Easy programming of Agent based Electronic Institution with UTOPIA

Pierre Schmitt, Cédric Bonhomme and Benjamin Gateau
Centre for IT Innovation / Centre de Recherche Public HENRI TUDOR
Luxembourg, G.D. of LUXEMBOURG
{pierre.schmitt,cedric.bonhomme,benjamin.gateau}@tudor.lu

Abstract—In Multi-Agent Systems (MAS), Organizations are means to structure cooperation and collaboration between agents. MoiseInst is a normative Organization model giving the possibility to constraint agents behavior according to four dimensions (structural, functional, contextual and normative). Mabeli as Electronic Institution model allows the supervision of MoiseInst Organizations compliance through an arbitration system. The difficulty is to easily instantiate such Organizations to obtain a dynamic entity in which agents can evolve. In this paper we introduce Utopia, our Institution-oriented and Institution-based programming framework. Utopia permits to easily and automatically set up a MAS thanks to a XML MoiseInst Specification file. The framework convert this file into an innovative mathematical structure namely a recursive graph. We show a concrete application of the prototype through RED, an EUREKA/CELTIC European project use-case.

Index Terms—Multi-Agent Systems, Electronic Institution, Recursive Graphs, Institution Oriented Programming

I. INTRODUCTION

Our field of research is multi-agent systems (MAS), which are build with a set of agents. These agents run in parallel so MAS can be a technique for parallel computing. Actually, agents aim at solving together a global problem or achieving a common goal. Multi-agent systems can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to solve. For these reasons a MAS is a very good way to better exploit the resources of a machine at software level.

A particular point of our interest about MAS is figuring out a way to organize agents and set up a self-governing behavior through cooperation between agents.

In human societies, Institutions define rules [1] that enclose all kinds of formal or informal constraints used by human beings to interact. In Multi-Agent System (MAS) domain, Electronic Institutions have been introduced to model rules with normative systems [2]. That is why we define Electronic Institutions as a set of agents which behave according to Norms and by taking into account their possible violation (and sanction).

These last years Electronic Institution platforms have been improved thanks to new services making them able to express cooperation schemes defined by the user with an Organization Modeling Language such as for instance MoiseInst [3], ISLANDER [4], OMNI [5]. The aim of these services is to constraint and supervise agent’s actions and interactions in order for them to achieve some global Goals. We call those explicit cooperation schemes Organization Specification (OS).

The model used to specify the organization of an Electronic Institution is MoiseInst [6]. In this context, the functioning of the agents is supervised and controlled with a set of Institution services regrouped in a specific “normative middleware” called SYNAl on which the agents execute themselves.

This paper aims at presenting how it is possible to easily implement an Electronic Institution specified with MoiseInst, supervised with SYNAl and in which standard agents provided with the platform evolve and achieve their Goals. For that, three steps have been needed:

1) Define the structure of data in which the OS will be stored.
2) Develop a set of agents (working in the JADE1 agent framework [7]) able to supervise an Organization Entity (OE) instantiating the OS defined by an user.
3) Develop a template of JADE based agents able to evolve in the OE (i.e. able to play Roles and achieve Goals) by loading specific behaviors provided by the user in order to execute actions achieving the Goals defined in the OS.

The paper is built as follows: in Section II we present rapidly MoiseInst and SYNAl (Section II-A) and then the concept of recursive graph (Section II-B) composing the foundations of our work. Next section deals with the implementation of the framework (named UTOPIA) allowing the implementation of such Electronic Institution (Section III). At last, before conclude, the Section IV illustrates the use of UTOPIA through an application of security policies deployment developed in the context of European RED project.

