adapted; finally, an inference mechanism to reason about the pragmatics of illocutionary formulae should be added to the model along the lines discussed in [54].

A compromise between a centralised commitment-based protocol approach and a low-level (mentalistic) communication language is proposed in the *Light Coordination Calculus* (LCC) of Robertson, McGinnis et al. [93]. Indeed, LCC was directly inspired by the work on electronic institutions. LCC addresses autonomous interactions in a peer-to-peer context with solutions that share many features with our work. In the LCC framework, a core set of primitive operations (message exchange actions, control flow and conditionals) are used by autonomous agents to express and agree on role-based interaction protocols, and coordinate their actions according to those protocols. Agreements and coordination is achieved through a simple dialogue game where agents exchange utterances to create a conversation, whose state is passed from one agent to the next, together with the label of the dialogical move and a Prolog-like structure that contains the protocol or the remaining part of it. (This is similar to the notion of continuation in functional programming.) The contributions of LCC are twofold. First, there is the very natural possibility of using these dialogue games for the dynamic establishment of agreements on ontologies, protocols and commitments, thus overcoming the limitation of a pre-defined class of admissible conversation states in an electronic institution. Second, there is the inherently distributed nature of peer-to-peer interactions that circumvents the implementation of a centralised institutional control.

LCC was presented by McGinnis and Robertson in [76] as an alternative to electronic institutions, claiming that the main elements of our model (dialogical framework, utterances, scenes, scene network, and state of a scene) may in principle be reified in their formalism. In fact, both that claim and the converse are true modulo the computational architecture chosen and, in particular, in relation to (i) the way one chooses to implement the maintenance of the institutional state – especially for multiple co-dependent conversations – and (ii) the way the dialogue games may be started and terminated. A full discussion of the correspondence is beyond the scope of this paper but, roughly speaking, one may implement an LCC-like peer-to-peer electronic institution with the computational architecture and tools presented in Section 8 as suggested in [42]. In an implementation of a peer-to-peer electronic institution, each agent would have a plug-in attached that constitutes the institutional support for that agent's interactions.[16] A full implementation of a peer-to-peer electronic institution involves three phases, each of which is in fact an institution: (i) a bootstrapping match-making phase where agents convene to define a joint interaction, (ii) an agreement phase where participating agents negotiate the actual institution (conventions) that will govern the interactions and (iii) the enactment of the agreed-upon institution. The main limitation that our current framework needs to overcome is the availability of the "metaoperations" needed for each of the three phases: that is, for the match-making process (search and invitation of partners, acceptance to become involved), for the negotiation phase (selection and assembly of institutional components by specifying new ones or choosing from repositories) and those that enable agents with the capability of enacting the agreed-upon institution (to activate and run processes).[17]

While the work we have referred to so far in this section has particular features in common with the work in this paper, the following four lines of work are guided by institutional and organisational intuitions that are closer to our own.

Colombetti and Formara have proposed a metamodel to build open multi-agent systems. They define an *artificial institution* [51] as an object formed by an ontology that characterises the social context of interaction, a set of empowerments, a set of linguistic conventions and a set of norms that constrain agent interactions. As in our work, institutional actions are dialogical but, as with Singh, they also define directed obligations subject to the above mentioned linguistic conventions. Here, atomic actions are organised into conversations whose structure and semantics are grounded on an application-independent semantics. Entities have natural and institutional attributes, which are changed only through institutional actions. There is also an associated model-checker and an implementation platform [50] currently under development.

Dignum et al. have made proposals (for example [109]) that share many elements with some of the ideas presented here. Stemming from work on the OperA [35] and HARMONIA [108] frameworks, they have proposed the OMNI model [36, 109] to build agent-based organisations and, more recently, the OperettA suite to support implementation. This model takes institutional and individual agent goals as the key features that articulate the domain ontology, social structures, plus regulations and governance. The model contains three types of interrelated institutional components, contextual, organisational and normative, that are to be expressed in three progressive levels of abstraction – from description to implementation. In particular, agent interactions are organised in agent scripts, called scenes, which are connected by transitions that establish temporal sequencing. These scenes are built around a collective goal and, in each scene, the repertoire of admissible agent actions is regulated by a set of rules. The framework includes formal tools to assess the normative properties and the proficiency of the organisational structures.

Lopez-Cardoso and Oliveira et al. [71–73] have also looked into the idea of electronic institutions as a coordination mechanism, focusing on the problem of e-contracting in virtual organisations. With this background they identify contracts with a set of norms and conceive the electronic institution as a collection of tools to support contract making and follow-up. In particular, they are concerned with the creation of normative relationships among agents at run-time, and their enactment.

[16] Thus, the plug-in performs the functions of those infrastructure agents of the computational architecture described in Section 8. In fact, depending on the specification of the institution, at some point the plug-in may take the functionalities of a governor, of a scene manager or of an institutional manager.

[17] Such metaoperations (to which we alluded above when discussing commitment-based protocols) would also be required for institutions that have the possibility of changing themselves or spawning new institutions. An obvious consequence of these metaoperations is the unboundedness of the class of potential commitments of an electronic institution, because each new scene that is added to an existing institution brings its own class of commitments.

Thus they propose a formal representation – using temporal logic – of directed contractual obligations, a hierarchical framework for norm representation and consistency management, plus an environment to support contract representation and to maintain the normative state of obligations. They have also developed a prototype implementation of these ideas.

In turn, Padget, deVos and Cliffe have developed answer-set programming tools to study electronic institutions [26] and have explored interactions among multiple institutions [25].

Finally, we should mention that the intuitions behind the electronic institution model (and, to certain extent, by those four groups we have just mentioned) are quite close to the notion of institution studied by economists [1,80] and also that of organisational theory or political science [84,88] where an institution is understood as a set of restrictions imposed on agents' behaviour in order to coordinate their interaction. These restrictions are capable of harnessing behaviour because they create a governed context in which agent actions have a particular character. Thus, from a conceptual perspective, the conventional notion of institution entails an institutional reality that is different from the ordinary reality of the world, as Searle discusses in [98]. Our work shares these fundamental intuitions but we are concerned with a more specialised notion, one that aims to be implemented and applied to articulate interaction in open systems in general and multi-agent systems in particular.

