Interaction Model Language Definition

Sindhu Joseph[1], Adrian Perreau de Pinninck[1], Dave Robertson[2],
Carles Sierra[1], Chris Walton[2]

1 Artificial Intelligence Research Institute, IIIA-CSIC, Spain
2 Informatics, University of Edinburgh, UK

Abstract. In this paper we introduce *ambient LCC*, a language to agent interaction models for P2P networks. The language is based on process algebra concepts and combines the notions of Light weight coordinate calculus(LCC) and ambient calculus. It is specially designed to support the execution of electronic institutions, an organization model for Multi Agent Systems.

1 Introduction

During recent years, Electronic institutions (EI) have proven to be one of the interesting and effective organization models for MAS communication and coordination [8, 5, 9, 4]. In the EI approach, protocols are expressed diagrammatically using graphs of finite-state automata (i.e. state-charts). The EI approach has two levels of abstraction: scenes in which agents communicate, and institutions which are composed of scenes. These two levels of abstraction make it possible to compose large protocols from smaller ones. Thus EIs combine the benefits of diagrammatic specification with levels of abstraction which modularize the protocol specification. EI is also a promising technology to tackle the complexity inherent in open systems due to the need for adequate interoperation of heterogeneous, independent, distributed, autonomous components/agents. One of the challenges that face the centralized EI model in this regard is the lack of a distributed specification. This becomes ever more relevant as the MAS community moves closer to P2P networks.

The light weight coordination calculus (LCC) proposed by Dave Robertson [11] is a protocol language designed to be completely peer to peer and does not rely on any centralized infrastructure. Being a process calculus variant, it has the advantage of being based on a powerful theoretical foundation. LCC can be used as a distributed interaction language to translate the centralized EI protocol. The EI model has two levels of abstraction: scenes in which agents communicate, and institutions which are composed of scenes. These two levels of abstraction make it possible to compose large protocols from smaller ones whereas LCC has only one level of abstraction, which is loosely equivalent to a single scene. A theoretical framework, an extension of LCC language with the additional layers of abstraction from EI is proposed in this paper. The theoretical basis for our unified approach is the Ambient Calculus [1] which is a process calculus[1] which gives us the notion of bounded locations (ambients) in addition to the normal concurrency operations.

[1] Process calculus is a family of related approached to the formal specification of concurrent systems. Process calculus enable the high-level description of interactions, communications, and synchronisations between a collection of agents. Furthermore, the algebraic laws of process calculus allow these descriptions to be manipulated and analysed via formal reasoning techniques.

The introduction of ambients gives us the ability to formally define scene composition, and the transitions between scenes, which were previously lacking in LCC. This is specially important as breaking protocols into small pieces (or combining small pieces to obtain more complex ones) is a basic software engineering need that ambient calculus facilitates. Furthermore, the EI concept has a hierarchical notion of scene (or activity) that maps easily into the ambient concept. Electronic institution concepts like joining or leaving a scene can naturally be mapped into *in* and *out* operations over ambients, and information models can be represented as accessible variables within the ambient. The notion of state, so important in P2P protocols, can then be lodged in an ambient and move with it. The potential loss of good synchronisation properties that a centralised EI approach has can be also recovered by the ambient synchonisation operations. Finally, the concept of ambient is very lightweight and model checking can be applied to interaction models written in ambient LCC.

We considered a number of already existing alternatives. Dynamic Logic [6] is a powerful specification tool to describe the consequences of (possibly concurrent) actions in the evolution of the state of computer systems. However, no efficient verification techniques exist, and the conceptual distance between the notion of electronic institution inspiring our approach and dynamic logic concepts is far too large. BPEL4WS [12] is an orchestration language for business stateful protocols over web services. However, it lacks the rich social structure and role flow that electronic institutions permit, as well as it does not provide support for norm specification.

The Web Services Choreography Description Language (WS-CDL) http://www.w3.org/TR/ws-cdl-10/ has a number of similarities to the approach that we define here. For example, it allows interactions between participants to be defined using sequence, choice and parallel operations. It also defines specific roles for the participants within the interaction, and permits separate interactions to be composed. However, WS-CDL has no formal semantics, and it is unclear how WS-CDL specifications should be enacted. There is no basis on which to verify interactions, or check that compositions are meaningful. Furthermore, there are no higher-level abstractions, such as scenes and institutions as provided in the EI approach. The WS-CDL language is also incomplete and the specification is still under development at the time of writing.

The next section provides a brief introduction to the concepts required to understand this work. The following sections introduce Ambient LCC syntax and the mapping of Electronic institution concepts into the newly introduced language. Finally we also provide an example translating an EI scene into ambient LCC syntax.

2 Background material

Each of the concepts in the following subsections require an in depth treatment. However due to space constraints, a detailed discussion on the background materials is beyond the scope of this paper, hence interested readers are adviced look into the references.

