Enhancing the Environment with a Law-Governed Service for Monitoring and Enforcing Behavior in Open Multi-Agent Systems

Rodrigo Paes[1], Gustavo Carvalho[1], Maíra Gatti[1],
Carlos Lucena[1], Jean-Pierre Briot[1,2], and Ricardo Choren[3]

[1] PUC-Rio, Rua M de S Vicente 225, Gávea
22453-900, Rio de Janeiro, Brazil
{rbp, guga, mgati, lucena}@inf.puc-rio.br
[2] LIP6, 4, place Jussieu
75005 Paris, France
jean-pierre.briot@lip6.fr
[3] IME, Pça General Tibúrcio 80, Praia Vermelha
22290-270, Rio de Janeiro, Brazil
choren@de9.ime.eb.br

Abstract. Environment is an essential part of any multi-agent system (MAS), since it provides the surrounding conditions for agents to exist. For some sort of systems, the environment can be viewed as providing a set of services, in which some of them, such as directory facilities, are used explicitly by the agents to perform their tasks, and other such as monitoring, behavioral enforcement and security can be done transparently by the environment. We join the idea that the specification of environments of open multi-agent systems should include laws that define what and when something can happen in an open system. Laws are restrictions imposed by the environment to tame uncertainty and to promote open system dependability. This paper proposes a design approach and application of a middleware based on laws in multi-agent systems. The approach can be viewed as a set of services provided by the environment.

1 Introduction

The agent development paradigm has posed many challenges to software engineering researchers. This is particularly true when the systems are distributed and inherently open to accept new modules that may have been independently developed by third parties. Such systems are characterized by having little or no control over the actions that agents can perform. The greater the dependence of our society on open distributed applications, the greater will be the demand for dependable applications.

Environment is an essential part of any multi-agent system (MAS), since it provides the surrounding conditions for agents to exist [25]. Several aspects of MAS that are vital for agents do not belong to agents themselves but are part of the environment [5]. We believe that the specification of environments of open

D. Weyns, H.V.D. Parunak, and F. Michel (Eds.): E4MAS 2006, LNAI 4389, pp. 221–238, 2007.

multi-agent systems (MAS) should include laws that define what and when something can happen in an open system [4][5]. Laws are restrictions imposed by the environment to tame uncertainty and to promote open system dependability [15][17]. In this sense, the environment performs an active role monitoring and verifying if behavior of agents are in conformance with the laws. This kind of environment is composed of a governance mechanism, which is the mediator that enforces the law specification. Examples of governance mechanisms are LGI [15], Islander [7] and MLaw [18].

Governance for open multi-agent systems can be defined as the set of approaches that aims to establish and enforce some structure, set of norms or conventions to articulate or restrain interactions in order to make agents more effective in attaining their goals or more predictable [14].

Despite the growing interest in the area, mature governance approaches will only be achieved through the use of Software Engineering techniques and tools that support the development of more predictable and better quality software systems. There has been advance in this area through well founded engineering tools for governed interaction such as the work of LGI [15], Islander [7] and Rubino [21]. This paper deals with the above mentioned problem through a design approach and application of a middleware for governance in multi-agent systems. Notably, we propose the modeling of laws for governance, based on XML. This includes norms but also other abstractions such as protocols, scenes, constraints and clocks, in order to achieve a good expressivity. In another paper, we also addressed the issue of specialization of governance specifications using abstractions for extension [4]. The middleware can be used in conjunction with a specific agent platform (such as JADE [2]), and it permits configuration of interaction rules, monitoring of agent interaction and verification of the conformity between the interaction specification and the actual interaction. We have already used the middleware in a variety of different situations, such as monitoring criticality of agents [11], tests [20] and mediation of inter-bank operations.

This paper is organized as follows. Section 2 introduces the relationship between the environment and the interaction Laws. Section 3 introduces the main Governance concepts used in this paper. Section 4 presents the middleware for governance used in this work. Section 5 shows how the framework can be effectively used to enhance the notion of environment and presents an experiment that was performed using the middleware. Section 6 presents some related work and details on where this paper gives the contribution. Finally, Section 7 presents some discussion and conclusions about this work.

2 Environment and Interaction Laws

The precise definition of environment is still under the core of the discussions among the research community [26]. One of the reasons that definitions of environment have proliferated is that MAS have been applied to a wide range of different application domains [24]. For example, it is natural for designers of a business-to-business application to associate the environment with the existent infrastructure of hardware and software on which the agents will have to execute. In another domain, such as an

agent-based simulation of an ecosystem, the environment as well as the agents will be custom built for the application [24].

A very common distinction for environments is between physical and virtual environment. In the physical environment there are the physical constraints of the existent entities. An example is that in an agent system that controls robots in which two robots are not allowed to occupy the same place at the same time. The virtual environment provides the principles and processes that govern and support the exchange of ideas, knowledge and information [24].

A very deep discussion on environments can be found in [26]. They define an environment as a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources.

An environment provides the conditions under which an entity exists [16]. Multi-agent environments are typically open and have no single centralized designer, they contain agents that are autonomous, distributed and may be self-interested or cooperative. Furthermore, environments provide a computational infrastructure that enables agents to communicate with one another [13].

