Electronic institutions
Towards Organisational-oriented Programming
Electronic Institutions Development Environment

IIIA-CSIC
Artificial Intelligence Research Institute
http://www.iiia.csic.es
eMarkets group
UTS
Overview

I. Introduction
II. Specifying electronic institutions
III. Running electronic institutions
IV. Are Electronic Institutions Enough?
V. Electronic Institutions Development Environment
VI. 3D Electronic Institutions
VII. Conclusions
I. Introduction
II. Specifying electronic institutions
III. Running electronic institutions
IV. Are Electronic Institutions Enough?
V. Electronic Institutions Development Environment
VI. 3D Electronic Institutions
VII. Conclusions
There are situations where individuals interact in ways that involve

*Commitment, delegation, repetition, liability and risk.*

These situations involve participants that are:

*Autonomous, heterogeneous, independent, non-benevolent, unreliable, liable.*
These situations are not uncommon: Markets, medical services, armies and many more.

It is usual to resort to trusted third parties whose aim is to make those interactions effective by establishing and enforcing conventions that standardize interactions, allocate risks, establish safeguards and guarantee that certain intended actions actually take place and unwanted situations are prevented.

These functions have been the basis for the development of many traditional human institutions.

They are even more necessary when interaction may involve software agents.
Open multi-agent systems are populated by heterogeneous, self-interested agents, developed by different people, using different languages and architectures. Participants change over time and are unknown in advance.

With the expansion of the Internet open multi agent systems represent the most important area of application of multi agent systems.
Research issue: methodologies and software tools to support their design, verification, development, and analysis.

Goal: principled design and development of open multi agent systems.
Institutions have proved to successfully regulate human societies for a long time:
- created to achieve particular goals while complying norms.
- responsible for defining the rules of the game (norms), to enforce them and assess penalties in case of violation.

Examples: auction houses, parliaments, stock exchange markets,....
Institutions in the sense proposed by North “… set of artificial constraints that articulate agent interactions”.
Electronic institutions development can be divided into two basic steps:

- **Formal specification** of institutional rules.
- **Execution** via an infrastructure that mediates agents’ interactions while **enforcing** the institutional rules.

The **formal specification** focuses on **macro-level** (rules) aspects of agents, not in their micro-level (players) aspects.

The **infrastructure** is required to be of **general purpose** (can interpret any formal specification).
I. Introduction

II. Specifying electronic institutions

III. Running electronic institutions

IV. Are Electronic Institutions Enough?

V. Electronic Institutions Development Environment

VI. 3D Electronic Institutions

VII. Conclusions
Textual specification language for electronic institutions.

ISLANDER editor: specification and verification tool for electronic institutions.

The editor combines textual and graphical specifications.
Electronic Institution Specification with ISLANDER

- Network of protocols
- Multi-agent protocols
- Norms
- Agent Roles
- Common Ontology and language
Electronic Institution Components

PERFORMATIVE STRUCTURE
(NETWORK OF PROTOCOLS)

SCENE
(MULTI-AGENT PROTOCOL)

AGENT ROLES

NORMS

Buyers’ Payment

done(inform(?x : a, all : b, sold(?good.id, ?buyer.id, ?price)), auction_room) ⇒

obliged(!x, pay(!buyer.id : b, ly : bac, sale(!good.id, !buyer.id, !price)), buyer_settlements)
The (“Hello World”) Chat Example

- A simple institution where agents interact simulating a chat.
- Each agent owns a main topic and a list of subtopics he is interested in.
- Agents create a chat room per main topic.
- They can join the scenes created by other agents.
- The institution keeps track of active chat rooms to provide information to agents.
Dialogic Framework Components

- Common ontology

- Valid communication language expressions
  - List of illocutionary particles
  - Content language

- Roles that agents can play
  - Internal Roles
  - External Roles

- Role relationships
Each **role** defines a **pattern** of behaviour within the institution (actions associated to roles).

**Agents** can play **multiple** roles at the same time.

**Agents** can **change** their roles.

**Two types of roles:**

- **Internal**: played by the **staff** agents to which the institution delegates its services and tasks.
- **External**: played by external agents.

**Role relationships:**

- Static incompatibility (ssd)
- Dynamic incompatibility (dsd)
- Hierarchy (sub)
Chat roles
Communication Language

- CL expressions are formulae of the form 
  \((i \ (\alpha_i r_i) \ \beta \ \gamma \ \tau)\) where:
  - \(i\) is an illocutionary particle (e.g. request, inform);
  - \(\alpha_i\) can be either an agent variable or an agent identifier;
  - \(r_i\) can be either a role variable or a role identifier;
  - \(\beta\) represents the addressee(s) of the message and can be:
    - \((\alpha_k r_k)\) the message is addressed to a single agent.
    - \(r_k\) the message is addressed to all the agents playing role \(r_k\).
    - “all” the message is addressed to all the agents in the scene.
  - \(\gamma\) is an expression in the content language.
  - \(\tau\) can be either a time variable or a time-stamp

\((\text{request (?x guest) (!y staff) (login ?user ?email)})\)
Communication Language

Roles

(request (?x guest) (!y staff) (login ?user ?email))
- **Specification level**
  - A scene is a pattern of multi-agent interaction.
  - Scene protocol specified by a finite state oriented graph where the nodes represent the different states and oriented arcs are labelled with *illocution schemes* or *timeouts*.