2.1 Electronic Institutions

The idea behind EIs is to mirror the roles traditional institutions play in the establishment of "the rules of the game"–a set of conventions that articulate agents' interactions– but in our case

applied to agents (humans or software entities) that interact through messages whose (socially relevant) effects are known to interacting parties. The essential roles EIs play are both descriptive and prescriptive: the institution makes the conventions explicit to participants, and it warrants their compliance[2].

EIs —as artifacts— involve a conceptual framework to describe agent interactions as well as an engineering framework to specify and deploy actual interaction environments. We have been developing the EI artifact for some time and advocating that open MAS can be properly designed and implemented with it [8, 5, 9, 4]. Our experiences in the deployment of applications as EIs, e.g. [10, 2] make us confident of the validity of this approach. We look at EIs as a framework for developing multiagent systems (MAS). We do so for two reasons, first because open systems can be viewed as a type of MAS, where the entities that interoperate in the open system are simply thought of as agents. Secondly, because, in that light, some recent methodologies and conceptual proposals for MAS engineering are then relevant for open systems. Our approach, has things in common with some of those methodologies and conceptual proposals, however we believe that it contributes to the engineering of this type of MAS through three salient distinctive features:

1. It is socially-centered, and neutral with respect to the participating agents internals and the application domain of their interaction
2. It has a uniform conceptual framework to manage components and interactions that prevails through the different views (high-level specification, implementation, monitoring, ...) of a given system.
3. It has an interaction-centered methodology that is embedded in a suit of software tools that support the system development cycle from specification to deployment.

2.2 The Lightweight Coordination Calculus (LCC)

LCC can be considered as a heavily-sugared variant of the π-calculus [7] with an asynchronous semantics. The extensions to the core calculus are designed to make the language more suited to the concepts found in multi-agent systems and dialogues. LCC was designed specifically for expressing Peer-to-Peer (P2P) style interactions within multi-agent systems, i.e. without any central control.The abstract syntax of LCC is presented in Figure 1.

Where *null* denotes an event which does not involve message passing; $Term$ is a structured term and Id is either a variable or a unique identifier for the agent.

There are five key syntactic categories in the definition, namely: $Framework$, $Clause$, $Agent$, Dn (Definition), and $Message$. These categories have the following meanings. A $Framework$, which bounds an interaction in our definition, comprises a set of clauses. Each $Clause$ corresponds to an agent, and each agent has a unique name a and a $Type$ which defines the role of the agent. The interactions that the agent must perform are given by a definition Dn. These definitions may be composed as sequences (*then*), choices (*or*), or in parallel (*par*). The actual interactions

[2] In terms of Simon's engineering design abstractions, EIs are the –social– interface layer between the problem space the participating systems deal with, on one side, and the internal decision or functional intricacies of the various participating systems, on the other.

$$
\begin{aligned}
Framework &:= \{Clause, \ldots\} \\
Clause &:= Agent :: Dn \\
Agent &:= a(Type, Id) \\
Dn &:= Agent \mid Message \mid Dn\ then\ Dn \mid Dn\ or\ Dn \mid Dn\ par\ Dn \mid null \leftarrow C \\
Message &:= M \ \Rightarrow\ Agent \mid M \ \Rightarrow\ Agent \leftarrow C \mid M \ \Leftarrow\ Agent \mid C \leftarrow M \ \Leftarrow\ Agent \\
C &:= Term \mid C \wedge C \mid C \vee C \\
Type &:= Term \\
M &:= Term
\end{aligned}
$$

Fig. 1. Abstract Syntax of LCC.

between agents are given by $Message$ definitions. Messages involve sending ($\Rightarrow$) or receiving ($\Leftarrow$) of terms M from another agent, and these exchanges may be constrained by C.

2.3 A Calculus of Mobile Ambients

The ambient calculus was developed as a way to express mobile computation. It can also be considered as an extension of the basic operators of the π-calculus [7]. The inspiration behind the ambient calculus is the observation that many aspects of mobility involve administrative considerations. For example, the authorisation to enter or exit a domain, and the permission to execute code in a particular domain. These issues were principally motivated by the needs of mobile devices. However, they are very similar to the issues faced by agents in an open environment, e.g. the Internet.

The ambient calculus addresses the administrative considerations by defining an ambient (informally) as a "bounded space where computation happens". The crucial point is the existence of a boundary, which determines what is inside and outside the ambient. This boundary is analogous to a firewall. Ambients can also be nested, leading to an administrative hierarchy, which is a commonly occurring structure on the Internet, e.g. intra-nets, and demilitarised zones. An ambient is also something that can be moved. For example, to represent a computer or agent moving from one place to another.