II. FOUNDATIONS

A. Normative Organization Modelling

MoiseInst [6], [8] is founded on the Moise Inst + organizational model [3]. It is composed of the following components that are used to specify an Organization of agents in terms of structure, functioning, evolution and Norms (see Figure 1):

- A Structural Specification (SS) defines: (i) the Roles that agents will play in the Organization, (ii) the relations between these Roles in terms of authority, communication

1Java Agent DEvelopment framework

2MoiseInst: Model of Organization for multi-agent System
or acquaintance, (iii) the Groups, additional structural primitives used to define and organize sets of Roles;

- A Functional Specification (FS) defines global business processes that can be executed by the different agents participating to the Organization according to their Roles and Groups;

- A Contextual Specification (CS) specifies, a priori, the possible evolution of the Organization in terms of a state/transition graph;

- A Normative Specification (NS) defines the deontic relations gluing the three independent Specification (SS, FS, CS). This NS clearly states rights and duties of each Roles/Groups of SS on sets of Goals (Missions) of FS, within specific states of CS.

These four Specifications form the Organizational Specification (OS). The Organizational Entity (OE) is then built from the set of agents that have adopted a Role according to the SS of the OS.

From this time the SYNAI[^3] [8] middleware manages and controls the functioning of this OE (composed of the entities - states- of every Specification NE, CE, SE and FE). As depicted on Figure 2, SYNAI is composed by a set of manager agents supervising the actions of agents “Agt” on the OE.

This layer is in charge of: (i) managing the life cycle of SS as entering/exiting of agents within the Organization, or requesting/leaving of Roles or Groups by the agents, (ii) coordination of the concurrent execution of FS as commitment to Missions or achievement of Goals, etc, (iii) dynamic and evolution of the Organization state through the CS, (iv) the monitoring and supervision of Norms of NS activated/deactivated by the evolution of the Organization.

While agents evolve inside the organization, agents of SYNAI have to interpret and “understand” the OS (in order to respect it or to control it). For that, we need to structure the data of the organization and to this end, we chose recursive graph.

**B. Recursive graph**

Recursive graphs are innovative mathematical structures [9] widely used to have a very generic representation of data. They could be very useful to represent any recursive structure such as UML diagrams [10] or to solve complex problems such as optimization of distributed processing [11]. In our workcase, recursive graph are helpful to store heterogeneous and recursive data of the Organization Specification.

Indeed, MOISE\textsuperscript{Inst} is mostly recursive: in the Structural Specification, Groups can include others Groups; in the Functional Specification, Missions can include others Missions, etc... In order to represent this kind of data with Object-Oriented Programming we have to design recursive classes for the four specifications. For example, in Figure 3, we show a simple way to represent the Roles and Groups of the Structural Specification.

To avoid repeating this design for every specification, we use recursive graphs to obtain a properly factored code (See Figure 4).

[^3]: SYstem of Normative Agents for Institution
using the AbstractLink class in order to manage several types of edges with the same graph. For the collections of data we use hash-maps (not represented on Figure 4)

Thanks to this structure, we can represent the whole OS. For the example above, Role implements SimpleVertex, Group implements RecursiveGraph and we have defined every type of relations between roles or groups (communication, authority...) with a class RoleLink which implements AbstractLink.

To conclude about our data representation, using recursive graphs allow us to :

• Have a factored an coherent code;
• Use the same type of objects for everything which makes the data sharing more easier and that is an obvious pro in a distributed software like UTOPIA⁴;
• See the whole problem of the implementation of an Electronic Institution as graph problems;

III. IMPLEMENTATION OF UTOPIA

A. Motivations

Our research objectives are to design and implement a flexible and intelligent MAS-factory (UTOPIA) using the Electronic Institution paradigm. Our challenge is to automatically obtain a MAS as close as possible to a MAS build from scratch for a specific use-case.