It is the responsibility of the environment to define the rules for, and enforce the effects of, the agents' actions. An appealing way to exert the necessary level of control over an agent in an open system is through an adequate MAS infrastructure [23], which can be viewed as part of the environment. The type of services provided by the infra-structure, and the way in which these services are enacted, limit the set of possible actions [23]. For that, a MAS designer can use a governing infrastructure to structure and shape the space of action within MAS in an open environment. This governing perspective allows managing agent interactions form an external and global point-of-view.

In this sense, the environment is viewed as an active entity that also contains the set of behavioral (social) laws, which is constantly monitoring and reacting to agents' actions. It is very important for the agents to be able to perceive the environment. Agents can use the percepts to update their knowledge about the environment or use it for decision making [24]. In this sense, a virtual environment should provide explicit data structures for notify changing on the environment state.

We have used the reference model of the environment (Figure 1) proposed in [26] to show how the environment can be viewed. It shows that the environment provides a set of services, in which some of them, such as directory facilities, are used explicitly by the agents to perform their tasks, and other such as monitoring, behavioral enforcement and security can be done transparently by the environment[1]. The list of services shown in Figure 1 shows an environment as the basic infrastructure for supporting agents' activities in a more dependable manner. In this paper we focus on the governance services (the law box in the figure), which are used to monitor and enforce the behavior of agents.

[1] In fact, it may be discussed if social laws (regulating interactions between agents) and social entities (like organizations) are, conceptually and architecturally speaking, part of the environment of a MAS or if they have a distinct existence (see, e.g., [9]). In this paper, we describe governance (of social laws) as part of the services offered by a MAS environment.

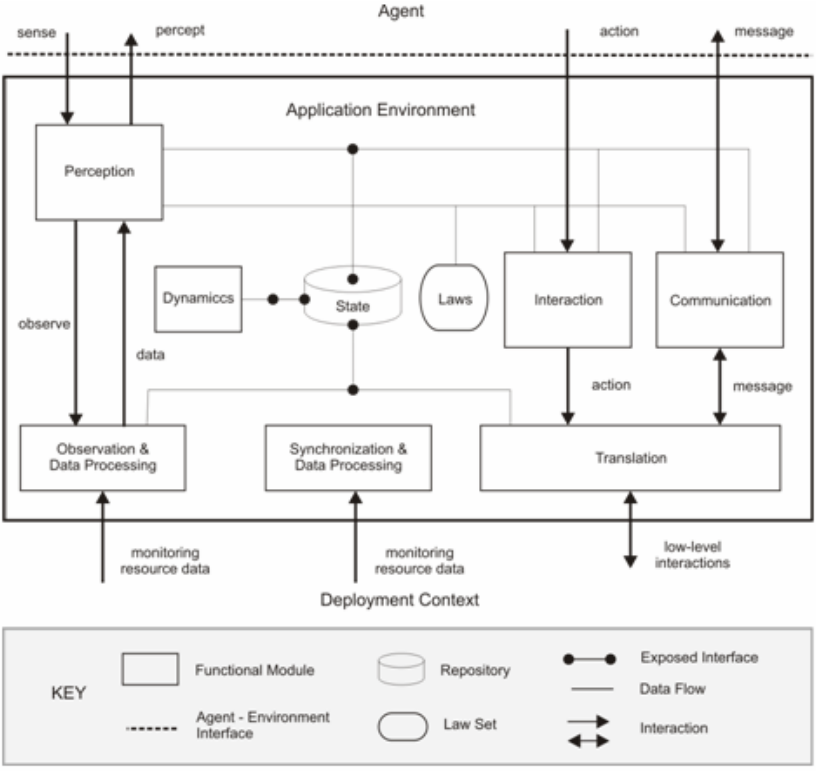

Fig. 1. Reference model of environments [26]

3 Governing Interactions

Law-governed architectures are designed to guarantee that the specifications of open systems will be obeyed. The core of a law-governed approach is the mechanism used by the mediators to monitor the conversations between components. We have developed a software support [18] that, whenever necessary, permits extending this mechanism to fulfill open system requirements or interoperability concerns. In this architecture, a communication component is provided to agent developers.

M-Law works by intercepting messages exchanged between agents, verifying the compliance of the messages with the laws and subsequently redirecting the message to the real addressee, if the laws allow it. If the message is not compliant, then the mediator blocks the message and applies the consequences specified in the law (Figure 2). This architecture is based on a pool of mediators that intercept messages and interpret the previously described laws. A more detailed explanation about how this architecture was in fact implemented can be found in [19]. As more clients are added to the system, additional mediators' instances can be added to improve throughput. In relation with the list of services shown in Figure 1, M-Law implements the monitoring and enforcing of agents' behavior.