More precisely, each ambient has a name, a collection of local agents that run directly within the ambient, and a collection of sub-ambients. The syntactic categories are processes (P, Q, and R), and capabilities (M). A process is analogous to an individual agent. A process may be placed inside an ambient, may be replicated, and may be composed in parallel with another process, which means that the processes execute together. We write $n[P]$ to denote an ambient named n, which contains the process (i.e. agent) P. For more information on ambient calculus refer [1]

3 Ambient LCC

Ambient LCC inherits the notion of ambient, the most important extension to LCC, from the theory of ambient calculus. Most of the properties of an ambient as defined by Cardelli stand valid in the context of ambient LCC as well. The following highlights the structure and properties of ambient LCC.

- **Ambient Structure** An ambient in Ambient LCC consists of two logical components, an ambient specification and an execution environment. The ambient specification specifies the signature of the ambient. It provides the details of what are the parameter values that define the ambient. This may include information such as how many agent processes can be active in an ambient at a given time, what are the entry and exit conditions for agent processes and what are the rules governing communication and message passing within the ambient environment, and so on. The execution environment provides the execution state of the ambient to hold the execution parameters. An ambient is also defined recursively to allow layers of abstraction in the protocol definition which can be mapped to a tree structure. At each layer, ambients correspond to bounded places, where processes can interact, with ambients providing the context for interaction.
- **Ambient Movement** A process entering an ambient must necessarily enter through the top layer of the ambient tree, and can successively traverse the tree structure. A process *in* any ambient node in the tree is necessarily in all the parent ambient nodes. Similarly if a process wants to come *out* of a parent ambient, it should do so only after it exits the child ambient. Processes are also given capabilities to create and destroy ambients upon satisfaction of certain conditions. Another concept that needs a mention here is the property of processes *holding* ambients. As Ambient LCC works in a distributed setting, there is no assumption of a global infrastructure. Processes are the only entities having state and hence capable of holding ambients. Hence, at any given point in time, all the ambient execution environments reside in some process and this process has special privileges to modify the ambient. This restricted update mechanism is used to implement the synchronization needed for agent communication.
- **Agent ambients** There is a special kind of ambient defined in the language, and associated with agent processes. Though agents can be naturally associated with processes, an *agent ambient* is meant to keep the state of an *agent process*. Most of the computation happens in ambients that may contain more than one agent, and the state of these computations is preserved in the execution environment of the enclosing ambient. Yet there are certain parameters that are relevant to an agent which need to be preserved throughout the life of an agent as the agent process moves from one ambient to another. The agent ambient is there to preserve such process specific states. Thus, in this context, agents are treated as special ambients with a process and an ambient around them. The agent ambients do not get separated from the agent processes at any point during the life of the process and always move along with the process. In this sense these are special ambients around agent processes. Yet in treatment of the ambients, a uniform syntax is adopted which includes the agent ambients, though certain operations are not allowed in the agent ambients.

4 Ambient LCC syntax

The previous section dealt with the conceptual background of ambient LCC. The more formal specification of the syntax and functional properties along with notations and conventions are discussed in this section. A detailed discussion on LCC specific syntax is omitted here and can be found in [11]. The ambient LCC syntax is as follows:

$$\begin{array}{lll} Framework & ::= & \langle \Delta, Clause^n \rangle \\ Clause & ::= & \mathcal{A} :: Def \end{array}$$

$$
\begin{array}{lcl}
\mathcal{A} & ::= & a(Id_{\mathcal{A}}, Id_{\mathcal{R}}, Id_{\Delta}) \\
Def & ::= & Action \mid \mathcal{A} \mid [E \leftarrow] Def [\leftarrow C] \mid Def \text{ or } Def \mid Def \text{ then } Def \mid Def \text{ par } Def \\
\Delta & ::= & \langle Id_{\Delta}, \tau_{\Delta} \rangle \mid Id_{\Delta} \mid \Delta(\Delta^n) \\
Action & ::= & Message \mid Op_{\Delta} \mid E \mid timeout(n) \\
Message & ::= & M \Rightarrow \mathcal{A} \mid M \Leftarrow \mathcal{A} \\
Op_{\Delta} & ::= & new\ (\langle Id_{\Delta}, \tau_{\Delta} \rangle, Id_{\delta}) \mid new\ (Id_{\Delta}, Id_{\delta}) \mid in\ Id_{\delta}(\mathcal{R}) \mid out\ Id_{\delta}(\mathcal{R}) \mid open\ Id_{\delta} \\
\tau_{\Delta} & ::= & Spec_{\Delta} \\
C & ::= & get(Term, V) \mid Term \mid C \wedge C \mid C \vee C \\
E & ::= & put(V, Term) \mid C \mid P(V^n, V^n) \\
M & ::= & Term : \tau \\
V & ::= & variable[: \tau] \\
n & ::= & integer
\end{array}
$$

An ambient is represented by the symbol Δ. The agents and roles are denoted by $\mathcal{A}$ and $\mathcal{R}$ respectively. $Id_{\mathcal{A}}, Id_{\mathcal{R}}, Id_{\Delta}$ respectively represent the unique identification of an agent, role and ambient, while Id_{δ} uniquely determines the specific ambient instance (execution environment). τ represents a type, which can be either an ambient type or the type associated with a term or a variable.