- **Execution level**
  - Agents may join or leave scenes.
  - Each scene keeps the *context* of its multi-agent interaction.
  - A scene can be multiply executed and played by different groups of agents.
### Scene Type: guest_admission_scene

<table>
<thead>
<tr>
<th>Specification</th>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name:</strong></td>
<td></td>
<td>guest_admission_scene</td>
</tr>
<tr>
<td><strong>Role</strong></td>
<td><strong>Participates</strong></td>
<td><strong>Min</strong></td>
</tr>
<tr>
<td>guest</td>
<td>✓</td>
<td>0</td>
</tr>
<tr>
<td>staff</td>
<td>✓</td>
<td>1</td>
</tr>
<tr>
<td>roomManager</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**Dialogic framework:**
- `helloWorld_df`

**Initial state:**
- W0

**Final states:**
- W3
Guest admission scene. State information
1. (request (?x guest) (?y staff) login(?user ?email))
2. (inform (!y staff) (!x guest) accept())
3. (failure (!y staff) (!x guest) deny(?code))
4. (request (?x guest) (!y staff) login(?user ?email))
5. (inform (!y staff) (all guest) close())
6. (inform (?y staff) (all guest) close())
Illocution schemes: at least the terms referring to agents and time are variables.

Semantics of variable occurrences:
- ?x: variable x can be bound to a new value.
- !x: variable x must be substituted by its last bound value.
  Example:
    (request (?x guest) (!y staff) login(?user ?email))

Context of a conversation captured on a list of variables’ bindings.
Guest admission scene. Illocutions

Two agents in the scene: John as guest and Mike as staff.

Agent John utters an illocution:

(request (John guest) (Mike Staff) login(John john@hotmail.com) )

The illocution matches arc 1 and the scene evolves to W1.

Substitutions:

[?x/John, ?y/Mike, ?user/John, ?email/john@hotmail.com]
Guest admission scene. Illocutions

2. (inform (!y staff) (!x guest) accept() )
3. (failure (!y staff) (!x guest) deny(?code) )

- Former bindings:
  [?x/John, ?y/Mike, ?user/John, ?email/john@hotmail.com]

- Only illocutions matching the following schemes will be valid:
  (inform (Mike staff) (John guest) accept() )

  (failure (Mike staff) (John guest) deny(?code) )
Constraints capture how past actions in a scene affect its future evolution:
  • restricting the valid values for a variable
  • restricting the paths that a conversation can follow

Examples:
  • A buyer can only submit a single bid at auction time.
  • A buyer must submit a bid greater than the last one.
  • An auctioneer can not declare a winner if two buyers have submitted a bid at the higher value.
  • An agent can not repeat an offer during a negotiation process.
Example

- Illocution scheme:
  \[
  \text{commit((?y \ buyer) (!x \ auctioneer) \ bid(!good_id, \ ?price))}
  \]

- Constraint:
  \[
  (\ > \ ?price \ !starting\_price) \\
  ?price \in (!starting\_price.\infty)
  \]

\[
\begin{array}{ccc}
\text{?price} & \in (0.\infty) \\
0 & \text{!starting\_price} & +\infty
\end{array}
\]
- \(?x\): binding occurrence
- \(!x\): stands for the last binding of variable \(x\).
- \(!x (w_i w_j)\): stands for the bindings of variable \(x\) the last time that the conversation evolve from \(w_i w_j\).
- \(!x (w_i w_j i)\): stands for the bindings of variable \(x\) the last \(i\) times that the conversation evolved from \(w_i w_j\).
- \(!x (w_i w_j *)\): stands for the bindings of variable \(x\) all the times that the conversation evolved from \(w_i w_j\).
Example: Vickrey 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 ?bid_price) )
5 ![bidding_time] )
6 (inform (!x auctioneer) (all buyer) sold(!good ?sold_price ?buyer_id) )
8 (inform (!x auctioneer) (all buyer) close() )
7 (inform (!x auctioneer) (all buyer) withdrawn(!good) )
Constraints

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 ?bid_price) )
5. ![bidding_time]
6. (inform (!x auctioneer) (all buyer)
   sold(!good ?sold_price ?buyer_id) )
7. (inform (!x auctioneer) (all buyer) close() )
8. (inform (!x auctioneer) (all buyer) withdrawn(!good) )

Constraint:

?y \notin !y (w3 w4)
?bid_price > !price
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 ?bid_price) )
5 ![bidding_time]
6 (inform (!x auctioneer) (all buyer)
  sold(!good ?sold_price ?buyer_id) )
7 (inform (!x auctioneer) (all buyer) withdrawn(!good) )

Constraint:

| ![w3 w4]| = 0
Complex activities can be specified by establishing relationships among scenes that define:

- **causal dependency** (e.g. a guest agent must go through the admission scene before going to the chat rooms)
- **synchronisation points** (e.g. synchronise a buyer and a seller before starting a negotiation scene)
- **parallelisation mechanisms** (e.g. a guest agent can go to multiple chat rooms)
- **choice points** (e.g. a buyer leaving an admission scene can choose which auction scene to join)
- **the role flow policy**
Performative structures as networks of scenes.

Transitions to link scenes.

Arcs connecting scenes and transitions labelled with constraints and roles.

Agents moving from a transition to a scene may join *one*, *some* or *all* current executions of the target scene(s) or start new executions.

The specification allows to express that simultaneous executions of a scene may occur.
Chat Performative Structure

Scene
Chat Performative Structure

And transition: synchronisation and parallelisation point
And transition: synchronisation and parallelisation point
Or transition: choice point
Or transition: choice point
Chat Performative Structure

**XOr transition:** exclusive choice point
XOr transition: exclusive choice point
Chat Performative Structure
Chat Performative Structure
Arcs connecting transitions to scenes determine whether agents join *one*, *some* or *all* current executions of the target scene(s) or whether *new* executions are started.
Norms define the consequences of agents actions within the institution.

Such consequences are captured as obligations.
  - Obl(x, Φ, s): meaning that agent x is obliged to do Φ in scene s.