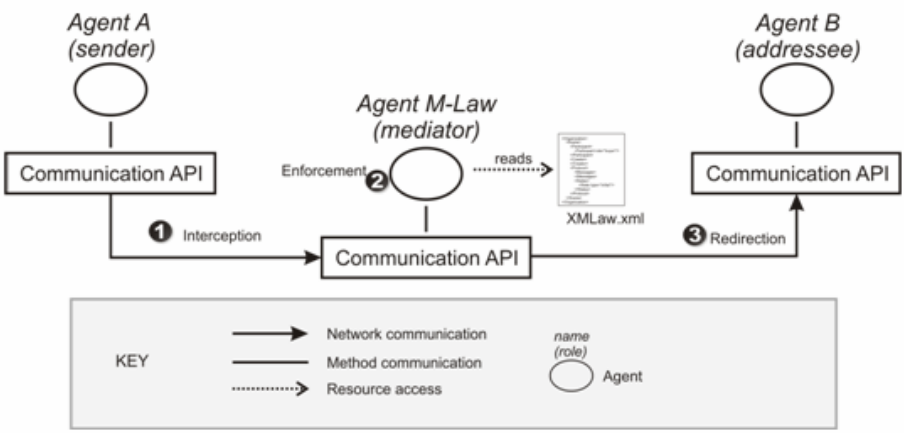

Fig. 2. M-Law architecture

M-Law was built to support law specification using XMLaw [17][3]. XMLaw is used to represent the interaction rules of an open system specification. As mentioned before, XMLaw is the description language used to configure the M-Law. These rules are interpreted by M-Law that analyzes the compliance of software agents with interaction laws at runtime. XMLaw represents the structure and the relationships between important law elements (Figure 3). A more detailed definition of the conceptual model can be found in [3] and [17]. A law is a description composed of law elements, such as e.g., protocols, norms, and scenes, as described in the next paragraphs.

Norms can be used to enhance scene and transition definitions; constraints in norms and transitions can act as filters of events; and actions can be used as an adaptation mechanism to support an active behavior of the environment in an open system. We selected some elements from XMLaw conceptual model to illustrate our proposal. Below, we will discuss XMLaw structure using the specification of laws for the TAC SCM example to facilitate its understanding.

The conceptual model uses the abstraction of Scenes to help to organize interactions. The idea of scenes is similar to theater plays, where actors play according to well defined scripts, and the whole play is composed of many scenes sequentially connected. Scenes are composed of Protocols, Constraints, Clocks, and Norms. It means that these four elements share a common interaction context through the scenes. Because protocols define the interaction among the agents, different protocols should be specified in different scenes. Scenes also specify which agent role has permission to create scene instances.

Statically, an interaction protocol defines the set of states and transitions (activated by messages or any other kind of event) allowed for agents in an open system. Norms are jointly used with the protocol specification, constraints, actions and also temporal elements, to provide a dynamic configuration for the allowed behavior of components in an open system. The mediator keeps information about the set of activated elements to verify the compliance of software agents, the set of deactivated elements and any other data regarding system execution.

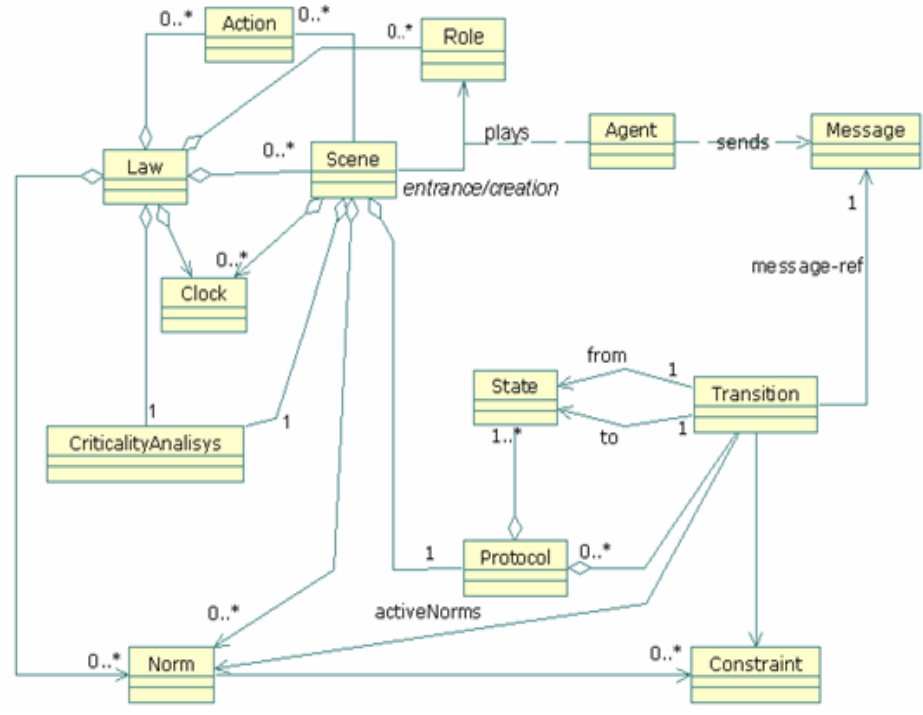

Fig. 3. Conceptual model of the elements that can be used to specify laws

Laws may be time sensitive, e.g., although an element maybe active at time t1, it might not be active at time t2 (t1 < t2). XMLaw provides the Clock element to take care of the timing aspect. Temporal clocks represent time restrictions or controls and they can be used to activate other law elements. Clocks indicate that a certain period has elapsed producing clock-tick events. Once activated, a clock can generate clock-tick events. Clocks are activated and deactivated by law elements. Both are referenced to other law elements. Below, we detail the structure of the elements that will be exemplified in this paper.