In ambient LCC, a $framework$ is the combination of an ambient and a number of clauses. This view is the main extension that is brought to LCC, where there was only the notion of agents and no entity above the agent layer. Thus there are six key syntactic categories in the definition, apart from the five categories that exist in the LCC language, the additional category introduced in ambient LCC is the *Ambient*. Next, we describe the ambient syntax and the changes introduced in other categories.

- **Ambient Definition** An ambient Δ can be defined as $\langle Id_{\Delta}, \tau_{\Delta} \rangle \mid Id_{\Delta} \mid \Delta(\Delta^n)$. Δ consists of the type of the ambient τ_{Δ} and the ambient identification Id_{Δ}, or just a reference to a previously defined ambient, Id_{Δ}. There is also a provision for defining an ambient inside another ambient through the definition of $\Delta(\Delta^n)$.The following explains the details of ambient definition.

  The type of the ambient is the signature of the ambient. Intuitively, τ_{Δ} is the set of parameters that define the ambient and that specify the norms of access and conduct within an ambient. τ_{Δ} is normally derived from the interaction model specification and hence is an interaction model dependent parameter.

  Ambients can be defined recursively to include the notion of an ambient inside another ambient. This permits us to define an ambient structure such as $\langle Id_{ps}, \tau_{ps} \rangle (Id_{s_1}, Id_{s_2}, Id_{s_3})$. By referring to an ambient identifier we can easily build complex ambients. The following example illustrates how this can be achieved:

  $\langle Id_{ps}, \tau_{ps} \rangle (\langle Id_{ps_1}, \tau_{ps_1} \rangle (\langle Id_{s_a}, \tau_{s_a} \rangle), Id_{s_a}, Id_{s_b}, \langle Id_{ps_2}, \tau_{ps_2} \rangle (\langle Id_{s_b}, \tau_{s_b} \rangle, \langle Id_{s_c}, \tau_{s_c} \rangle))$

- **Agent Definition** In the light of the ambient addition, there are slight variations in the interpretation of the other categories. A $Framework$, which still bounds an interaction now comprises an ambient and a set of clauses. Each $Clause$ corresponds to an agent in an ambient. An agent is redefined in the new language as an entity having a unique identifier $\mathcal{A}$ enacting a role $\mathcal{R}$ in an ambient Δ and is represented as $a(Id_{\mathcal{A}}, Id_{\mathcal{R}}, Id_{\Delta})$. They are subsequently defined as agents enacting certain roles in specific ambient instances, or carrying out actions, upon satisfying certain conditions (preconditions C) and has certain effects (post conditions E).

The interactions that the agent must perform are given by a definition Def. These definitions may be composed as sequences (*then*), choices (*or*), or parallel execution (*par*). Additionally the definitions are optionally constrained by a precondition and affected by a post condition.

– **Action Definition** In the new definition, the fifth syntactic category is replaced by an enclosing category, namely *Action*. There are two kinds of *Actions*, $Messages$ form one category used as an interaction mechanism among agents. Another category is the *ambient operations* that an agent can perform to function effectively within the *ambient* framework. The actual interactions between agents are given by $Message$ definitions. Messages involve sending ($\Rightarrow$) or receiving ($\Leftarrow$) terms M from another agent and the process is made generic by the definition $[E \leftarrow]Def[\leftarrow C]$. In this context, C stands for the preconditions that an agent needs to satisfy before sending any message. E represents in general the effect of an action that an agent has performed. The definition also includes the possibility of combining more than one condition in a conjunctive or disjunctive manner. In summary the different kinds of message operations in Ambient LCC remain the same as that in LCC, with the only difference that they all must be carried out in the context of an ambient now.

The actual ambient operations are: creating an ambient type and instance at the same time *new* $(\langle Id_\Delta, \tau_\Delta \rangle, Id_\delta)$, creating an ambient instance of a predefined ambient type *new* (Id_Δ, Id_δ), moving into an ambient instance *in* $Id_\delta(r)$, moving out of an ambient instance *out* $Id_\delta(r)$ and opening an ambient instance *open* Id_δ. Here, Id_δ represents the unique identifier for the ambient instance (execution environment).

new $(\langle Id_\Delta, \tau_\Delta \rangle, Id_\delta)$ is two operations encapsulated into one construct. *new* first creates a new ambient and declares its type as τ_Δ. Then the id, Id_Δ is returned as the id of the newly created ambient. Using this new ambient, an instance of the same is created and finally the id of the instance Id_δ is returned. Thus in this case, both the ambient id and the instance id are returned as the result of the *new* operation. Whereas the *new* (Id_Δ, Id_δ) takes an existing ambient Id_Δ and creates an instance of this ambient, and returns the id Id_δ of the newly created instance.