 Indeed, for each Electronic Institution to implement, there is redundant work to do if you do not reuse previously written code. We can briefly reduce the work to:

1) Manage the data of the Organization Specification (OS)
2) Create agents for every Role
3) Manage communication possibilities between agents
4) Allow appropriate data transfers from OS to agents
5) Manage contexts and transitions in order to let the Institution evolve
6) Synchronize agents
7) Find what goals an agent have to realize
8) Implement arbitration layer (SYNAI )
9) Implement what actions have to be done for every specified Goals

As seen before, we use recursive graphs for the data representation and its sharing between agents (1 & 4). Thanks to Jade, a framework dedicated to the agent oriented programming, we handle problems such as communication and synchronization (2, 3 & 6). To fit the others requirements of an Electronic Institution we have implemented several managers, a set of particular agents:

• ContextManager related to the Contextual Specification (5);
• RoleManager for the Structural Specification (2);
• GoalManager dedicated to the Functional Specification (7);
• Supervisor an agent which create and supervise all the managers above and implements SYNAI . (8)

Finally, thanks to UTOPIA to implement and deploy an Electronic Institution, there is only two things to do:

1) Describe the needed Electronic Institution thanks to a XML file;
2) Implement what actions have to be done for every specified Goals, with dedicated classes runnable on the fly by UTOPIA. (9)

B. Managers

1) ContextManager: It aims at managing all Contexts (states) in which the system can be found according to the Contextual Specification (see Figure 5). It can be seen as a finite-state parallel machine running a thread for each Scene. A Scene is a set of contexts, transitions and possibly other sub-scenes.

The Scene’s thread starts with the initial vertex and goes to the next state if the received message is a proper transition. The commands run by the ContextManager are mainly transitions and queries for all current states.

Finally, when each thread has finished its execution (when the final state of the CS is reached), ContextManager informs all the agents that the instantiation of the Specification is complete. ContextManager is thus the actor that allows termination of UTOPIA.

2) RoleManager: It allows the management of the Roles (see Figure 6) which involves to:

• Find a distribution of the Roles (what Roles will be played by which agents) and to return the number of Agents needed to instantiate the Structural Specification.
• Provide all the Roles an agent must play or provide the same list again if a given agent asks several times.
• Indicate if all agents play an appropriate Role as needed.

3) GoalManager: GoalManager mainly performs a mapping between the Functional Specification Goal set and the concrete instances of classes implementing the actions associated with each Goals (see Figure 7). When this mapping is done, it can provide classes instances after each agent’s queries.
4) **Supervisor:** It is at the top of the system hierarchy (see Figure 8) and its goals are:
- Creating a recursive graph based on the XML file of the Organization Specification (OS).
- Creating the 3 managers that we have seen: ContextManager, RoleManager and GoalManager.
- Sending OS’ subgraphs to the corresponding managers namely CS for the ContextManager, SS for the RoleManager and the FS to GoalManager.
- Caring for the Normative Specification (NS) as the entire OS is required to manage it.

5) **UtopiaAgent:** A first step towards intelligent behavior from agents is to develop generic agents (see Figure 9) able to specialize independently in order to behave differently. To this end, the agent will retrieve a Role and will act (by running appropriate Goals thanks to the GoalManager) according to the Contexts of the Institution and Norms that apply to it. The main idea is to obtain a self-organization of agents.

### IV. **Utopia use-case: the RED project**

Our use-case is part of a demonstrator set up in the context of the RED project [12] which defines and designs solutions to enhance the detection/reaction process, improves the overall resilience of IP networks to attacks by embedding means to enrich the alert with better characterized information, and additional information about the origin and the impact of the security incident.

To provide the detection and reaction functionalities, RED proposes an architecture containing a set of elements, depicted in Figure 10:
- **ACE (Alert Correlation Engine):** this element is in charge to receive alerts from network nodes, and enhances the detection of attacks by combining several diagnosis combinations.
- **PIE (Policy Instantiation Engine):** this element receives the information about attacks from the ACE and instantiates new security policies to react to the attack in a high level reaction loop. This paper is focused on this element.
- **PDP (Policy Decision Point):** this element receives the new security policies defined by the PIE and deploys them in the enforcement points.
- **RDP (Reaction Decision Point):** this element receives the information about attacks from the ACE and decides of how to act in a mid level reaction loop.
- **PEP/REP (Policy Enforcement Point/Reaction Enforcement Point):** This component, outside the RED node, enforces the security policies provided by the PDP and the reaction provided by the RDP. It also performs an immediate low level reaction.