3.1 Simple Scenario

The example that will guide our explanation is the Trading Agent Competition - Supply Chain Management (TAC SCM). The TAC SCM [22][1][6] editions provide some evidence that the interaction specification evolves over time and so an extension support can reduce maintenance efforts.

The TAC SCM has been designed with a simple set of rules to capture the complexity of a dynamic supply chain. SCM applications are extremely dynamic and involve an important number of products, information and resources among their different stages. In our case study, we mapped the requirements of TAC SCM into interaction laws and agents are implemented with JADE [2].

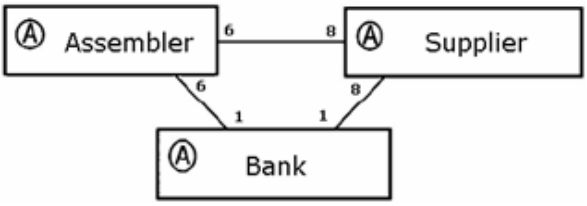

Fig. 4. Roles, relationships and cardinalities of TAC SCM

In TAC SCM, we chose the scenario of negotiation between the suppliers and assemblers to explain how interaction laws are used (Figure 1). According to [1], the negotiation process involves an assembler agent that buys components from suppliers. A bank agent also participates in this negotiation because an assembler must pay the components for the supplier. In this scenario, an assembler may send RFQs to each supplier every day to order components offered by the supplier. Each RFQ represents a request for a specified quantity of a particular component type to be delivered on a specific date in the future. The supplier collects all RFQs received during the "day" and processes them. On the following "day", the supplier sends back to each agent an offer for each RFQ, containing the price, adjusted quantity, and due date. If the agent wishes to accept an offer, it must confirm it by issuing an order to the supplier.

3.2 Norms

There are three types of norms in XMLaw: obligations, permissions and prohibitions. The obligation norm defines a commitment that software agents acquire while interacting with other entities. For instance, the winner of an auction is obligated to pay the committed value and this commitment might contain some penalties to avoid breaking this rule. The permission norm defines the rights of a software agent at a given moment, e.g. the winner of an auction has permission to interact with a bank provider through a payment protocol. Finally, the prohibition norm defines forbidden actions of a software agent at a given moment; for instance, if an agent does not pay its debts, it will not be allowed future participation in a scene.

In TAC SCM, one permission norm was created about the maximum number of requests for quotation that an assembler can submit to a supplier. According to the TAC SCM specification, each day each agent may send up to a maximum number of RFQs. Besides this permission, the constraint on the acceptable due date of an RFQ regulates the same interaction, the request for quote message.

The structure of the Permission (Code Fragment 1), Obligation and Prohibition elements are equal. Each type of norm contains activation and deactivation conditions. In the example, an assembler will receive the permission upon logging in to the scene (scene activation event) and will lose the permission after issuing an order (event orderTransition). Furthermore, norms define the agent role that owns it through the attribute Owner. In that case, the assembler agent will receive the permission. Norms also have constraints and actions associated with them, but these elements will be explained later. Norms also generate activation and deactivation events. For instance, as a consequence of the relationship between norms and

transitions, it is possible to specify which norms must be made active or deactivated for firing a transition. In this sense, a transition could only fire if the sender agent has a specific norm.

```
<Norm type="permission" id="AssemblerPermissionRFQ">
 <Assignee role-ref="assembler"
           role-instance="$assembler.instance"/>
  <Activations>
   <Element ref="negotiation"
            event-type="scene_creation"/>
  </Activations>
  <Deactivations>
   <Element ref="orderTransition"
            event-type="transition_activation"/>
  </Deactivations>
  <Constraints>
   <Constraint id="checkCounter" class="CounterLimit"/>
  </Constraints>
  <Actions>
   <Action id="permissionRenew" class="ZeroCounter">
    <Element ref="nextDay" event-type="clock_tick"/>
   </Action>
   <Action id="orderID" class="RFQCounter"/>
    <Element ref="rfqTransition"
             event-type="transition_activation"/>
   </Action>
  </Actions>
</Norm>
```

Code 1. Permission structure

3.3 Constraints

Constraints are restrictions over norms or transitions and generally specify filters for events, constraining the allowed values for a specific attribute of an event. For instance, messages carry information that is enforced in various ways. A message pattern enforces the message structure fields [17]. A message pattern does not describe what the allowed values for specific attributes are, but constraints can be used for this purpose. In this way, developers are free to build as complex constraints as needed for their applications.