Norms are a special types of rules specified by three elements:
  - Antecedent: the actions that provoke the activation of the norm and boolean expressions over illocution scheme variables.
  - Defeasible antecedent: the actions that agents must carry out in order to fulfil the obligations.
  - Consequent: the set of obligations

Actions expressed as pairs of scene and illocution schema.
**Inspect Dialog**

<table>
<thead>
<tr>
<th>MAS Data</th>
<th>Graphical Data</th>
</tr>
</thead>
</table>

**Name:**
obligation2pay

**Antecedent definition:**

```prolog
(((auction-room (inform (?y auctioneer) (?x buyer) (sold ?good-id ?price ?x))))
```

**Defeasible Antecedent definition:**

```prolog
((buyer-settlement (inform (?x buyer) (?y buyer-accountant) (payment !price)))
```

**Consequent definition:**

```prolog
((obl ?x (inform (?x buyer) (?y buyer-accountant) (payment !price)) buyer-settlement))
```
Electronic Institutions Definition
Electronic Institution Specification with ISLANDER

Best prototype paper award at AAMAS (2002)
Overview

I. Introduction
II. Specifying electronic institutions
III. Running electronic institutions
IV. Electronic Institutions Development Environment
V. 3D Electronic Institutions
VI. Conclusions
Electronic institution execution

ENVIRONMENT

AGENT₁
AGENT₂
AGENT₃

NORMS

ELECTRONIC INSTITUTION

AGENT₁
AGENT₂
AGENT₃

EXECUTION STATE: Ω
Electronic institution execution

EXECUTION STATE: \( \Omega \)
Electronic institution execution

EXECUTION STATE: $\Omega \rightarrow \Omega'$
Electronic institutions are populated at run-time by heterogenous, self-interested agents.

The institution execution can be regarded as the execution of its different scenes (processes).

Agents move from scene to scene.

Agents interact within scenes via speech acts.

Agents acquire and fulfil obligations.
Electronic institution execution monitoring

**SCENES**

- **name**: E Institution
- **root**: root_output_scene
- **output**: root_output_scene
- **meetingRoom**: meeting_scene
- **tradeRoom**: double_auction_scene

**TRANSITIONS**

- **SupplyChainTrader**: tradeMgr
- **toMeetingRoom**: OR-OR
- **toTradeRoom**: OR-OR

**Description**

- **Type**: convId: 13 convName: tradeRoom convType: double_auction_scene
- **Description**: fromConvId: 12 toConvId: 13 agentName: SupplyChainTrader agent...
Electronic institution execution monitoring

Several executions of a tradeRoom scene

Events in a tradeRoom scene
Electronic institution execution monitoring

SimpleMonTool (MONITORING MODE)

File Monitoring View Help

Scene Agent

- name: E.Institution
  - root: root_output_scene
  - output: root_output_scene
  - meetingRoom: meeting_scene
  - tradeRoom: double_auction_scene
    - id 13: Finished
      - SupplyChainTrader:tradeMgr
        - BIGWire:buyer
        - BIGNail:buyer
    - id 29: state W1
      - SupplyChainTrader:tradeMgr
        - BIGWire:buyer
        - BIGClip:buyer
    - id 38: state initial
      - toMeetingRoom: OR-OR
      - toTradeRoom: OR-OR
      - toOutputFromTradeRoom: OR-OR

Type Description

- convId: 29 convName: tradeRoom convType: double_auction_scene
- fromConvId: 28 toConvId: 29 agentName: SupplyChainTrader agentRole: tradeMgr
- fromConvId: 33 toConvId: 29 agentName: BIGWire agentRole: buyer
- fromConvId: 35 toConvId: 29 agentName: BIGClip agentRole: buyer
- convId: 29 message: (inform (BIGWire:buyer) (SupplyChainTrader:tradeMgr) (demand 100 100))
- convId: 29 message: (inform (BIGClip:buyer) (SupplyChainTrader:tradeMgr) (demand 300 20.0))
- convId: 29 message: (inform (BIGWire:buyer) (SupplyChainTrader:tradeMgr) (demand 1071073955741))
Execution State

- $\Omega = \langle Ag, \Sigma, T, Obl \rangle$ stands for an institution execution state where:
  - $Ag = \{ag_1, \ldots, ag_n\}$ is a finite set of participating agents.
  - $\Sigma = \{\sigma_i^k | s_i \in S, k \in N\}$ is the set of all scene executions.
  - $T = \{T_1, \ldots, T_n\}$ stands for all transitions executions.
  - $Obl \{obl_1, \ldots, obl_n\}$ is the set of agents' pending obligations.

- $\sigma_i^k = \{\omega, B, A\}$ stands for scene execution state where:
  - $\omega$ represents the scene's current state.
  - $B = \{\beta_1, \ldots, \beta_n\}$ stands for the context (bindings) produced by illocutions.
  - $A = \{(ag, r) | ag \in Ag, r \in R\}$ is the set participating agents along with their roles.