Along lines similar to Searle's concerns – but with an interest in implementation and using logical tools – Jones and Sergot formalise the notion of institutional power through an analogous *counts as* relationship, the notion of entitlement and the use of normative positions [67]. On top of these developments, Artikis [8] formalises and implements, with the use of the event calculus, the notion of a norm-governed open multi-agent system – covered by our electronic institution model – and in particular discusses its dynamics [6], an issue that is also explored by Esteva et al. in the context of electronic institutions [21].

11. Conclusions

11.1. On applications

The components of the abstract model of electronic institutions have been introduced and specified. All this provides an institutional environment that is beyond the interpretation of agents, enables actual interactions, enforces conventions and supports awareness. The specification provides a foundation for further developments of open multi-agent systems. To illustrate some of these aspects, we have provided brief examples of actual instantiations of languages and computational architecture, which aim to show that the model is feasible. Importantly, however, we also claim the model is useful, in support of which we now discuss the types of system that may be developed, based on this model of electronic institutions.

There is a key conceptual distinction between the electronic institution model and other agent-based models and conventional software engineering paradigms. In the electronic institution model, interactions between processes are conceived as group meetings (scenes) in which all intervening agents share a common state that changes only when valid actions take place. Moreover, agents may enter and leave these group meetings only through gates (transitions) whose specifications are role-specific. In this fashion, the scene network determines the evolution of a shared state of interactions and the flow of agents playing particular roles, rather than just the flow of information as in conventional workflows. From a workflow perspective, this facilitates the implementation of flexible coordination conventions. From an application development perspective, the electronic institution model and its implementation have other distinctive features that make them eminently suitable for a large class of applications. We summarise the most salient features below.

1. Electronic institutions are concerned with the design and implementation of the environment and not with the internals of participating agents. This fosters:
   (a) a sharper awareness of coordination and governance features;
   (b) a separation of concerns between the design of social conventions and the design of the capabilities of individuals; and
   (c) an encapsulation of best practises and procedural conventions through scenes and scene networks (that reflect best practises).
2. Electronic institutions implement open systems in which external agents:
   (a) may enter and leave the institution at any labelled entry or exit state of a scene;
   (b) may be independent (built or owned independently of the institution);
   (c) are self-motivated (have their own goals); and
   (d) have their own internal model not controlled by the institution.
3. The electronic institution model is neutral with respect to the agent architecture, thus facilitating the participation of both human and computational agents – in debugging or enactment – if adequate interfaces are provided (e.g., [19,20]).
4. Scenes and the scene networks foster modular design and reusability. In particular, scenes and role changes between scenes may be individually edited. Consequently, scene variants are simple to implement and available scenes may be pruned or spliced into a network.
5. The electronic institution model allows the coexistence of several active scenes at any time and, more specifically, the possibility for any agent to be active, simultaneously, in more than one. Each scene is subject to its particular conventions but the electronic institution model guarantees that commitments are duly propagated among scenes.

At the risk of oversimplifying, we can identify four kinds of applications for which this electronic institution model is distinctively applicable because of the type of features enumerated above. For each kind of application, we first suggest the features that are most prominent for it, listed in order of significance, and second, we give examples of such systems, referring to some of those that we have actually implemented.

- Interactive systems involving stable virtual spaces, especially those for which independence and autonomy of agents is essential but wrong-doing may have significant cost and, therefore, a trustworthy third party and strong convention enforcement are paramount (features 1(b), 1(a), 2, 5, 3). For example, auctioning [78], voting polls, electronic markets [31], automated negotiation [91] and the Grid [9].
- Regulated systems whose conventions may be fixed but where the consequences of those conventions are difficult to assess without some systematic experimentation (features 1, 2, 3). For example, mechanism design [4,3], and public policy management [57].
- Coordination systems where, in spite of the need for a sharp specification of interactions, the precise flow of activity may be difficult to know beforehand, or multiple variants of some activities may be necessary at some point or another during the system life-span (features 4, 3, 1(a), 1(c)). For instance, supply networks and public procurement, vertical-market corporate information systems [94], social opinion gathering [82], and web-service choreographing.
- Regulated systems whose debugging or actual use involves a mix of human and software agents (features 3, 4, 5, 1(c), 1(b), 1(a)) [19]; like on-line multiplayer games [2], participatory simulation [16], and on-line dispute resolution [79].

11.2. Contributions

In conclusion, this paper proposes an abstract model of electronic institutions. It contains a formal specification of a model that provides, on the one hand, a crisp exposition of the concepts underlying our intuitions about electronic institutions, and on the other hand, for those interested in development of open systems, a blueprint for the implementation of the model.

The model shifts the concerns of agent architectures, from a socially stateless view towards one in which meaning refers to a collective context. It also provides a reference ontology of institutional notions that may be used, refined and extended to characterise or contextualise a wide range of systems. Moreover, the model includes the core institutional governance means to articulate autonomous interactions.

The model is sufficient to describe those open systems: (i) that involve heterogeneous, self motivated and autonomous agents, (ii) that need to interact with other agents in order to accomplish their goals, (iii) whose interactions involve group activities within a shared context and (iv) whose participants are willing and able to conform to semantic, procedural and governance regulations that are enforced by the system. Our model does not apply, for instance, to systems whose components are not able nor willing to comply with any governance regulations.

The model is not restrictive. It does not commit to particular formalisms or implementations, although it does specify all those elements that are necessary to build the type of open systems described above. The model may be extended – with additional operations and data structures – to address further functionalities needed in particular types of that class of open systems. In particular the model does not prevent, for example, systems that involve the possibility of on-line definition of new activities or the modification of the current ones (sometimes called dynamic institutions); and likewise with respect to different enforcement mechanisms or systems with multiple interacting institutions. Thus, valuable future work might include operations on the infrastructure (to add a scene, change a protocol, or add more terms to the language, for example), and the corresponding operations that would allow the infrastructure to accomplish this. Similarly, while the specific relationship between the infrastructure and the implementation of a concrete environment is out of the scope of this paper, we have illustrated how this might be done in Section 8. In general, one would assume an actual computational architecture, together with all the pragmatic aspects of connecting the domain language with the actual application domain entities (databases, anchoring of terms, identity and entitlements of participants, terms of liability, and so on), but again, addressing this in detail is left to future work.