Constraints are defined inside the Transition (Code Fragment 2) or Norm (Code Fragment 1) elements Constraints are implemented using Java code. The Constraint element defines the class attribute that indicates the java class that implements the filter. The use of Java code allows for the specification of complex user defined filter implementation. This class is called when a transition or a norm is supposed to fire, and basically the constraint analyzes if the message values or any other events' attributes are valid. In Code Fragment 2, a constraint will verify if the date expressed in the message is valid according to TAC rules; if it is not, the message will be blocked. In Code Fragment 1, a constraint is used to verify the number of messages

that the agent has sent until now; if it has been exceeded, the permission is no longer valid.

```
<Transition id="rfqTransition" from="as1" to="as2"
            message-ref="rfq>
 <Constraints>
  <Constraint id="checkDueDate" class="ValidDate"/>
 </Constraints>
 ...
</Transition>
```

Code 2. Constraint in transition tags

3.4 Actions

Environment actions, or just actions, are domain-specific Java codes that run integrated with XMLaw specifications. Actions can be used to plug services in an environment. For instance, an environment can call a debit service from a bank agent to automatically charge the purchase of an item during a negotiation. In this case, we specify in the XMLaw that there is a class that is able to perform the debit.

Since actions are also XMLaw elements, they can be activated by any event such as transition activation, norm activation and even action activation. The action structure is showed in the example of Code Fragment 1. The class attribute of an Action specifies the java class in charge of the functionality implementation. The Element tag references the events that activate this action, and as many Element tags as needed can be defined to trigger an action. In this example, the action is used to update the context of the norm, counting the number of submitted messages.

An action can be defined in three different scopes: Organization, Scene and Norms. An action defined in a Norm is only visible at this level. This means that any element in this scope can reference events issued by this action and that this action can get and update information at this level and upper levels. Actions defined in the scene scope can be referenced by any element at this level. And actions defined in the organization scope are visible to all elements at this level.

4 M-Law

Agent technology advances rely on the development of models, mechanisms and tools to build high quality systems. The design and implementation of such systems is still expensive and error prone. Software frameworks deal with this complexity by reifying proven software designs and implementations in order to reduce the cost and improve the quality of software. In this way, an object-oriented framework is a reusable, semi-complete application that can be specialized to produce custom applications [8].

M-Law was designed as an object-oriented framework, and its hotspots make possible to plug-in existing agent infrastructures, change the communication mechanism used by the agents, and plug-in new functionalities through the component module (to be detailed further in this section).

M-Law middleware has to provide the means to effectively support XMLaw and its evolutions. M-Law is composed of four modules: agent module, communication module, a mediator agent and an event module. The agent module contains classes that agent developers may use to develop agents. This module provides a set of facilities to interact with both the mediator and other agents through methods for receiving and sending messages. The Agent module uses a Communication module to send and receive messages. In fact, the Communication module contains a set of abstract classes and interfaces that have to be extended in order to provide real functionality. We have made some experiments using JADE Agent Framework to implement this module. In addition to Jade, we have also implemented a communication module using pure socket communication. This flexibility provides the means to build agents using different existing agent frameworks.

On the side of the mediator agent, which is in charge of monitoring and enforcing agent interaction, there are three main modules: Event, Component and Communication. Those modules are not visible to agent developers but they were used to build the mediator agent and they can be extended to support new functionalities. Agent criticality analysis presented in [11] is an example of the component module extension.

The event module implements event notification and propagation. It is basically an implementation of the observer design pattern [10] allowing elements for listening and receiving events. The communication module has a similar implementation as the communication module on the client side.

The elements such as scenes, clocks and norms, are implemented to be plugged into the component module. The component module defines a set of concrete and abstract classes and interfaces that allows new functionality to be inserted.