- Each transition execution state $T_i = \{ (ag, \delta) | ag \in Ag, \delta = \{ (\sigma_i^k, r) | \sigma_i^k \in \Sigma, r \in R\} \}$ contains agents' target scenes.
### Infrastructure operations

<table>
<thead>
<tr>
<th>Specification</th>
<th>Functionality</th>
</tr>
</thead>
</table>
| Institution   | `enter(ag, Roles)`  
                `exit(ag)`                |
| Performative structure | `create_scene(s)`  
                        `close_scene(σ)`          |
| Scene         | `join(σ, SAgents)`  
                `update_state(σ, ν)`  
                `update_state(σ, τ)`  
                `leave(σ, SAgents)` |
| Transition    | `add_agents(t, TAgents)`  
                `move_to(t, ag, Target)`  
                `fire(t)`  
                `remove_agents(t, TAgents)` |
| Norm          | `add_obligations(Obligations)`  
                `remove_obligations(Obligations)` |

Add and remove agents
## Infrastructure operations

<table>
<thead>
<tr>
<th>Specification</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institution</td>
<td>( \text{enter}(ag, \text{Roles}) ) &lt;br&gt; ( \text{exit}(ag) )</td>
</tr>
<tr>
<td>Performative structure</td>
<td>( \text{create}<em>\text{scene}(s) ) &lt;br&gt; ( \text{close}</em>\text{scene}(\sigma) )</td>
</tr>
<tr>
<td>Scene</td>
<td>( \text{join}(\sigma, S\text{Agents}) ) &lt;br&gt; ( \text{update}<em>\text{state}(\sigma, \iota) ) &lt;br&gt; ( \text{update}</em>\text{state}(\sigma, \tau) ) &lt;br&gt; ( \text{leave}(\sigma, S\text{Agents}) )</td>
</tr>
<tr>
<td>Transition</td>
<td>( \text{add}<em>\text{agents}(t, T\text{Agents}) ) &lt;br&gt; ( \text{move}</em>\text{to}(t, ag, \text{Target}) ) &lt;br&gt; ( \text{fire}(t) ) &lt;br&gt; ( \text{remove}_\text{agents}(t, T\text{Agents}) )</td>
</tr>
<tr>
<td>Norm</td>
<td>( \text{add}<em>\text{obligations}(\text{Obligations}) ) &lt;br&gt; ( \text{remove}</em>\text{obligations}(\text{Obligations}) )</td>
</tr>
</tbody>
</table>
## Infrastructure operations

<table>
<thead>
<tr>
<th>Specification</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institution</td>
<td>( \text{enter}(\text{ag, Roles}) )</td>
</tr>
<tr>
<td></td>
<td>( \text{exit}(\text{ag}) )</td>
</tr>
<tr>
<td>Performative structure</td>
<td>( \text{create_scene}(s) )</td>
</tr>
<tr>
<td></td>
<td>( \text{close_scene}(\sigma) )</td>
</tr>
<tr>
<td>Scene</td>
<td>( \text{join}(\sigma, SAgents) )</td>
</tr>
<tr>
<td></td>
<td>( \text{update_state}(\sigma, \nu) )</td>
</tr>
<tr>
<td></td>
<td>( \text{update_state}(\sigma, \tau) )</td>
</tr>
<tr>
<td></td>
<td>( \text{leave}(\sigma, SAgents) )</td>
</tr>
<tr>
<td>Transition</td>
<td>( \text{add_agents}(t, TAgents) )</td>
</tr>
<tr>
<td></td>
<td>( \text{move_to}(t, \text{ag, Target}) )</td>
</tr>
<tr>
<td></td>
<td>( \text{fire}(t) )</td>
</tr>
<tr>
<td></td>
<td>( \text{remove_agents}(t, TAgents) )</td>
</tr>
<tr>
<td>Norm</td>
<td>( \text{add_obligations}(\text{Obligations}) )</td>
</tr>
<tr>
<td></td>
<td>( \text{remove_obligations}(\text{Obligations}) )</td>
</tr>
</tbody>
</table>
## Infrastructure operations

<table>
<thead>
<tr>
<th>Specification</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institution</td>
<td>$\text{enter}(ag, \text{Roles})$</td>
</tr>
<tr>
<td></td>
<td>$\text{exit}(ag)$</td>
</tr>
<tr>
<td>Performative</td>
<td>$\text{create}_{-}\text{scene}(s)$</td>
</tr>
<tr>
<td>structure</td>
<td>$\text{close}_{-}\text{scene}(\sigma)$</td>
</tr>
<tr>
<td>Scene</td>
<td>$\text{join}(\sigma, S\text{Agents})$</td>
</tr>
<tr>
<td></td>
<td>$\text{update}_{-}\text{state}(\sigma, \nu)$</td>
</tr>
<tr>
<td></td>
<td>$\text{update}_{-}\text{state}(\sigma, \tau)$</td>
</tr>
<tr>
<td></td>
<td>$\text{leave}(\sigma, S\text{Agents})$</td>
</tr>
<tr>
<td>Transition</td>
<td>$\text{add}_{-}\text{agents}(t, T\text{Agents})$</td>
</tr>
<tr>
<td></td>
<td>$\text{move}_{-}\text{to}(t, ag, \text{Target})$</td>
</tr>
<tr>
<td></td>
<td>$\text{fire}(t)$</td>
</tr>
<tr>
<td></td>
<td>$\text{remove}_{-}\text{agents}(t, T\text{Agents})$</td>
</tr>
<tr>
<td>Norm</td>
<td>$\text{add}_{-}\text{obligations}(\text{Obligations})$</td>
</tr>
<tr>
<td></td>
<td>$\text{remove}_{-}\text{obligations}(\text{Obligations})$</td>
</tr>
</tbody>
</table>