The *in* and the *out* operations manage the movement of the agent across the ambients. Upon satisfying a set of preconditions, as specified in τ_Δ, *in* $Id_\delta(r)$ will let the agent executing the operation inside an ambient instance. *out* $Id_\delta(r)$ is similar. *open* Id_δ destroys the ambient instance δ and provides a logical end of an ambient execution environment.

– **Access primitives** The access primitives are there to support the actions that an agent wants to execute. There are two primitive operations provided to access variables (in general Terms), $get(Term, V)$ $put(V, Term)$. This can be used to access and update variables in the various levels of the ambient hierarchy. This is achieved through typing the primitives with the ambient identifier Id_δ. For example, $get(Term, V) : \delta_1$ will get the value of the term from the ambient instance δ_1 and will make V bound to it. The variables are generalized to a *prolog term* to include the possibility of containing complex expressions. Ambient LCC is a strongly typed language, and as far as possible typed variables are used. τ can be an ambient type, an ambient instance id, or any other previously defined type at specification time.

The $Timeout$ is used to introduce a minimal time management into the language. Interaction models normally constrain many agent actions to happen within a time interval. That is to say, agents can no longer perform a certain operation once a time interval is elapsed.

4.1 A few methodological issues

So far what we have discussed is the syntax of Ambient LCC. There are a few concepts, which are not specifically part of the language syntax, but nevertheless methodologically important as they provide additional structure and ease of understanding.

- **Ambient execution environment** An ambient Δ consists of two logical components, the ambient specification τ_Δ which we have previously dealt with, and the ambient instance, also called the ambient execution environment δ. The following describes the notion of δ and elaborates on what constitutes its structure. The ambient execution environment is a runtime environment of the ambient Δ of type τ_Δ. For any ambient Δ of type τ_Δ, there can be an infinite enumeration of its runtime counterpart. The execution environment deals with the execution variables, the current state the ambient is in and so on. It is defined in such a way that the execution variables comply with the specification requirements of τ_Δ. δ is not part of the formal syntax of the ambient LCC, and this omission is intended so as not to have to repeat the execution parameters in the protocol specification. Nevertheless, the variables have reserved names and can be accessed by these names scoped by the ambient identifier. As with τ_Δ definition, the δ definition varies with the interaction model under consideration.

  The recursive definition of ambients is also valid at the ambient instance level. That is, the recursive definition specifies the ambient hierarchy as explained previously. This is a τ_Δ specification at the highest level, and hence taken as the blueprint while creating the execution ambients. The structure of δ is created at runtime based on the *in* and *out* operations. These operations should respect the ambient hierarchy defined at the type level. Thus the following statements hold for nested ambients at instance level.

  - To move into a sub ambient of an enclosing ambient, it is required to be in the enclosing ambient. That is, given the spec $\langle Id_{\Delta_1}, \tau_{\Delta_1}\rangle(\langle Id_{\Delta_2}, \tau_{\Delta_2}\rangle)$ where Δ_1 encloses Δ_2 and the operations $new(Id_{\Delta_1}, Id_{\delta_1})$ and $new(Id_{\Delta_2}, Id_{\delta_2})$, for an agent $a \in \mathcal{A}$, with role $r \in \mathcal{R}$, *in* $Id_{\delta_2}(r)$ is valid only if $a \in \delta_1$ due to a prior operation, *in* $Id_{\delta_1}(r)$, that moved a into the enclosing ambient.

  - After *open* Id_{δ_2} is executed, a is assumed to be in the surrounding ambient, δ_1. *open* moreover destroys the ambient Id_{δ_2}.

  - Similarly, after *out* $Id_{\delta_2}(r)$ is executed, a is assumed to be in the surrounding ambient, δ_1

- **Domain Type** It is also useful to divide τ_Δ into two parts, that of τ_I and τ_D. While τ_I holds the interaction model specification variables, τ_D holds the domain variables which are specific to a domain. The separation between τ_I and τ_D is brought in to keep the domain independent specification from the domain dependent variables. τ_D is the place holder for all the variables that are part of the domain. By including τ_D in the specification, the execution environment variables are forced to adhere to the τ_D specification. This also provides the possibility of automatic verification.

5 Mapping Electronic Institutions into Ambient LCC

Ambient LCC can be used to implement electronic institutions in a distributed environment (i.e. without any central infrastructure) while preserving the semantics of scenes and performative structures. From an Electronic institution perspective, an ambient is an entity which can represent a performative structure, a scene or an agent. It has a nested structure as well, so we can have a performative structure ambient containing a collection of scene ambients, and a scene ambient containing a collection of agent ambients. The level of nesting corresponding to the nesting in the performative structure. The ambient processes can be *agent processes* that run in an agent ambient, scene ambient or a performative structure ambient, each of which are expanded below.