RED proposes three different types of reaction based on level of diagnosis required to apply them:
- Immediate reaction, which is an automatic response with a diagnosis based on the capabilities embedded in the device and decided by the PEP/REP,
• Short term reaction, where the diagnosis is done with a limited and local vision of the monitored information system, decided by the RDP based on the information provided by the ACE and which does not instantiate new security policies,
• Long term reaction, where the diagnosis is done with a global vision of the monitored information system, decided by the PIE and which generate new security policies based on the ACE alerts which are sent to the PDP to deploy them in PEP.

A multi-agent system is used to represent RED nodes. Each component is represented by an agent playing a Role (ACE, RDP, PIE, PDP, REP, PEP) of the node which is represented as a MioseJust Organization. In the following, we will describe the Goals that agents have to achieve in a context of a black-hole attack.

A. Black-hole attack and countermeasures

In our scenario Alice and Bob are communicating with help of a VoIP service provided by a SIP server. A Malicious node executes an attack structured in two successive steps. First, the Malicious node changes the ARP tables of Alice, Bob and the SIP Server (ARP poisoning) in order to have all the traffic routed by itself. Then, it carries out a black-hole attack by dropping (not retransmitting) the packets. As a result, the conversation between Alice and Bob cannot progress.

Once the attack succeeded, an intrusions detection tool detects the attack and sends alerts to the PIE and the RDP through the ACE. The agent playing the Role of RDP have to apply a short term reaction by asking PEP to delete their ARP entries corresponding to the MAC address of the malicious node. The agent playing the Role of PIE aims at implementing new policies forbidding the input and the forward of traffic coming from the malicious node (via its MAC address) and adding static ARP entries binding the real IP addresses and MAC addresses. Then the PIE agent sends these new policies to PDP which transform them into script and/or executable command regarding to PEP’s specifications (type, host, OS, etc.). At last, agents playing PEP Role have to execute command and/or scripts on the device they interface. We will see more precisely in the next section how an Organization is implemented with UTOPIA in order to represent a RED node as an Electronic Institution.

B. Implementation with UTOPIA

As seen before, using UTOPIA to design and deploy a MAS implies to do two things:

• Describe the needed Electronic Institution thanks to a XML file;
• Implement what actions have to be done for every specified Goals

1) Seeing RED as an Electronic Institution:

• Structural Specification: we can deploy a MAS where agents play the appropriate Roles, namely ACE, PIE, PDP and PEP. Thanks to cardinalities, the MAS composition can respect the RED architecture: ACE, PIE and PDP are played by only one agent and PEP are distributed over the network devices.

   `<StructuralSpecification>
   <Group id="RED" min="1" max="1"><Role id="rACE" min="1" max="1"/>
   <Role id="rPDP" min="1" max="1"/>
   <Role id="rPIE" min="1" max="1"/>
   <Role id="rPEP" min="2" max="2"/></Group>
   <Link source="rPIE" destinations="rPDP">
   type="communication" [...]
   </Link>
   [...]
   </StructuralSpecification>

• Functional Specification: we can handle the agent behavior after an attack with several Missions composed by Goals dedicated to messages reception or message sending:

   `<FunctionalSpecification>
   <GoalId>gPIESend</GoalId>
   <GoalId>gACESend</GoalId>
   <GoalId>gPDPListen</GoalId>
   <GoalId>gPEPLListen</GoalId>
   <GoalId>gPEPILListen</GoalId>
   </FunctionalSpecification>

• Contextual Specification: here there is no reasons to use specific Contexts, we only distinguish two Contexts (life and dead) and a transition “finish” in order to terminate RED:

   `<ContextualSpecification>
   <Scene id="RED" initialCtx="life" finalCtx="dead">
   <GoalId>gPIESend</GoalId>
   <ContextDesc id="dead"/>
   <ContextDesc id="life"/>
   <Transition id="t1" source="life" target="dead">
   eventId="finish"
   </Transition>
   </Scene>
   </ContextualSpecification>

• Normative Specification: force the agents playing the Roles of ACE, PIE, PDP and PEP to do their associated Missions, that is to say, to run appropriate Java Goal implementations.