**Add and remove agents’ requests & Fire transitions**
## Infrastructure operations

<table>
<thead>
<tr>
<th>Specification</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institution</td>
<td>( \text{enter}(ag, \text{Roles}) ) &lt;br&gt; ( \text{exit}(ag) )</td>
</tr>
<tr>
<td>Performative structure</td>
<td>( \text{create_scene}(s) ) &lt;br&gt; ( \text{close_scene}(\sigma) )</td>
</tr>
<tr>
<td>Scene</td>
<td>( \text{join}(\sigma, \text{SAgents}) ) &lt;br&gt; ( \text{update_state}(\sigma, \iota) ) &lt;br&gt; ( \text{update_state}(\sigma, \tau) ) &lt;br&gt; ( \text{leave}(\sigma, \text{SAgents}) )</td>
</tr>
<tr>
<td>Transition</td>
<td>( \text{add_agents}(t, \text{TAgents}) ) &lt;br&gt; ( \text{move_to}(t, \text{ag}, \text{Target}) ) &lt;br&gt; ( \text{fire}(t) ) &lt;br&gt; ( \text{remove_agents}(t, \text{TAgents}) )</td>
</tr>
<tr>
<td>Norm</td>
<td>( \text{add_obligations}(\text{Obligations}) ) &lt;br&gt; ( \text{remove_obligations}(\text{Obligations}) )</td>
</tr>
</tbody>
</table>

Add and remove agents’ obligations
Electronic Institution Infrastructure

TRADITIONAL APPROACH

Autonomous Agents Layer

Agent 1  …  Agent n

Communication Layer

JADE

Electronic Institution (Islander Spec)

INSTITUTIONAL APPROACH

Autonomous Agents Layer

Agent 1  …  Agent n

AMELI

Communication Layer

JADE
**AMELI functionalities**

- **MEDIATION**
  - To facilitate agent communication within scenes.

- **COORDINATION AND ENFORCEMENT**
  - To guarantee the correct evolution of each scene.
  - To guarantee legal movements between scenes.
  - To control the obligations participating agents acquire and fulfil.

- **INFORMATION MANAGEMENT**
  - To facilitate the information agents need in the institution.
AMELI implementation

Current implementation composed of four types of agents:

- An institution manager (IM) that starts the institution, authorises agents to enter, and controls the creation of scenes.

- Scene managers (SM) responsible for governing scenes.

- Transition managers (TM) control agents’ movements between scenes.

- Governors (G) devoted to mediate the interaction of an agent with the rest of agents within the institution and to control agents’ obligations.
- Mediates between institution and participating agent.
- Controls that an agent behaves according to the institution specification (rules).
<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enterInstitution</td>
<td>Request to enter the institution</td>
</tr>
<tr>
<td>moveToTransition</td>
<td>Request to move from a scene to a transition</td>
</tr>
<tr>
<td>moveToScenes</td>
<td>Request to move from a transition to several scenes</td>
</tr>
<tr>
<td>saySceneMessage</td>
<td>Request to say a message in a scene</td>
</tr>
<tr>
<td>accesScenes</td>
<td>Ask for the scenes the agent can join from a transition</td>
</tr>
<tr>
<td>accesTransitions</td>
<td>Ask for the transitions the agent can join from a scene</td>
</tr>
<tr>
<td>agentObligations</td>
<td>Ask for pending obligations</td>
</tr>
<tr>
<td>sceneState</td>
<td>Ask for a scene’s current state</td>
</tr>
<tr>
<td>scenePlayers</td>
<td>Ask for agents in a scene</td>
</tr>
</tbody>
</table>
## Governor to Agent Messages

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enteredInstitution</td>
<td>The agent has entered the institution</td>
</tr>
<tr>
<td>exitedInstitution</td>
<td>The agent leaves the institution</td>
</tr>
<tr>
<td>enteredInstitutionFailed</td>
<td>The agent could not enter the institution</td>
</tr>
<tr>
<td>saidSceneMessage</td>
<td>An agent message has been said within a scene</td>
</tr>
<tr>
<td>saySceneMessageFailed</td>
<td>Agent message in a scene has failed</td>
</tr>
<tr>
<td>receivedSceneMessage</td>
<td>Reception of a message for the agent within a scene</td>
</tr>
<tr>
<td>timeoutTransition</td>
<td>The scene state has evolved as a consequence of the expiration of a timeout</td>
</tr>
<tr>
<td>enteredAgent</td>
<td>An agent has entered the scene</td>
</tr>
<tr>
<td>exitedAgent</td>
<td>An agent has left the scene</td>
</tr>
<tr>
<td>finishedScene</td>
<td>The scene has finished</td>
</tr>
<tr>
<td>currentAccessScenes</td>
<td>List of all the scenes that the agent can move into from a transition</td>
</tr>
<tr>
<td>movedToScene</td>
<td>The agent has entered a scene</td>
</tr>
<tr>
<td>moveToSceneFailed</td>
<td>Agent attempt to move to a scene failed</td>
</tr>
<tr>
<td>currentAccessTransitions</td>
<td>Informs of all the transitions that the agent can move into from a scene</td>
</tr>
<tr>
<td>moveToTransition</td>
<td>The agent has been moved to a transition</td>
</tr>
<tr>
<td>moveToTransitionFailed</td>
<td>Agent attempt to move to a transition failed</td>
</tr>
<tr>
<td>acquiredObligations</td>
<td>Informs of acquired obligations by the agent</td>
</tr>
<tr>
<td>obligationsFulfilled</td>
<td>Informs of fulfilled obligations by the agent</td>
</tr>
<tr>
<td>currentObligations</td>
<td>Informs about the current obligations of the agent</td>
</tr>
<tr>
<td>currentSceneState</td>
<td>Informs about the scene state</td>
</tr>
<tr>
<td>currentScenePlayers</td>
<td>Informs about the agents within a scene</td>
</tr>
</tbody>
</table>
Agents make a scene execution evolve by uttering illocutions which are correct with respect to the scene specification and its run-time "context".