5.1 Structural mapping

The mapping between EIs and Ambient LCC is based on the following structural translations:

- **Performative structure ambients** An institution's top performative structure is mapped into the highest level ambient, that we call *rootambient*, which includes other performative structure ambients, scene ambients and agent ambients. The nesting is achieved by the recursive definition of the ambients. Any agent process entering an institution, first enters the root ambient, becoming an agent ambient (i.e. an ambient with a process running the decision making processes and a set of parameters to maintain the private state of the agent), and until it exits the institution resides in the root ambient (or in any ambient in the nested structure). In the process of enacting different scenes and transitions, the agent ambient enters and exits various sub ambients, at various layers of the nested structure.
- **Scene ambients** As for any ambient in Ambient LCC, a scene ambient is composed of a scene ambient specification and an execution environment. The specification of a scene in an EI is mapped into a number of parameters, the ambient type, that keep as values the different components of a scene specification (e.g. the states of the state machine, the arcs, the allowed illocutions, etc.) The execution environment parameters keep the context of the scene in execution and provides an environment for enacting the scene ambient specification. By this we mean, the scene ambient execution environment contains, for instance, the names of the participating agent processes at any given point in time, their roles, and the state of the computation at that instant of time. The participating agents are aware of the ambients they are in, and can communicate among themselves by referring to their identification within the ambient. Scene ambients are the most dynamic in the sense that most of the computation occurs within a scene ambient. The capabilities of movement defined in the language is expressive enough to model the movement of agents within an EI scene.
- **Transition Ambients** Transitions are considered as special kinds of scenes in electronic institutions. Owing to their similarity with the scenes and following similar conventions, it is fair enough to treat them similarly in ambient LCC too. Thus, transitions are mapped into ambients and derive all the properties of ambients as defined in the language. Transitions in ambient LCC facilitate the initiation of scene ambients. They also execute the agent movements into succeeding scenes. Apart from these, transitions provide the settings for a scene ambient to start the execution, or for the agents to start their protocols.

– **Agent Ambients** We map agents into agent ambients. As mentioned before, an agent ambient contains a process (for internal thinking) and the parameters that represent the internal state of the agent. We don't map the functionality of governors. A governor provides an abstraction that facilitates the interaction between external agents and an institution. They ensure that the institution norms are followed as all agent messages go through the governors. Agent ambients provide the agent specific context of computation, but do not quite ensure that agents follow the norms of the institution.

6 An Example of an EI expressed as an interaction model in *ambient LCC*

As an example of conversion of an institution entity, we choose for translation an *Auction* scene. When translated into ambient LCC, the scene corresponds to protocol fragments from the *Buyer* and *Auctioneer* role perspectives. In an upward bidding protocol, the auction scene starts when the *Auctioneer* declares the *startauction* followed by the start of a round selecting a particular *Good*, initial price and bidding time. Then the auction proceeds when the *Buyer* agents request to buy the *Good* for the *Price* announced. If there are one or more *Buyer* agents requesting the *Good* for the last called *Price* then the *Auctioneer* increases the *Price*. The process gets repeated until there is a *Timeout*. If there is no more *Buyer* agents requesting for the *Good* before the *Timeout* expires, then the *Good* is either sold to the last *Buyer* who requested the *Good* for the previous *Price*, or else the *Good* is withdrawn from the auction. The following are the protocol fragments (illocution schema) as specified in the original electronic institution.

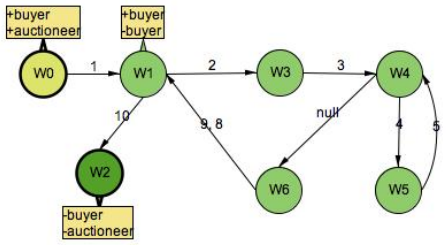

Fig. 2. Upward Bidding Auction

1. $inform((?x\ Auctioneer)(all\ Buyer)(startauction\ ?a))$
2. $inform((!x\ Auctioneer)(all\ Buyer)(startround\ ?good\ ?price\ ?bidding\_time))$

3. $inform((!x\ Auctioneer)(all\ Buyer)(offer\ !good\ !price))$
4. $request((?y\ Buyer)(!x\ Auctioneer)(bid\ !good\ !price))$
   $null:\ timeout\ [!bidding\_time]$
5. $inform((!x\ Auctioneer)(all\ Buyer)(offer\ !good\ ?price))$
8. $inform((!x\ Auctioneer)(all\ Buyer)(sold\ !good,\ !price,\ !Buyer_id))$
9. $inform((!x\ Auctioneer)(all\ Buyer)(withdrawn\ !good))$
10. $inform((!x\ Auctioneer)(all\ Buyer)(close))$

The *Auction scene* ambient is defined as follows.