Finally, we have shown that the electronic institution model is feasible, and that the model is applicable in practise.

11.3. Towards social intelligence

The Internet has brought about new forms of work, entertainment and social interaction that are digitally mediated, profusely interconnected and ever changing. In particular, it has fostered web-enabled interactions involving humans as well as software entities, situations in which multiple rational agents engage in a common endeavour. This reality summons the challenge of addressing it in the best tradition of artificial intelligence: to develop a principled way of studying social intelligence, to build systems that support it and to design and develop systems that share those features that make it interesting and successful.

One way to approach the challenge of social intelligence is to frame it as a mode of collective problem solving that involves two distinct phases: a first process through which those entities that participate in the problem reach an agreement, and a second one by which they put that agreement into practise. Only by manipulating (creating, modifying, exchanging) such agreements in meaningful ways can agents create and execute complex social processes. Since, in view of the autonomy

of individual agents, and the openness of the system in which agents may not know each other in advance, those social processes we have alluded to cannot in general be specified a priori, they need to be adaptive and, in consequence, agents must themselves come to agreements and then put them into practise.

It is this ability of agents to operate in environments in which they can understand, participate in, create and modify agreements that provides a foundation for agreement-based social interaction. Thus, software environments that enable social intelligence need to include the explicit generation and representation of social interaction process agreements (for example, jointly starting, jointly finishing or jointly entering an activity), on top of the simpler capacity to work together once the agreements are set. Our framework includes those aspects that we claim are necessary to support the processes of how an activity may be organised and put into action. In particular, we make explicit: (i) the requirements for meaningful communication among agents; (ii) the requirements to set up coordination or interaction processes; and (iii) the required operations that the environment needs to support social interactions and those that agents need in order to interoperate.

In this perspective, our model is a first approximation to the set of building blocks required by agreement computation and initial steps toward artificial social intelligence. Yet at the same time it provides a concrete model, supported by, and in support of, implemented and implementable tools and applications.

Acknowledgements

We are extremely grateful for the very detailed reviews and insightful comments of the anonymous reviewers. We are all, to a man, immensely grateful to Mas Puig for limitless hospitality over the very many years of completing this paper, and to Carme Roig for her tolerance, patience and support. Finally, thanks to Marc Esteva for discussions on his work on AMELI and electronic institutions in general. The IIIA researchers have been supported by the Agreement Technologies CONSOLIDER project under contract CSD2007-0022, EVE (TIN2009-14702-C02-01) and the Generalitat of Catalunya grant 2009-SGR-1434. The first author wishes to acknowledge the support by grant SAB2010051 under the *Subprogram of mobility stays of professors and researchers in Spanish centers (Ministry of Education)* which allowed him to take a sabbatical at IIIA.

Appendix A. Worked examples in the Z specification language

In this appendix, we briefly introduce the syntax of the Z specification language by way of a few examples. The most important construct in Z is the schema, which consists of two parts, the upper declarative part, which declares variables and their types, and the lower predicate part, which relates and constrains those variables. It is therefore appropriate to liken the semantics of a schema to that for Cartesian products. For example, suppose we define a schema as follows.

$$\begin{array}{|l} \hline Pair \\ first : \mathbb{N} \\ second : \mathbb{N} \\ \hline \end{array}$$

This is very similar to the following Cartesian product type.

$$Pair == \mathbb{N} \times \mathbb{N}$$

The difference between these forms is that there is no notion of order in the variables of the schema type. In addition, a schema may have a predicate part that can be used to constrain the state variables. Thus, we can state that the variable, *first*, can never be greater than *second*.

$$\begin{array}{|l} \hline Pair \\ first : \mathbb{N} \\ second : \mathbb{N} \\ \hline first \leq second \\ \hline \end{array}$$

An important aspect of Z is that schemas can be included within other schemas. We can select a state variable, *var*, of a schema, $s : S$, by writing $s.var$. For example, if we have a schema p of type $iPair$ (in other words, $p : Pair$), then $p.ifirst$ refers to the variable *first* in the schema *Pair*.

Now, operations in a state-based specification language are defined in terms of *changes to the state*. Specifically, an operation relates variables of the state after the operation (denoted by dashed variables) to the value of the variables before the operation (denoted by undashed variables). Operations may also have inputs (denoted by variables with question mark suffixes), outputs (with exclamation mark suffixes) and a precondition. In the *GettingCloser* schema below, there is an operation with an input variable, *new*?; if the value of *new*? lies between the values of variables *first* and *second*, then the value of *first* is replaced with the value of *new*?. The original value of *first* is output as *old*!. Here, the $\Delta Pair$ symbol is an

abbreviation for two state schemas, $Pair \wedge Pair'$ and, as such, includes in this schema all the variables and predicates of the state of *Pair* before and after the operation.

$$
\begin{array}{|l}
\hline
GettingCloser \\
\hline
new? : \mathbb{N} \\
\Delta Pair \\
old! : \mathbb{N} \\
\hline
first \leq new? \\
new? \leq second \\
first' = new? \\
second' = second \\
old! = first \\
\hline
\end{array}
$$

The *relation* type, expresses a mapping between *source* and *target* sets. The type of a relation with source X and target Y is $\mathbb{P}(X \times Y)$, and any element of this type (or *relation*) is a set of ordered pairs.