A scene manager updates a scene execution state after:
- Validating an agent’s illocution
- Authorising agents to join or leave
- Time-out expirations

The scene manager and participating agents’ governors coordinate to guarantee the correct evolution of the scene execution.
Protocol $P$: scene protocol played by the participating agents and mediated by governors.
**Scene Execution**

**Protocol P’**: protocol between participating agents and governors.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enterInstitution</td>
<td>Request to enter the institution</td>
</tr>
<tr>
<td>moveToTransition</td>
<td>Request to move from a scene to a transition</td>
</tr>
<tr>
<td>moveToScenes</td>
<td>Request to move from a transition to several scenes</td>
</tr>
<tr>
<td>saySceneMessage</td>
<td>Request to say a message in a scene</td>
</tr>
<tr>
<td>accessScenes</td>
<td>Ask for the scenes the agent can join from a transition</td>
</tr>
<tr>
<td>accessTransitions</td>
<td>Ask for the transitions the agent can join from a scene</td>
</tr>
<tr>
<td>agentObligations</td>
<td>Ask for pending obligations</td>
</tr>
<tr>
<td>sceneState</td>
<td>Ask for a scene’s current state</td>
</tr>
<tr>
<td>scenePlayers</td>
<td>Ask for agents in a scene</td>
</tr>
</tbody>
</table>
Protocol $P''$: coordination protocol between scene manager and governors.
Transition management

- Each transition is managed by a transition manager.

- Agents within a transition can ask for target scenes to join.

- The transition manager is in charge of controlling when the transition can be fired (agents can move).
Norms managed as a rule-based system.

Constructed from an ISLANDER specification.

The facts are illocutions uttered by agents.

Each governor manages his agent’s obligations.
A norm $N_i$:

$$(s_1, \gamma_1) \land \ldots \land (s_m, \gamma_m) \land e_1 \land \ldots \land e_k \land$$

$$\land \neg (s_{m+1}, \gamma_{m+1}) \land \ldots \land \neg (s_{m+n}, \gamma_{m+n}) \rightarrow obl_1 \land \ldots \land obl_p$$

is transformed into:

$$R_1_i : ( s_1, \gamma_1 ) \land \ldots \land ( s_m, \gamma_m ) \land e_1 \land \ldots \land e_k \rightarrow$$

$$assert(obl_1 \ldots obl_p) \land addRule(R2'_i, RB)$$

$$R_2_i : ( s_{m+1}, \gamma_{m+1} ) \land \ldots \land ( s_{m+n}, \gamma_{m+n} ) \rightarrow$$

$$retract(obl_1 \ldots obl_p) \land dropRule(R2_i, RB)$$
Norm management. 2\textsuperscript{nd} Generation

- Based on embedding the formal model of norms by Dignum et al. (EUMAS 05)
- Their approach adds \textit{conditional}, \textit{temporal} and \textit{precedence} notions to obligations:

\[
\text{OBLIGED}((\text{register\_admin \ DO \ correct(data)}) \ \text{IF} \ (\text{incorrect(data)}))
\]

\[
\text{OBLIGED}((\text{allocator \ DO \ assign(heart, recipient)}) \ \text{BEFORE} \ (\text{time(done(extraction(heart, donor))) + 6hours}))
\]
The main idea of our approach is the conversion of Dignum’s norms into Jess rules.

We classify norms into:

- **conditional** *(IF clause)*,
- **time-dependent** *(BEFORE, AFTER or BETWEEN clause followed by a date or time period)*
- **action-dependent** *(BEFORE, AFTER or BETWEEN clause followed by an action)*.
Norms – Classification examples

- **Conditional**
  
  \[ \text{SANCTION} \quad (A \text{ credit} = A \text{ credit} = 10) \]
  
  \[ \text{IF VIOLATED} \quad (\text{OBLIGED}(\text{utter}(S, W,
  \quad \text{inform}(A, R1, B, R2, \text{deliver}(IT'))))) \]
  
  \[ \quad \text{BEFORE} \quad 15/10/05 \) \)

- **Time-dependent**
  
  \[ \text{OBLIGED}(\quad \text{utter}(\text{deliver}, w0,
  \quad \text{inform}(C, \text{storemanager}, A, \text{buyer}, \text{deliver}(IT'))) \]) \]
  
  \[ \quad \text{BETWEEN} \quad \text{3 day, 15 days} \]

- **Action-dependent**
  
  \[ \text{PERMITTED} \quad (\text{utter}(\text{auction}, W,
  \quad \text{inform}(A, \text{buyer}, B, \text{auctioneer}, \text{bid}(IT, P)))) \]
  
  \[ \quad \text{BETWEEN} \quad (\text{uttered}(\text{auction}, w0,
  \quad \text{inform}(B, \text{auctioneer}, \text{all}, \text{buyer},
  \quad \text{offer}(IT, P))) \)) \]
  
  \[ \quad \text{uttered}(\text{auction}, w2,
  \quad \text{inform}(B, \text{auctioneer}, \text{all}, \text{buyer},
  \quad \text{sold}(IT, P, C))) \)) \]

Action1 \[ t_1 \]

Action2 \[ t_2 \]
Norms – Conversion into rules

- **Conditional**: The translation of IF sections is directly realised by placing the conditions in the LHS of a Jess Rule.

- **Time-dependent**: We use the user-defined function `(set-deadline ?deadline ?rule)` to add the rule after the deadline has passed. Then, we check with the rule that asserts a violation if the illocution has not been uttered.