   `<NormativeSpecification>
   <Norm id="N1" weight="1" operator="O"/>`
2) Implementation of the Goals: Obviously, all the Goals specified imply concrete actions. For that, UTOPIA permits a mapping between declared Goals and Java classes. To make them runnable, they must implement the GoalImplementation class and more particularly the "run()" virtual method. Thanks to this virtual class we offer some interesting primitives to the final user so it is very easy to send or receive messages and XML alerts.

We have to link all the implementations with declared Goals into the XML OS file using the “DomainKnowledgeSpecification” part:

```xml
<DomainKnowledgeSpecification>
  <Goal id="gPDPListen" class="red.pdp.GPDPListen"></Goal>
  <Goal id="gPEPListen" class="red.pep.GPEPListen"></Goal>
  <!-- ...
```

Finally, we can use the framework like this to implement, for example, the PDP specialization:

```java
public class GPDPListen extends GoalImplementation{
    public void run(){}
    while(true){
        Object obj=getActorMessage();
        if(obj==null){
            if(obj instanceof ActorMessage){
                ActorMessage am=(ActorMessage)obj;
                if(am.getSender().contains("PIE")){
                    //This is an ORBAC formatted message ...
                } else if(am.getSender().contains("ACE")){
                    //This is an IDMEF formatted message ...
                }
            }
        }
        try { Thread.sleep(100); } catch (InterruptedException e) {}  
    }
}
```

3) Results: As results we can see UTOPIA running agents automatically specialized in PDP (Figure 11) and PEP (Figure 12) thanks to its supervisor and managers.

V. CONCLUSION

In this paper we described an Electronic Institution programming framework named UTOPIA based on MOISEInst for the Organization Specification and on recursive graph for the Organization representation. Thanks to a recursive graph, all the homogeneous data are stored in an unique recursive structure, allowing us to easily distribute the shared information between agents of UTOPIA using concepts such as sub-recursive graphs.

A set of supervisor and manager agents has been developed in order to help generic agents to specialize themselves autonomously (and therefore smartly) to achieve Organization's objectives by knowing their relationships with other agents (RoleManager), by knowing what they may, must or must not do (Supervisor) and finally by knowing how to act properly (GoalManager).
With the RED use-case we showed how easily the essential problematics of setting up a Multi-Agent System could be solved with UTOPiA and its powerful architecture using an Electronic Institution paradigm. Actually, UTOPiA allows to simply deploy a MAS without any need of network programming (as Socket coding or thread management). Furthermore, with this kind of network abstraction, the implementation of RED is completely reusable: we can run the system on many different networks. Moreover, it is far easier to brings into the MAS development many security specialists, as Electronic Institution permits to clearly separate the different system Goals and thus, the different security problematics.

Despite the easiness of implementing a working Electronic Institution that UTOPiA brings, as demonstrated in a real use-case, some improvements can be considered. Actually, the way of managers and supervisor to control the functioning of the organization is basically a centralized arbitration system. However the multi-agent system principles advocate decentralization. As a consequence, a first evolution could be done in order to obtain an Electronic Institution allowing the distribution of the OE and SYNAI without putting the optimization of the role distribution aside. Moreover, the agents’ decision taking mechanisms could be improved to exhibit a smarter behavior in order to choose the right Goals to achieve at the right time more efficiently.

ACKNOWLEDGMENT

This work has been co-funded by European CELTIC project RED (CP3-011) and by Luxembourger FNR-CORE project TITAN (C08/IS/21).

REFERENCES