The definition of functions is also standard: relations are functions if no element from the source is related to more than one element in the target set. If every element in the source set is related, then the function is *total*; *partial* functions do not relate every source set element. Total functions are represented by $(\rightarrow)$ and *partial* functions by $(\nrightarrow)$. Sequences are simply special types of function where the domain consists of contiguous natural numbers.

$$\big\{(1, H), (2, E), (3, L), (4, L), (5, O)\big\} = \langle H, E, L, L, O\rangle$$

We introduce two examples of relations, *Rel*1 and *Rel*2, by way of illustration. *Rel*1 defines a *function* between nodes, while *Rel*2 defines a *sequence* of nodes. If the only elements of the source set are *node*1, *node*2 and *node*3, then the function is total, otherwise it is partial. In this case it is partial because *node*4 is not related.

$$Node ::= node1 \mid node2 \mid node3 \mid node4$$

$$
\begin{array}{|l}
rel1 : Node \leftrightarrow Node \\
rel2 : \text{seq}\, Node \\
\hline
rel1 = \{(node1, node2), (node2, node3), (node3, node2)\} \\
rel2 = \{(3, node3), (2, node2), (1, node4)\} \\
\end{array}
$$

The sequence *Rel*2 is commonly written:

$$\langle node4, node2, node3\rangle$$

The *domain* of a relation or function is the set of source elements that are related. Similarly, the *range* is the set of target set elements that are related. The *inverse* of a relation is obtained by reversing each of the ordered pairs so that the domain becomes the range, and the range becomes the domain. A relation can be restricted to a particular subset of its domain using *domain restriction*. There is a similar operation on the range called *range restriction*. A relation can be anti-restricted by a set in such a way such that the resulting relation does not contain any ordered pairs whose second element is in the restricting set. This is known as *anti-range restriction*, and a similar operation on the domain is called *anti-domain restriction*. Lastly, one relation can be updated by another relation using *relational overriding*, where the second relation can be considered as *new* information about its domain elements, overwriting any existing pairs whose first element is in the domain of the second. Examples of these operators can be seen below.

$$\text{dom}\, Rel1 = \{node1, node2, node3\}$$

$$\text{ran}\, Rel1 = \{node2, node3\}$$

$$\text{dom}\, Rel2 = \{1, 2, 3\}$$

$$\text{ran}\, Rel2 = \{node2, node3, node4\}$$

$$Rel1^{\sim} = \big\{(node2, node1), (node3, node2), (node2, node3)\big\}$$

$$\{node2, node3, node4\} \lhd Rel1 = \big\{(node2, node3), (node3, node2)\big\}$$

$$Rel1 \rhd \{node1, node2\} = \big\{(node1, node2), (node3, node2)\big\}$$

$$\{node1, node2\} \lhd\!\!\!- Rel = \big\{(node3, node2)\big\}$$

$$Rel1 \rhd\!\!\!- \{node1, node2\} = \big\{(node2, node3)\big\}$$

$$Rel1 \oplus \big\{(node1, node3), (node2, node2), (node2, node3)\big\}$$
$$= \big\{(node1, node3), (node2, node2), (node2, node3), (node3, node2)\big\}$$

Sets of elements can be defined using set comprehension. For example, the expression:

$$\{x : \mathbb{N} \mid x < 4 \bullet x * x\}$$

denotes the set:

$$\{0, 1, 4, 9\}$$

To state that, say, the square of every natural number greater than 10 is greater than 100, we write:

$$\forall n : \mathbb{N} \mid n > 10 \bullet n * n > 100$$

The expression:

$$\mu a : A \mid P$$

selects the unique element from the type A that satisfies the predicate P.