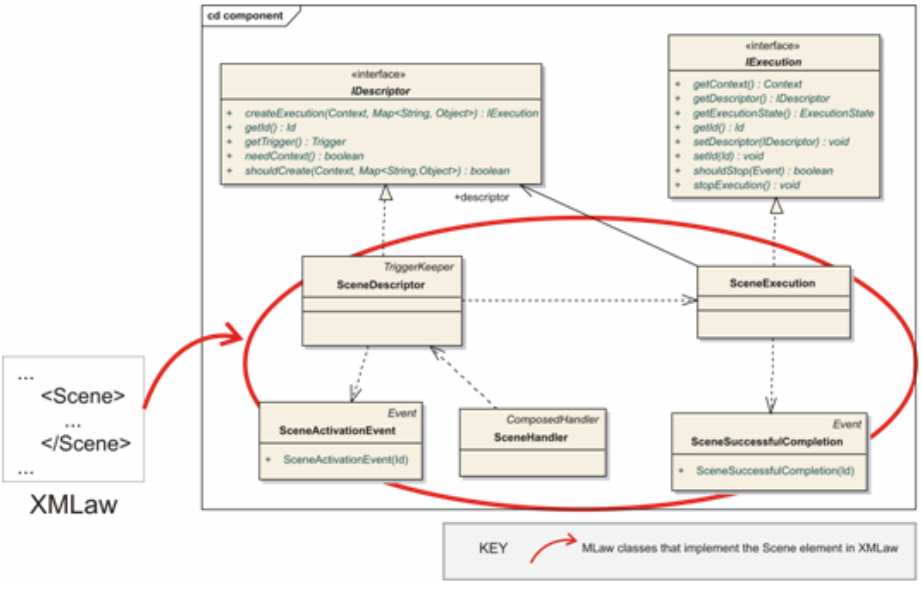

Fig. 5. Scene component

Components are the set of classes placed in the mediator that implements the behavior of the XMLaw language elements; for example, Figure 5 shows the scene element in XMLaw and its set of classes that implements its behavior. In Figure 5, the scene Element in XMLaw (therefore an XML element) is mapped to a descriptor (IDescriptor) and execution (IExecution) hierarchies in the internal implementation of the MLaw. Those hierarchies are hotspots of the component module, and they are used to plug in new elements in the law definition, allowing, for example, to change the XMLaw conceptual model.

As shown in Figure 5, the core classes and interfaces of the component module that provide the hotspots that have to be extended by concrete elements are:

- Handlers: SimpleHandler and ComposedHandler – The component module has a default XML parser that reads a law specification in XML and delegates the treatment of each XML tag to a specific handler. The handler is in charge of receiving the tokens provided by the parser and building the descriptor of the element.
- IDescriptor – It represents the object model of the XML tag. For example, in Figure 6, the scene tag in XML is represented by the SceneDescriptor class. Its main responsibility is creating execution instances of the descriptor.
- IExecution – An object that implements the IExecution interface is an instance of an element represented by an IDescriptor object. For example, a scene may be instantiated many times and even various scenes may be running at the same time (various auctions running in parallel, for instance). Each instance (IExecution) has to keep its instance attributes and control its lifecycle. The IExecution interface defines all the callback operations needed by the component module to control instances.

As an example of how M-Law works in a practical scenario, suppose an agent playing the role of an employer asks for increasing its own salary to other agent playing the role of accountant. However, there is a norm specified in XMLaw stating that employers are prohibited from asking for salary increase. Despite the simplicity of this scenario, the example is useful to illustrate the basic flow of events inside the M-Law. Then, M-Law works in the following way:

1. Mediator agent reads the XMLaw specification and starts the component module;
2. Employer agent calls its communication module and sends the request message asking for salary increase;
3. The communication module redirects the message to the mediator;
4. Mediator receives the message through its communication module;
5. Mediator fires an event of message arrival through event module;
6. Event module notifies the component module;
7. Norm element, which is part of the component module, receives the event, verifies that the message is not allowed and fires a message not compliant event;
8. The mediator receives the message not compliant event and as a consequence, does not redirect the message to the accountant agent.

The main design objectives of M-Law were simplicity, flexibility and reuse. That is why the elements were implemented as components. In this way, some architectural decisions made have direct impact over scalability in the current version of M-Law. The following items discuss in more details some architectural and design aspects of M-Law:

- Scalability – M-Law is implemented as a centralized mediator. In this way, it may become a bottleneck for very large systems. In fact, the design of XMLaw aimed primarily at expressivity (using abstractions such as constraints, scenes, etc.) and at flexibility (specialization of laws [4]). Also, although a centralized solution poses scalability questions, it allows the easy specification of global laws with no need for state synchronization. Furthermore, temporal problems are also avoided, once there is just one host controlling clocks. We are currently working on alternative, more decentralized, solutions to this problem. We are performing experiments on both a network of decentralized mediators, such as LGI [15], and hierarchal organized mediators, such as Internet DNS.
- Expressivity – M-Law provides full support for XMLaw, which means it is possible to specify non-deterministic state-based machines, notions of commitments through norms, time sensitive laws, and execution of java code.
- Flexibility – The use of indirect communication through events in combination with the component-based module makes it possible to add new functionality with little difficulty. However, it is known that event-based communication may lead to software that is harder to understand and debug due to the implicit nature of communication. We have tried to deal with this drawback by systematically building test cases, performing code inspections and writing exhaustive documentation.

5 Using the Middleware

To use the middleware it is necessary perform at least four tasks. First, one must write the interaction laws using XMLaw language. Second, the mediator has to be started by execution of the script files provided with M-Law. Then, one has to inform the mediator about the existence of the new Law (XMLaw file). Finally, the application agents may be started. The idea is to use the middleware as an environmental service. In other words, the middleware should be available for the agent developers once they have their agents running on the environment.

Regarding the development of the application agents, agent developers may want to extend the agent class provided by the client API of M-Law. This class provides methods for sending and receiving of messages and methods for direct communication with the mediator, once the mediator can provide useful information about the current status of interaction, such as which scenes are running and how many agents are interacting. In fact, the class LawFacade provides methods for direct communication with the mediator, and agent class provides methods for sending and receiving messages. Figure 6 shows the details of those classes.