$\Delta_A = \langle Id_A, \tau_A \rangle$
Where The scene specification parameters
$\tau_{I_s} = \langle R, DF_S, W, w_0, W_f, (WA_r)_{r \in R},$
$(WE_r)_{r \in R}, \Theta, \lambda, min, Max \rangle$ for the above scene A are:

$$
\begin{array}{ll}
R & = \{Auctioneer, Buyer\} \\
DF_A & = \langle \{(startauction\ ?a), (startround\ ?good\ ?price\ ?bidding\_time), \\
 & \quad (offer\ !good\ !price), (bid\ !good\ !price), (sold\ !good,\ !price,\ !Buyer_id), \\
 & \quad (withdrawn\ !good), (close)\}, \_, \_ \rangle \\
W & = \{w0, w1, w3, w4, w5, w6, w2\} \\
w_0 & = w0 \\
W_f & = \{w2\} \\
(WA_r)_{r \in R} & = \{\{w0, w1\}_{Buyer}, \{w0\}_{Auctioneer}) \\
(WE_r)_{r \in R} & = \{\{w2\}_{Buyer}, \{w2\}_{Auctioneer}) \\
\Theta & = \{(w0, w1), (w1, w2), (w1, w3), (w3, w4), (w4, w5), (w4, w6), (w6, w1)\} \\
\lambda & = \{(w0, w1) \rightarrow 1, (w1, w2) \rightarrow 10, (w1, w3) \rightarrow 2, (w3, w4) \rightarrow 3, (w4, w5), \\
 & \quad (w4, w6), (w6, w1)\} \\
 & \quad (w3, w5) \rightarrow msg5\} \\
min & = \{Auctioneer \rightarrow 1, Buyer \rightarrow 1\} \\
Max & = \{Auctioneer \rightarrow 1, Buyer \rightarrow 15\}
\end{array}
$$

Here τ_A specifies the rules that need to be kept while creating and executing instances of *Auction* scene ambient. It specifies that the only roles allowed to play the *Auction* scene are the *Buyer* and the *Auctioneer* roles. It also specifies the *start* and *end* states of the scene as $w0$ and $w2$. Then it goes on further to specify that $w0$ is the only state through which both the *Auctioneer* and *Buyer* can enter the scene ambient and $w2$ is the only state through which both the *Auctioneer* and *Buyer* can exit the ambient. It also specifies that the minimum and maximum number of agents that can be present at the ambient at any instance of time is 0 and 100 for the *Buyer* role and 1 in both cases for the *Auctioneer* role. It then specifies how the state change from one state to the next can happen through a number of parameters including λ which stands for the illocutions mentioned above.

The entry into the scene is done from outside the scene, and handled by the preceding transaction ambients. After a successful entry into the scene ambient is done, the scene protocols from the different role perspectives namely the *Auctioneer* and the *Buyer* expanded below are enacted. There are a few calls to external functions such as *getSaleinfo(Buyer_id, Quantity)* and *getGoodsinfo(G, P,B)* as part of the protocol. We don't specify them here and their meaning should be clear from their names.

$a(X, Auctioneer, Auction)$ ::

$put(\text{“}w\text{”}, \text{“}w1\text{”}) \wedge put(\text{“}GList\text{”}, G_r) \wedge put(\text{“}PList\text{”}, P_r) \wedge put(\text{“}BList\text{”}, B_r) \leftarrow$
$startauction(Id_{auction}) \Rightarrow a(\_, Buyer, Auction))$
$\leftarrow get(\text{“}w\text{”}, \text{“}w0\text{”}) \wedge getGoodsinfo(G, P, B)$

or

$put(\text{“}w\text{”}, \text{“}w2\text{”}) \leftarrow close(Id_{auction}) \Rightarrow a(\_, Buyer, Auction))$
$\leftarrow get(\text{“}w\text{”}, \text{“}w1\text{”}) \wedge GList = []$

or

$put(\text{“}w\text{”}, \text{“}w3\text{”}) \leftarrow$
$startround(Good, Price, Bidding\_time) \Rightarrow a(\_, Buyer, Auction))$
$\leftarrow get(\text{“}w\text{”}, \text{“}w1\text{”}) \wedge$
$GList = [Good|G_r] \wedge PList = [Price|P_r] \wedge BList = [Bidding|B_r]$

or

$put(\text{“}w\text{”}, \text{“}w4\text{”}) \wedge offer(Good, Price) \Rightarrow a(\_, Buyer, Auction)) \leftarrow get(\text{“}w\text{”}, \text{“}w3\text{”})$

or

$put(\text{“}w\text{”}, \text{“}w6\text{”}) \leftarrow get(\text{“}w\text{”}, \text{“}w4\text{”}) \wedge timeout[Bidding\_time]$

or

$bid(Good, Price) \Leftarrow a(Buyer(\_, Buyer, Auction)) \leftarrow get(\text{“}w\text{”}, \text{“}w5\text{”})$

or

$put(\text{“}w\text{”}, \text{“}w4\text{”}) \leftarrow offer(Good, Price) \Rightarrow a(\_, Buyer, Auction))$