References

[1] Masahiko Aoki, Toward a Comparative Institutional Analysis, MIT Press, 2001.
[2] Gustavo Aranda, Tomas Trescak, Marc Esteva, Carlos Carrascosa, Building quests for online games with virtual institutions, in: Workshop on Agents for Games and Simulations at AAMAS 2010, Toronto, Canada, 10/05/2010, pp. 125–139.
[3] Josep Lluis Arcos, Pablo Noriega, Juan A. Rodriguez-Aguilar, Carles Sierra, E4Mas through electronic institutions, in: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Environments for Multi-Agent Systems III, 08/05/2006, in: Lecture Notes in Computer Science, vol. 4389, Springer, Berlin/Heidelberg, 2007, pp. 184–202.
[4] Josep Lluis Arcos, Juan A. Rodriguez-Aguilar, Bruno Rosell, Engineering autonomic electronic institutions, in: Danny Weyns, Sven Brueckner, Yves Demazeau (Eds.), Engineering Environment-Mediated Multi-Agent Systems, in: Lecture Notes in Computer Science, vol. 5049, Springer, Berlin/Heidelberg, 2008, pp. 76–87.
[5] Estefanía Argente, Vicent Botti, Vicente Julian, Gormas: An organizational-oriented methodological guideline for open mas, in: Proceedings of the 10th International Conference on Agent-Oriented Software Engineering, AOSE'10, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 32–47.
[6] Alexander Artikis, Formalising dynamic protocols for open agent systems, in: International Conference on Artificial Intelligence & Law (ICAIL), 2009, pp. 68–77.
[7] Alexander Artikis, Marek Sergot, Jeremy Pitt, An executable specification of a formal argumentation protocol: Argumentation in artificial intelligence, Artificial Intelligence 171 (10–15) (2007) 776–804.
[8] Alexander Artikis, Marek Sergot, Jeremy Pitt, Specifying norm-governed computational societies, ACM Transactions on Computational Logic 10 (January 2009) 1:1–1:42.
[9] Ronald Ashri, Terry Payne, Michael Luck, Mike Surridge, Carles Sierra, Juan A. Aguilar, Rodriguez-Aguilar, Pablo Noriega, Using electronic institutions to secure grid environments, in: Matthias Klusch, Michael Rovatsos, Terry Payne (Eds.), Cooperative Information Agents X, in: Lecture Notes in Computer Science, vol. 4149, Springer, Berlin/Heidelberg, 2006, pp. 461–475.
[10] Mihai Barbuceanu, Tom Gray, Serge Mankovski, Coordinating with obligations, in: Proceedings of the Third International Conference on Autonomous Agents (AGENTS '99), ACM Press, 1998, pp. 62–69.
[11] Federico Bergenti, Marie-Pierre Glezies, Franco Zambonelli (Eds.), Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook, Kluwer Academic, Dordrecht, 2004.
[12] Tim Berners-Lee, James Hendler, Ora Lassila, The semantic web, Scientific American 284 (5) (2001) 34–43.
[13] Guido Boella, Pablo Noriega, Gabriella Pigozzi, Harko Verhagen (Eds.), Normative Multi-Agent Systems, 15/03/2009, Dagstuhl Seminar Proceedings, vol. 09121, Schloss Dagstuhl, Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2009.
[14] Guido Boella, Gabriella Pigozzi, Leender Van der Torre, Five guidelines for normative multiagent systems, in: Guido Governatore (Ed.), Legal Knowledge and Information Systems, JURIX, October 22–24, 2009, IOS Press, Amsterdam, 2009, pp. 21–30.
[15] Guido Boella, Leendert van der Torre, Harko Verhagen, Introduction to the special issue on normative multiagent systems, Autonomous Agents and Multi-Agent Systems 17 (2008) 1–10.
[16] Anton Bogdanovych, Simeon Simoff, Establishing social order in 3D virtual worlds with virtual institutions, in: Alan Rea (Ed.), Security in Virtual Worlds, 3D Webs, and Immersive Environments: Models for Development, Interaction, and Management, IGI Global, 2011, pp. 140–169.
[17] Jeremy Boile, Michael Cardella, Stany Blanyalet, Matjaz Juric, Sean Carey, Chandran Praveen, Yves Coene, Kevin Geminiuc, Markus Zirn, Harish Gaur, BPEL Cookbook, Packt Publishing, 2006.
[18] Eva Bou, Maite Lopez-Sanchez, Juan A. Rodriguez-Aguilar, Adaptation of autonomic electronic institutions through norms and institutional agents, in: Gregory O'Hare, Alessandro Ricci, Michael O'Grady, Oguz Dikenelli (Eds.), Engineering Societies in the Agents World VII, in: Lecture Notes in Computer Science, vol. 4457, Springer, Berlin/Heidelberg, 2007, pp. 300–319.
[19] Ismel Brito, Nardine Osman, Jordi Sabater-Mir, Carles Sierra, CHARMS: A Charter Management System. Automating the integration of electronic institutions and humans, in: 8th European Workshop on Multi-Agent Systems (EUMAS'10), Paris, France, 16/12/2010.
[20] Ismel Brito, Isaac Pinyol, Daniel Villatoro, Jordi Sabater-Mir, HIHEREI: Human Interaction within Hybrid Environments, in: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), Budapest, Hungary, 2009, pp. 1417–1418.
[21] Jordi Campos, Maite Loopez-Sanchez, Marc Esteva, Using a two-level multi-agent system architecture, in: Marina De Vos, Nicoletta Fornara, Jeremy V. Pitt, George Vouros (Eds.), Coordination, Organization, Institutions and Norms in Agent Systems VI (COIN 2010), in: Lecture Notes in Computer Science, vol. 6541, Springer, Berlin/Heidelberg, 2011, pp. 303–320.
[22] Luca Cardelli, Andrew D. Gordon, Mobile ambients, in: Proceedings of the First international Conference on Foundations of Software Science and Computation Structure, in: Lecture Notes in Computer Science, vol. 1378, Springer, 1998, pp. 140–155.
[23] Amit Chopra, Munindar Singh, Nonmonotonic commitment machines, in: Frank Dignum (Ed.), Advances in Agent Communication, in: Lecture Notes in Computer Science, vol. 2922, Springer, Berlin/Heidelberg, 2004, p. 1959.
[24] Amit K. Chopra, Munindar P. Singh, Specifying and applying commitment-based business patterns, in: Tumer, Yolum, Sonenberg, Stone (Eds.), Proceedings of the 10th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011, International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 475–482.