- **Action-dependent**: We add a rule that asserts a violation if the action has been done and the obliged illocution has not.
Norms – Conversion Example

Obligation
Condition
Deadline
Deadline calculation
Post-deadline rule
Countdown start

(defrule cob-1
  (uttered (agent ?a) (scene payment)
    (state w0) (receiver payee)
    (performative inform)
    (content pay ?it ?price ))
  =>
  (assert (0 (agent storemanager) (scene delivery)
    (state w0) (receiver ?a)
    (performative inform)
    (content deliver ?it)))
  (bind ?date (new java.util.Date))
  (bind ?deadline (add-date ?date 0 0 15 0 0 0))
  (bind ?rule (str-cat
"(defrule cob-1-deadline "
  "(not(uttered (agent payee))"
  "(scene delivery)"
  "(state w0) (receiver " ?a ")"
  "(performative inform)"
  "(content deliver " ?it ") ) )")
  
  "(assert (V (type negative))"
  "(constraints \"before \"
    (?deadline toString) \\
    \")"
  "(agent payee) (scene deliver)"
  "(state w0) (receiver " ?a ")"
  "(performative inform)"
  "(content deliver " ?it "))))")
  (set-deadline ?deadline ?rule ) )

OBLIGED( utter(delivery, w0,
  inform(C, storemanager, A, payer, deliver(IT))))
BEFORE 15 days
IF uttered(payment, w0,
  inform(A, payer, B, payee, pay(IT, P))))
AMELI implementation features

- **GENERAL PURPOSE**
  - Institution Specification (XML format)

- **SCALABLE**

- **AGENT-BASED**
  - Participation

- **MIDDLEWARE**
  - Communication Neutral
  - Social layer (AMELI)

- **ARCHITECTURALLY NEUTRAL**
  - Communication Layer

- **COMMUNICATION NEUTRAL**
  - Participating Agents Layer
I. Introduction
II. Specifying electronic institutions
III. Running electronic institutions
IV. Are Electronic Institutions Enough?
V. Electronic Institutions Development Environment
VI. 3D Electronic Institutions
VII. Conclusions
Are Electronic Institutions Enough?

Let’s summarise first!

- EIs enact norms to structure the environment.
- Norms can be thought as physical laws or as social conventions that shape/constrain the evolution of interactions.
- Dynamics of the environment restricted to those that satisfy the social laws represented by norms and enacted by the coordinated actions of governors and staff agents.
- The flexibility of EIs comes from its clear separation of concerns between the internal behaviour of agents and their external interactions (environment modeling).
Are Electronic Institutions Enough?

- The environment is given structure so that agents have an easy comprehension of its working laws.
- These restrictions help a lot in the programming of agents by restricting the set of actions agents have to consider at each moment in time.
- And yet...

- ...MAS applications are usually concerned with some external world-of-interest (WoI) in addition to the agent society issues.
- The WoI is application-specific and refers to the part of the world that is relevant to the application. F.i. for a climate control application, the WoI comprises rooms, doors, heaters, etc.
The **World of interest** is application-specific and refers to the part of the world that is relevant to the application. F.i. for a climate control application, the WoI comprises rooms, doors, heaters, etc.
• AMELI allows to incorporate entities in the world of interest by adding services.

• Examples of services:
  • TimeService – Management of timers.
  • NormService – Tracking of agents’ normative positions.
  • ReputationService – Centralised reputation service.
  • ......
Example: Adding a reputation service to AMELI

```
<interface> EInstitutionService
ReputationService
+addInteraction()
+addReputation()
+getReputation()
+getPartners()
+getInteractionFreq()

<interface> EInstitutionProfile
RepProfile
+addReputation()
+getReputation()

Governor
Agent
```

Bridging Electronic Institutions with the WoI
I. Introduction
II. Specifying electronic institutions
III. Running electronic institutions
IV. Are Electronic Institutions Enough?
V. Electronic Institutions Development Environment
VI. 3D Electronic Institutions
VII. Conclusions
Electronic Institutions Development Environment

http://e-institutions.iiia.csic.es
Goal: To ease agent development

Approach:

- Graphical specification of an agent’s inner behaviour
- Automatic generation of agent skeletons via graphical tools
- Based on graphical specifications produced with ISLANDER
- Agent architecture based on tasks and performances
  - Performance – Actions within a particular scene
  - Task – Sequence of performances related by a performative structure path
Agent Builder GUI

Performance Graphical Specification
Goal: To verify dynamic properties of EIs

Approach:
  • To run discrete event simulations at the object level
  • Simulations include interleaving of EIs with some simulator of the world-of-interest
Simulating electronic institutions - SimDEI

Functions
- Synchronise Simulations
- Event observation
- Event translation

Scheduler
Probes

Environment (internal agents)

AMELI

Simulation Bridge
System Dynamics Simulator
I. Introduction
II. Specifying electronic institutions
III. Running electronic institutions
IV. Are Electronic Institutions Enough?
V. Electronic Institutions Development Environment
VI. 3D Electronic Institutions
VII. Conclusions
The Vision

Electronic Institution

3-D

avatar

perceives

controls

perceives

interface

knowledge exchange and intelligence enhancement

autonomous agent

human

interacts
Objectives

- New Metaphor: 3D Electronic Institutions
- Establish Trust
- Interact within a Social Environment
- Participants are
  - Human Users
  - Autonomous Software Agents
Objectives

- Reliability and Security
- Immersive 3D Experience
- Information-rich Environment
- Learning
One of the most important social concepts
Helps humans to cope with their social environment
It is present in all human interaction
Implicit trust assessment underlies every traditional bargain
Drawback in electronic markets: face-to-face interaction is missing
Increased risk level related to online markets
Social Environments

- Social interaction is the key feature
- Conversations
- Virtual Worlds are spaces where people meet
- Strong relation to Trust

“Whom do you know?”
3D Virtual Worlds
3D Virtual Worlds

- Space is designed and arranged according to *human daily experiences*

- Space and objects in space produce an *immersive environment*

- Construct a *virtual representation* of a particular domain
3D Virtual Worlds

- A convenient and alternative *interface* for user interaction
- Social experience: already through the simple presence of others
- Versatile ways of Interaction
Humans live in well-structured environments designed according to well-known **Metaphors**
- Rooms, Buildings, Streets, Districts, ...