or

$put(\text{“}w\text{”}, \text{“}w1\text{”}) \wedge put(\text{“}Buyer\text{”}, Buyer\_id) \wedge put(\text{“}SaleQty\text{”}, Quantity) \leftarrow$
$sold(Good, Quantity, Price, Buyer\_id) \Rightarrow a(\_, Buyer, Auction))$
$\leftarrow get(\text{“}w\text{”}, \text{“}w6\text{”} \wedge Sold\_constarints \wedge getSaleinfo(Buyer\_id, Quantity)$

or

$put(\text{“}w\text{”}, \text{“}w1\text{”}) \leftarrow withdrawn(Good) \Rightarrow a(\_, Buyer, Auction))$
$\leftarrow Withdraw\_cnstr$

or

$a(X, Auctioneer, Auction) \leftarrow get(\text{“}w\text{”}, \text{“}w2\text{”}) \wedge G_r! = []$

or

$open\ Id_{auction} \leftarrow get(\text{“}w\text{”}, \text{“}w2\text{”}) \wedge G_r = []$

$a(B, Buyer, Auction))$ ::

$put(\text{“}w\text{”}, \text{“}w5\text{”}) \leftarrow bid(Good, Price) \Rightarrow a(X, Auctioneer, Auction) \leftarrow get(\text{“}w\text{”}, \text{“}w4\text{”}) \wedge$
$offer(Good, Price) \Leftarrow a(X, Auctioneer, Auction)$

or

$sold(Sale) \Leftarrow a(X, Auctioneer, Auction) \leftarrow get(\text{“}w\text{”}, \text{“}w6\text{”})$

7 Discussion

This paper introduces a new language, (its concepts, syntax and mappings) based on process algebra concepts and specially adapted to the implementation of electronic institution models. The combination of a peer to peer interaction language (LCC) and *Ambient* concepts gives us specific avantages in addressing the challenges of developing open MAS. LCC by its design cater to distributed MAS and Ambients give the notion of bounded places and a rich structure to express even complex organization models such as Electronic Institutions. We also have developed a translation algorithm (not detailed here) which translates an EI scene into its *Ambient LCC* counter part. Further extensions needs to be developed to map the entire EI institution into the new language. We

also have developed algorithms to take care of synchronization issues at the EI scene level. This too needs to be extended at the institution level. Though this work concentrates on the EI organization model, we believe that the language is general enough to map other organization models for MAS [3], specially those that are distributed.

References

1. L. Cardelli and A. D. Gordon. Mobile Ambients. In Maurice Nivat, editor, *Foundations of Software Science and Computational Structures*, number 1378 in Lecture Notes in Computer Science, pages 140–155. Springer-Verlag, 1998.
2. Guifré Cuní, Marc Esteva, Pere Garcia, Eloi Puertas, Carles Sierra, and Teresa Solchaga. Masfit: Multi-agent systems for fish trading. In *16th European Conference on Artificial Intelligence (ECAI 2004)*, pages 710–714, Valencia, Spain, August 2004.
3. V. Dignum. *A Model for Organizational Interaction*. PhD thesis, Dutch Research School for Information and Knowledge Systems, 2004. ISBN 90-393-3568-0.
4. Marc Esteva. *Electronic Institutions: from specification to development*. IIIA PhD Monography. Vol. 19, 2003.
5. Marc Esteva, Juan A. Rodríguez-Aguilar, Carles Sierra, Josep L. Arcos, and Pere Garcia. On the formal specification of electronic institutions. In Carles Sierra and Frank Dignum, editors, *Agent-mediated Electronic Commerce: The European AgentLink Perspective*, number 1991 in Lecture Notes in Artificial Intelligence, pages 126–147. Springer-Verlag, 2001.
6. D. Harel. Dynamic logic. In D. M. Gabbay and F. Gunthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. D. Reidel, 1984.
7. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes (Part 1/2). *Information and Computation*, 100(1):1–77, September 1992.
8. Pablo Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor*. IIIA Phd Monography. Vol. 8, 1997.
9. Juan A. Rodríguez-Aguilar. *On the Design and Construction of Agent-mediated Electronic Institutions*. IIIA Phd Monography. Vol. 14, 2001.
10. Juan A. Rodríguez-Aguilar, Pablo Noriega, Carles Sierra, and Julian Padget. Fm96.5 a java-based electronic auction house. In *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology(PAAM'97)*, pages 207–224, 1997.
11. C. Walton and D. Robertson. Flexible Multi-Agent Protocols. In *Proceedings of UKMAS 2002. Also published as Informatics Technical Report EDI-INF-RR-0164, University of Edinburgh*, November 2002.
12. Petia Wohed, Wil M. P. van der Aalst, Marlom Dumas, and Arthur H. M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *Proceedings 22nd International Conference on Conceptual Modelling (ER)*, pages 200–215, 2003.