[25] Owen Cliffe, Marina de Vos, Julian Padget, Specifying and reasoning about multiple institutions, in: Coordination, Organization, Institutions and Norms in Agent Systems II, COIN 2007, in: Lecture Notes in Computer Science, vol. 4386, Springer, Berlin/Heidelberg, 2007, pp. 67–85.
[26] Owen Cliffe, Marina De Vos, Julian A. Padget, Answer set programming for representing and reasoning about virtual institutions, in: CLIMA VII, 2006, pp. 60–79.
[27] Marco Colombetti, A commitment-based approach to agent speech acts and conversations, in: Proceedings of the Fourth International Conference on Autonomous Agents, Workshop on Agent Languages and Conversation Policies, 2000, pp. 21–29.
[28] Rosaria Conte, Rino Falcone, Giovanni Sartor, Introduction: Agents and norms: How to fill the gap? Artificial Intelligence and Law 7 (1999) 1–15.
[29] Stephen Cranefield, Modelling and monitoring social expectations in multi-agent systems, in: Pablo Noriega, Javier Vázquez-Salceda, Guido Boella, Olivier Boissier, Virginia Dignum, Nicoletta Fornara, Eric Matson (Eds.), Coordination, Organizations, Institutions, and Norms in Agent Systems II, in: Lecture Notes in Computer Science, vol. 4386, Springer, Berlin/Heidelberg, 2007, pp. 308–321.
[30] Natalia Criado, Estefania Argente, Antonio Garrido, Juan Gimeno, Francesc Igual, Vicente Botti, Pablo Noriega, Adriana Giret, Norm enforceability in electronic institutions? in: Marina De Vos, Nicoletta Fornara, Jeremy Pitt, George Vouros (Eds.), Coordination, Organizations, Institutions, and Norms in Agent Systems VI, in: Lecture Notes in Computer Science, vol. 6541, Springer, Berlin/Heidelberg, 2011, pp. 250–267.
[31] Guifré Cuní, Marc Esteva, Pere Garcia, Eloi Puertas, Carles Sierra, Teresa Solchaga. Masfit, Multi-agent systems for fish trading, in: 16th European Conference on Artificial Intelligence (ECAI 2004), Valencia, Spain, August 2004, pp. 710–714.
[32] Mehdi Dastani, Davide Grossi, John-Jules Meyer, Nick Tinnemeier, Normative multi-agent programs and their logics, in: Guido Boella, Pablo Noriega, Gabriella Pigozzi, Harko Verhagen (Eds.), Normative Multi-Agent Systems, Dagstuhl, Germany, 2009, in: Dagstuhl Seminar Proceedings, vol. 09121, Schloss Dagstuhl, Leibniz-Zentrum fuer Informatik, Germany, 2009.
[33] Mehdi Dastani, Nick A.M. Tinnemeier, John-Jules C. Meyer, A programming language for normative multi-agent systems, in: Virginia Dignum (Ed.), Multi-Agent Systems: Semantics and Dynamics of Organizational Models, IGI Global, 2009, pp. 397–414.
[34] Chrysanthos Dellarocas, Mark Klein, Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces, in: Alexandros Moukas, Fredrik Ygge, Carles Sierra (Eds.), Agent Mediated Electronic Commerce II, in: Lecture Notes in Computer Science, vol. 1788, Springer, Berlin/Heidelberg, 2000, pp. 24–39.
[35] Virginia Dignum, John-Jules Meyer, Hans Weigand, Towards an organizational model for agent societies using contracts, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2, AAMAS '02, ACM, New York, NY, USA, 2002, pp. 694–695.
[36] Virginia Dignum, Javier Vázquez-Salceda, Frank Dignum, OMNI: Introducing social structure, norms and ontologies into agent organizations, in: PROMAS, in: Lecture Notes in Computer Science, vol. 3346, Springer, 2004, pp. 181–198.
[37] Mark d'Inverno, Michael Fisher, Alessio Lumuscio, Michael Luck, Maarten de Rijke, Mark Ryan, Michael Wooldridge, Formalisms for multi-agent systems, Knowledge Engineering Review 12 (3) (1997) 315–321.
[38] Mark d'Inverno, David Kinny, Michael Luck, Interaction protocols in agentis, in: Third International Conference on Multi-Agent Systems, ICMAS'98, IEEE Computer Society, Paris, France, 1998, pp. 112–119.
[39] Mark d'Inverno, Michael Luck, Michael P. Georgeff, David Kinny, Michael Wooldridge, The dMARS architechure: A specification of the distributed multi-agent reasoning system, Autonomous Agents and Multi-Agent Systems 9 (1–2) (2004) 5–53.
[40] Marc Esteva, Electronic Institutions: From specification to development, PhD Thesis Universitat Politècnica de Catalunya (UPC), 2003. Number 19 in IIIA Monograph Series. IIIA, 2003.
[41] Marc Esteva, David de la Cruz, Carles Sierra, ISLANDER: An electronic institutions editor, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '02), July 2002, ACM Press, 2002, pp. 1045–1052.
[42] Marc Esteva, Juan A. Rodriguez-Aguilar, Josep Lluis Arcos, Carles Sierra, Socially-aware lightweight coordination infrastructures, in: 12th International Workshop on Agent-Oriented Software Engineering, AAMAS'11, 2011, pp. 117–128.
[43] Marc Esteva, Juan A. Rodriguez-Aguilar, Bruno Rosell, Josep L. Arcos, AMELI: An agent-based middleware for electronic institutions, in: Carles Sierra, Liz Sonenberg (Eds.), Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '04), vol. 1, July 19–23, 2004, IFAAMAS, ACM Press, New York, USA, 2004, pp. 236–246.
[44] Marc Esteva, Juan A. Rodríguez-Aguilar, Josep Lluis Arcos, Carles Sierra, Pablo Noriega, Bruno Rosell, Electronic Institutions Development Environment, in: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '08), Estoril, Portugal, 12/05/2008, International Foundation for Autonomous Agents and Multiagent Systems, ACM Press, 2008, pp. 1657–1658.
[45] Jacques Ferber, Olivier Gutknecht, Fabien Michel, From agents to organizations: An organizational view of multi-agent systems, in: Paolo Giorgini, Jörg P. Müller, James Odell (Eds.), Agent-Oriented Software Engineering IV, in: Lecture Notes in Computer Science, vol. 2935, Springer, Berlin/Heidelberg, 2003, pp. 443–459.
[46] Jacques Ferber, Olivier Gutknetch, 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), 1998, pp. 128–135.
[47] FIPA (2002), FIPA abstract architecture specification, Technical report, FIPA – Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00001/SC00001L.pdf.
[48] Roberto Flores, Philippe Pasquier, Brahim Chaib-draa, Conversational semantics sustained by commitments, Autonomous Agents and Multi-Agent Systems 14 (2007) 165–186.
[49] Nicoletta Fornara, Marco Colombetti, Defining interaction protocols using a commitment-based agent communication language, in: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '03, ACM, New York, NY, USA, 2003, pp. 520–527.
[50] Nicoletta Fornara, Marco Colombetti, Specifying and enforcing norms in artificial institutions, in: Matteo Baldoni, Tran Son, M. van Riemsdijk, Michael Winikoff (Eds.), Declarative Agent Languages and Technologies VI, in: Lecture Notes in Computer Science, vol. 5397, Springer, Berlin/Heidelberg, 2009, pp. 1–17.
[51] Nicoletta Fornara, Francesco Viganó, Mario Verdicchio, Marco Colombetti, Artificial institutions: A model of institutional reality for open multiagent systems, Artificial Intelligence and Law 16 (2008) 89–105.
[52] Ian Foster, Carl Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1998.
[53] Dorian Gaertner, Garcia-Camino Andres, Pablo Noriega, Juan A. Rodriguez-Aguilar, Wamberto W. Vasconcelos, Distributed norm management in regulated multi-agent systems, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07), Honolulu, Hawaii, 14/05/2007, ACM Press, 2007, pp. 624–631.
[54] Andres Garca-Camino, Pablo Noriega, Juan A. Rodriguez-Aguilar, Implementing norms in electronic institutions, in: Simon Thompson, Michal Pechoucek, Donald Steiner (Eds.), Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '05), ACM Press, Utrecht, NL, 2005, pp. 667–673.
[55] Andres Garcia-Camino, Normative regulation of open multi-agent systems, PhD Thesis Universitat Autònoma de Barcelona, 2010, No. 35, Monograph Series IIIA, 2011.
[56] Andres Garcia-Camino, Juan A. Rodriguez-Aguilar, Carles Sierra, Wamberto Vasconcelos, Constraint rule-based programming of norms for electronic institutions, Autonomous Agents and Multi-Agent Systems 18 (1) (2009) 186–217.
[57] Antonio Garrido, Adriana Giret, Pablo Noriega, mWater: A sandbox for agreement technologies, in: Artificial intelligence Research and Development (CCIA-2009), IOS Press, 2009, pp. 252–261.