Virtual Worlds might be designed according to similar metaphors

**Avatars** are the embodied representation of human users within the **Virtual World**
Design Considerations

- **Space** is used for modelling impressions
  - objects in space convey implicit meaning (buildings)
  - social power is expressed by “height”
- **Proximity** might indicate things of a similar type or group
  - eg. avatars located close to an object talking about that object (social context)
- **Space** and **Proximity** relate strongly to a person’s cultural background
Design Considerations

- Navigation: “Don’t get lost!”
- Visual Distance
  - the distance to an object influences the level of detail
- Audibility Distance
- Viewpoint
  - “What is an Avatar looking at?”
- Appearance of Avatars
Consider: “Besides the benefits obtained by adding an additional dimension for visualization purposes, this new *degree of freedom* might introduce *new difficulties*. Not every application domain has a suitable and usable representation in a 3D.”
No, 2D and 3D!

- 2D elements support the usability of 3D representations

- Combine the Advantages

- Selection techniques
3D Electronic Institutions combine the two metaphors of Electronic Institutions and 3D Virtual Worlds into one single metaphor

- retain the features and advantages of the original metaphors
- the essence is "opening" Electronic Institutions to human users
- explore the relationship between humans and software agents in a 3D Virtual Space
A contradiction?
3D Electronic Institutions

- Framework consists of three layers
  - Social User Interface Layer (top)
    - 3D Virtual World
  - Communication Layer (middle)
    - Connection and Communication
  - Electronic Institution Layer (bottom)
    - The actual Institution execution
3D Electronic Institutions

3D Interface (U1) 3D Interface (U2)

Social User Interface Layer

Causal Connection Server

Atmosphere Community Server

Communication Layer

Governor P1  AMELI  Governor P2

Electronic Institution Layer
Agent-mediated Environment: *The Governor*

Agents are connected to the infrastructure via Governors

“safe-guarding” the institutions and check whether a particular message is allowed to be said at the current stage or not
3D Electronic Institution

Diagram showing the layers and components of a 3D Electronic Institution:
- **3D Interface (U1)** and **3D Interface (U2)**
- **Social User Interface Layer**
- **Causal Connection Server**
- **Atmosphere Community Server**
- **Communication Layer**
- **Governor P1**, **AMELI**, and **Governor P2**
- **Electronic Institution Layer**
Communication Layer

- Consists of two parts
  - Adobe Atmosphere Community Server
  - Causal Connection Server
- Adobe Atmosphere Community Server handles multiple users; i.e. the Virtual World is shared among users
- Causal Connection Server builds on top
Causal Connection: “whenever a change is made in the representation of the system, the system itself changes to maintain a consistent state and vice versa“

Reflective systems: “the representation of the system is part of the system itself“
Each instantiated Virtual World (i.e. the human user) communicates via the Communication Layer with its associated Governor.

Actions performed by users are passed in terms of messages from the user interface to the communication layer.
The Causal Connection Server captures these messages and postpones the actual execution.

Messages are sent to AMELI in order to determine their “validity" and checks whether a particular message goes in line with the Electronic Institution rules or not.

A positive validation results in executing the requested action.
3D Electronic Institution

Diagram:
- 3D Interface (U1) and 3D Interface (U2)
- Social User Interface Layer
- Causal Connection Server
- Atmosphere Community Server
- Communication Layer
- Governor P1 and Governor P2
- AMELI
- Electronic Institution Layer
Social User Interface Layer

- 3D User Interface
- Adobe Atmosphere Player
  - Free download from Adobe Website
  - Player is embedded in HTML Page
  - Multimedia (Sound, Movies, Live Streams ...)
  - A combination of JavaScript and Java is used to communicate with “outer world“
- One Player per User
Social User Interface Objectives

- Immersive and fancy ;-) 
- Usable
  - consistent, convenient, efficient, clearly designed
  - not overcrowded, not distracting
- Social interaction
  - visual (e.g. via gestures)
  - facial expressions
  - sound (even speech)
  - textual
Social User Interface Objectives

- Awareness
  - knowledge about my own location and the location of other participants
  - being aware of different types of participants (human users or software agents)
  - distinguish between internal and external participants (avatar visualization code)
Believability

- Real vs. Surreal visualization; for instance, do we need *gravity* or *collisions*?
- “Can a visualization be too real?“
- Spatial layout of the Virtual World: Euclidian approach vs. “teletransportation“
John: I like this one, what are the exact dimensions of the painting?

Artist: 3.30m x 4.70 m

John: It's a pity that I'm still not finished with the 3D model of my office.

Artist: yeah, otherwise we could try it on.

Would be great.
3D Electronic Institution Specification
  • As an XML schema which enhances the Electronic Institution Specification

Specification Transformer
  • Automatically converts the specification of an Electronic Institutions into a 3D Electronic Institution Specification
- Design Tool for System Engineers
- Input: Electronic Institution Specification
- Output: 3D Virtual World
- A basic conversion is done automatically
- Additional features are incorporated by the System Engineer (eg., furniture)
Discussion
I. Introduction
II. Specifying electronic institutions
III. Running electronic institutions
IV. Are Electronic Institutions Enough?
V. Electronic Institutions Development Environment
VI. 3D Electronic Institutions
VII. Conclusions
Conclusions

- Engineering open multi-agent systems is a highly complex task.
- Electronic institutions reduce this complexity by introducing normative (regulatory) environments.
- We have presented an Electronic Institutions Development Environment (EIDE) that facilitates the deployment of electronic institutions.
- EIDE targeted at supporting environment engineering in open multi-agent systems.
- 3D Virtual institutions as a user friendly environment