[58] Les Gasser, Carl Braganza, Nava Herman, MACE: A flexible test-bed for distributed AI research, in: M.N. Huhns (Ed.), Distributed Artificial Intelligence, Pitman Publishers, 1987, pp. 119–152.
[59] Guido Governatori, Representing business contracts in RuleML, International Journal of Cooperative Information Systems 14 (2–3) (2005) 181–216.
[60] Davide Grossi, Huib Aldewereld, Frank Dignum, Ubi lex, ibi poena: Designing norm enforcement in e-institutions, in: Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems II, in: Lecture Notes in Computer Science, vol. 4386, Springer, 2007, pp. 101–114.
[61] Carl Hewitt, Offices are open systems, ACM Transactions of Office Automation Systems 4 (3) (1986) 271–287.
[62] Jomi Fred Hübner, Olivier Boissier, Rosine Kitio, Alessandro Ricci, Instrumenting multi-agent organisations with organisational artifacts and agents, Autonomous Agents and Multi-Agent Systems 20 (3) (2010) 369–400.
[63] Jomi Fred Hübner, Jaime Simo Sichman, Olivier Boissier, A model for the structural, functional, and deontic specification of organizations in multiagent systems, in: Proceedings of the 16th Brazilian Symposium on Artificial Intelligence (SBIA'02), 2002, pp. 118–128.
[64] Nicholas R. Jennings, On agent-based software engineering, Artificial Intelligence 117 (2) (2000) 277–296.
[65] Nicholas R. Jennings, Peyman Faratin, Mark J. Johnson, Paul O. O'Brien, Mike E. Wiegand, Using intelligent agents to manage business processes, in: Proceedings of the First International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96), 1996, pp. 345–360.
[66] Nicholas R. Jennings, Katia Sycara, Michael Wooldridge, A roadmap of agent research and development, Autonomous Agents and Multi-Agent Systems 1 (1998) 275–306.
[67] Andrew Jones, Marek Sergot, A formal characterization of institutionalized power, Logic Journal of the IGPL 4 (3) (1996) 427–446.
[68] Adam I. Juda, David C. Parkes, An options-based solution to the sequential auction problem, Artificial Intelligence 173 (7–8) (2009) 876–899.
[69] Robert A. Kowalski, Marek J. Sergot, Computer representation of the law, in: IJCAI, 1985, pp. 1269–1270.
[70] Yannis Labrou, Tim Finin, A proposal for a new KQML specification, Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250, February 1997.
[71] Henrique Lopes Cardoso, Eugenio Oliveira, Virtual enterprise normative framework within electronic institutions, in: 5th International Workshop on Engineering Societies in the Agents World, Toulouse, October 2004.
[72] Henrique Lopes Cardoso, Eugenio Oliveira, Electronic institutions for B2B dynamic normative environments, Artificial Intelligence and Law 16 (2008) 107–128.
[73] Henrique Lopes Cardoso, Eugenio C. Oliveira, Institutional reality and norms: Specifying and monitoring agent organizations, International Journal of Cooperative Information Systems 16 (1) (2007) 67–95.
[74] Fabiola Lopez y Lopez, Michael Luck, Mark d'Inverno, Constraining autonomy through norms, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: part 2 (AAMAS '02), ACM Press, 2002, pp. 674–681.
[75] Fabiola Lopez y Lopez, Michael Luck, Mark d'Inverno, Normative agent reasoning in dynamic societies, in: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '04), ACM Press, 2004, pp. 732–739.
[76] Jarred McGinnis, David Robertson, Dynamic and distributed interaction protocols, in: Proceedings of the AISB 2004 Convention, 2004, pp. 45–54.
[77] Robin Milner, Joachim Parrow, David Walker, A calculus of mobile processes, Parts I and II, Journal of Information and Computation 100 (September 1992) 1–77.
[78] Pablo Noriega, Agent-mediated auctions: The fishmarket metaphor, PhD thesis Universitat Autònoma de Barcelona, 1997, No. 8, Monograph Series IIIA, 1999.
[79] Pablo Noriega, Carlos López de Toro, Towards a platform for on-line mediation, in: Marta Poblet, Uri Shild, John Zeleznikow (Eds.), Proc. Workshop on Legal and Negotiation Decision Support Systems (LDSS 2009) in conjunction with ICAIL 2009, 12/06/2009, CEUR Workshop Proceedings, Barcelona, 2009, pp. 67–75.
[80] Douglass C. North, Institutions, Journal of Economic Perspectives 5 (1) (1991) 97–112.
[81] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Artifacts in the A&A meta-model for multi-agent systems, Autonomous Agents and Multi-Agent Systems 17 (3) (2008) 432–456.
[82] Osman Nardine, Carles Sierra, Jordi Sabater-Mir, Joseph R. Wakeling, Judith Simon, Gloria Origgi, Roberto Casati, Liquidpublications and its technical and legal challenges, in: Danièle Bourcier, Pompeu Casanovas, Mélanie Dulong de Rosnay, Catharina Maracke (Eds.), Intelligent Multimedia: Managing Creative Works in a Digital World, vol. 8, European Press Academic Publishing, Florence, 2010, pp. 321–336.
[83] Elinor Ostrom, An agenda for the study of institutions, Public Choice 48 (1) (1986) 3–25.
[84] Elinor Ostrom, Institutional Rational Choice: An Assessment of the Institutional Analysis and Development Framework, Westview Press, 1999, pp. 35–71.
[85] Young pa So, Edmund H. Durfee, Simulating Organizations. Computational Models of Institutions and Groups, chapter Designing Organizations for Computational Agents, The MIT Press, 1998, pp. 47–63.
[86] Lin Padgham, Michael Winikoff, Prometheus: A methodology for developing intelligent agents, in: Fausto Giunchiglia, James Odell, Gerhard Weiß (Eds.), Agent-Oriented Software Engineering III, in: Lecture Notes in Computer Science, vol. 2585, Springer, Berlin/Heidelberg, 2003, pp. 174–185.
[87] H. Edward Pattison, Daniel D. Corkill, Victor R. Lesser, Instantiating descriptions of organizational structures, in: M.N. Huhns (Ed.), Distributed Artificial Intelligence, Pitman Publishers, 1987, pp. 59–96.
[88] Walter W. Powell, Paul J. Dimaggio, The New Intitutionalism in Organizational Analysis, University of Chicago Press, 1991.
[89] Michael Prietula, Kathleen Carley, Les Gasser (Eds.), Simulating Organizations: Computational Models of Institutions and Groups, vol. 1, 1st edition, The MIT Press, 1998.
[90] Vijay Rajan, James R. Slagle, The use of artificially intelligent agents with bounded rationality in the study of economic markets, in: AAAI-96, 1996, pp. 102–107.
[91] Sarvapali D. Ramchurn, Carles Sierra, Lluis Godo, Nicholas R. Jennings, Negotiating using rewards, Artificial Intelligence 171 (10–15) (2007) 805–837.
[92] Paolo Remagnino, Gian Luca Foresti, Tim Ellis (Eds.), Ambient Intelligence: A Novel Paradigm, Springer, 2004.
[93] David Robertson, A lightweight coordination calculus for agent systems, in: Declarative Agent Languages and Technologies, vol. 3476, DALT 2004, Springer, 2005, pp. 183–197.
[94] Armando Robles, Pablo Noriega, Francisco Cantú, An agent oriented hotel information system, in: Keith S. Decker, Jaime Simão Sichman, Carles Sierra, Cristiano Castelfranchi (Eds.), Proceedings of 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), Budapest, Hungary, 10/05/2009, International Foundation for Autonomous Agents and Multiagent Systems, ACM Press, 2009, pp. 1415–1416.
[95] Juan A. Rodriguez-Aguilar, On the design and construction of agent-mediated electronic institutions, PhD thesis, Universitat Autònoma de Barcelona, 2001, No. 14, Monograph Series IIIA, 2003.
[96] Juan A. Rodriguez-Aguilar, Francisco J. Martín, Pablo Noriega, Pere Garcia, Carles Sierra, Towards a test-bed for trading agents in electronic auction markets, AI Communications 11 (1) (1998) 5–19.
[97] Juan A. Rodriguez-Aguilar, Pablo Noriega, Carles Sierra, Julian Padget, Fm96.5 a java-based electronic auction house, in: Proceedings of the Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97), 1997, pp. 207–224.
[98] John R. Searle, What is an institution? Journal of Institutional Economics 1 (01) (2005) 1–22.
[99] John W. Shepherdson, Simon G. Thompson, Brian Odgers, Cross organisational workflow co-ordinated by software agents, in: Proceedings of the Workshop on Cross-Organisational Workflow Management and Co-ordination, 1999.

[100] Carles Sierra, Pablo Noriega, Institucions electròniques, in: Proceedings of the Primer Congrès Català d'Intelligència Artificial, 1998, pp. 125–129.
[101] Carles Sierra, John Thangarajah, Lin Padgham, Michael Winikoff, Designing institutional multi-agent systems, in: AOSE, 2006, pp. 84–103.
[102] Herbert A. Simon, The Sciences of the Artificial, third edition, MIT Press, 1996.
[103] Munindar P. Singh, Agent communication languages: Rethinking the principles, IEEE Computer 31 (12 (Dec.)) (1998) 40–47.
[104] Munindar P. Singh, A social semantics for agent communication languages, in: Proceedings of the 1999 IJCAI Workshop on Agent Communication Languages, in: Lecture Notes in Computer Science, vol. 1916, Springer, 2000, pp. 31–45.
[105] J. Michael Spivey, The Z Notation, second edition, Prentice Hall International, Hemel Hempstead, England, 1992.
[106] Tomas Trescak, Marc Esteva, Inmaculada Rodriguez, Vixee an innovative communication infrastructure for virtual institutions, in: Proceedings of the 10th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei (Taiwan), 03/05/2011.
[107] Wamberto W. Vasconcelos, David Robertson, Jaume Agustí-Cullell, Carles Sierra, Jordi Sabater-Mir, Simon Parsons, Chris Walton, Michael Wooldridge, A lifecycle for models of large multi-agent systems, in: M. Wooldridge, G. Weiss, P. Ciancarini (Eds.), Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001, in: Revised Papers and Invited Contributions, vol. 2222, Springer-Verlag, 2002, pp. 307–325.
[108] Javier Vázquez-Salceda, The harmonia framework, KI 19 (1) (2005) 38.
[109] Javier Vázquez-Salceda, Huib Aldewereld, Davide Grossi, Frank Dignum, From human regulations to regulated software agents' behavior, Artificial Intelligence and Law 16 (2008) 73–87.
[110] Danny Weyns, Andrea Omicini, James Odell, Environment as a first class abstraction in multiagent systems, Autonomous Agents and Multi-Agent Systems 14 (1) (2007) 5–30.
[111] Michael Winikoff, Wei Liu, James Harland, Enhancing commitment machines, in: DALT Workshop, LNAI, Springer-Verlag, 2005, pp. 198–220.
[112] Pinar Yolum, Munindar P. Singh, Flexible protocol specification and execution: Applying event calculus planning using commitments, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2, AAMAS '02, ACM, New York, NY, USA, 2002, pp. 527–534.