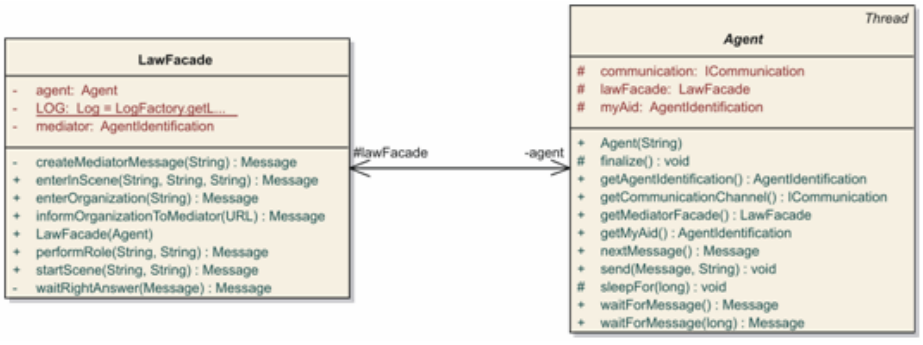

Fig. 6. Client API: LawFacade and Agent classes

We encourage agent developers to use the agent class either by inheritance or delegation. Yet, developers are free to build their agents using any architecture or technology. The only requirement is that the agent should know both how to "speak" FIPA-Agent Communication Language[2] and which messages the mediator expects.

5.1 Case Study (CB)

The Central Bank of Brazil regulates and supervises the national financial system. This experiment is running based on the SELIC system requirements. SELIC is the central depository of securities issued by the National Treasury and the Central Bank of Brazil. It also settles outright and rep transactions with these securities. Besides the National Treasury and the Central Bank of Brazil, commercial banks, investment banks, savings banks, dealers and brokers, clearing operators, mutual investment funds and many other institutions that integrate the financial system participate in SELIC as holders of custody accounts. In December 2004, the system was composed of 4,900 participants (or agents).

SELIC system is clearly a system that has a central entity (Central Bank) that mediates and controls the interaction among the other entities. We have, then, specified the laws that the institutions must follow using XMLaw, and we have used M-Law as a mediator that control the interactions. We have implemented a prototype of a subset of the actual SELIC system for doing this experiment. The experiment was performed with 10 agents representing different financial institutions and 1 mediator agent (the MLaw).

There are several requirements that influence the interaction of financial institutions in a committed operation, as the several types of messages that could be sent and the several behaviors that should be implemented according to the messages specified, including norms and constraints. Below (Figure 7), an interaction scenario is taken from SELIC. This interaction scenario contains three entities: two financial institutions and the SELIC.

[2] FIPA is the organization that establishes specifications for agents. http://www.fipa.org/

In this scenario, a financial institution A (FI A) needs to sell bonds to the financial institution B (FI B). Below (Code Fragment 3), we illustrate the description of this basic negotiation protocol in XMLaw. We called it Negotiation. The specification provides the details regarding the expected attributes of the messages. Furthermore, there in the transition st1 there is a verification step implemented within a constraint.

This example shows how the laws can be specified and then provided to the environment through the M-Law in order to effectively monitor and enforce agents' behavior. On the side of the developers of the domain agents, it is necessary to extend the class Agent provided with the client API (Figure 6) and then use its methods to communicate.

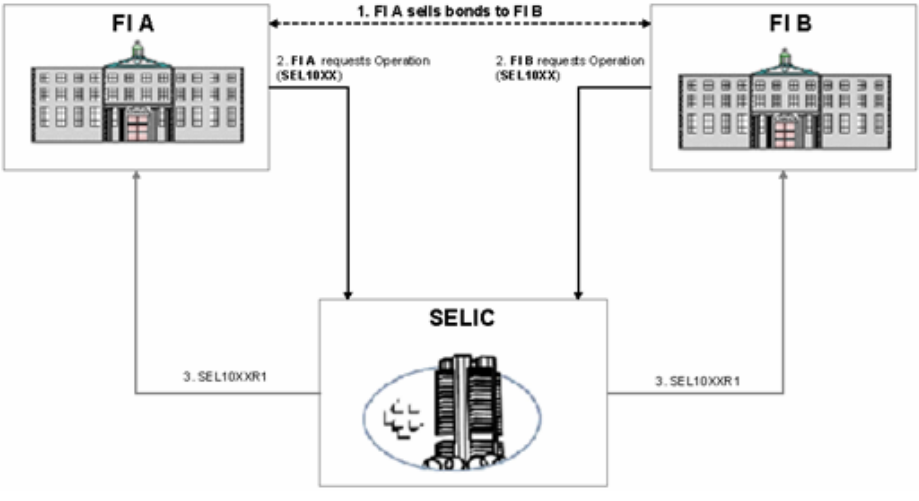

Fig. 7. SELIC example

The preliminary results have shown that M-Law and XMLaw have brought some consequences such as:

- Transparency of the process – Before using our governance solution, the system had all the laws hard-coded into the source code. With XMLaw the laws are specified in a purpose specific language which brings the specification to a higher level of abstraction and then decreases the distance between the requirements and the implementation. M-Law has a crucial role in this scenario, since it monitors and interprets the laws.
- Better support for rules customization/configuration – Changing a law with XMLaw and M-Law is a matter of changing the XMLaw specification, i.e., there is no need to go into source code of the application.

```
<Scene id="Negotiation" time-to-live="infinity">
 <Protocol id="negotiation-protocol">
  <Messages>
   <Message id="startMsg">...</Message>
   <Message id="request">
    <Content>
     <Entry key="CodMsg" value="SEL1054"/>
     <Entry key="TpCompr" value="02" />
     <Entry key="NOPRET"/>
    </Content>
   <Message id="inform">
    <Content>
     <Entry key="CodMsg" value="SEL1054"/>
     <Entry key="TpCompr" value="04" />
     <Entry key="NOPRET"/>
    </Content>
   </Message>
  </Messages>
  <States>
   <State id="s1" type="initial"/>
   <State id="s2" type="execution"/>
   <State id="s3" type="execution"/>
   <State id="s4" type="success"/>
  </States>
  <Transitions>
   <Transition id="start" from="s1" to="s2"
       ref="startMsg" event-type="message_arrival" />
   <Transition id="st1" from="s2" to="s3" ref="request"
       event-type="message_arrival"
    <Constraint
      class="selic.constraint.ConditionNOPRET"
      semantics="NOPRETEmpty" />
   </Transition>
   <Transition id="st2" from="s3" to="s4" ref="inform"
      event-type="message_arrival"/>
  </Transitions>
 </Protocol> ...
</Scene>
```

Code 3. Negotiation structure definition

6 Related Work

It is possible to cite at least two important research projects in which the goals are in some sense similar to the goals of the work presented here. The first approach is proposed by Esteva [7]. He uses a set of concepts that have points of intersection with those used in XMLaw, in fact XMLaw has borrowed some of the ideas of Esteva and proposed new ideas. For example, both Esteva scenes and protocol elements specify the interaction protocol using a global view of the interaction. The time aspect is represented in the Esteva approach as timeouts. Timeouts allow activating transitions after a given number of time units passed since a state was reached. On the other hand, due to our event model, the clock element proposed in XMLaw can both

activate and deactivate not only transitions, but also other clocks and norms. Connecting clocks to norms allows a more expressive normative behavior; norms become time sensitive elements.

Furthermore, XMLaw also includes the concept of actions, which allows execution of java code in response to some interaction situation. From the implementation point of view, Esteva does not provide the internal details of its framework (ISLANDER); but the general architecture proposes to use a set of mediators instead of using only one mediator. One consequence of this solution is that once a law is specified as a global view, all the mediators must constantly synchronize their internal states to keep them consistent. It means that for every message sent by an application agent, messages are broadcast among the mediators to synchronize the state.

From the point of view of integration with other existent solutions, Islander allows the use of JADE as communication layer; but only Jade is allowed and there is no support for extension on this point. Furthermore, there is no indication about the possibility of integration between Islander and different approaches, such as integration tests and criticality analysis.

Minsky [15] proposes a coordination and control mechanism called law governed interaction (LGI). This mechanism is based in two basic principles: the local nature of the LGI laws and the decentralization of law enforcement. The local nature of LGI laws means that a law can regulate explicitly only local events at individual home agents, where home agent is the agent being regulated by the laws; the ruling for an event e can depend only on e itself, and on the local home agent's context; and the ruling for an event can mandate only local operations to be carried out at the home agent. On the other hand, the decentralization of law enforcement is an architectural decision argued as necessary for achieving scalability.

However, when it is necessary to have a global view of the interactions, the decentralized enforcement demands state consistency protocols, which may not be scalable. Furthermore, it provides a language to specify laws and it is concerned with architectural decisions to achieve a high degree of robustness. In contrast, M-Law uses XMLaw, which provides an explicit conceptual model and focuses on different concepts, such as Scenes, Norms and Clocks. In other words, in our opinion, LGI design aimed primarily at decentralization and XMLaw design aimed primarily at expressivity and at possibilities for specialization [4]. A current limitation of XMLaw is the centralization of the mediator.

A promising direction is to investigate how XMLaw specification could be compiled into decentralized LGI mediators. In this way, LGI could be viewed by having the basic foundation to build higher level elements, such the ones in XMLaw. Moreover, by using M-Law, it is possible to extend the framework hotspots and introduce new components, which represent concepts in the conceptual model; and change the communication mechanism.

7 Conclusions

Governance is required in open MAS Environments. A trustable Environment should offer a service to guarantee that the rules of interaction are obeyed. In this paper, we proposed to include a governance service within any open MAS Environment. This

paper has presented some of the main ideas behind research into governance of agent systems. Then, from a Software Engineering perspective, we have proposed a middleware that allows the development of law-regulated systems. We have presented the main design goals of the middleware, such as flexibility and integration with other platforms. We believe that this middleware is an enhancement of the current state-of-the-art of Software Engineering for Governance in the sense that it supports a language that expresses the main concepts of governance with a good level of expressivity, and also due to its design concerns, which make use of techniques such as frameworks and components. Current implementation relies on a centralized mediator, which is a limitation on scalability. We are currently studying the design of a future distributed version, inspired from LGI decentralized mediators.

We have shown the use of the middleware on an example inspired from a real application of the central bank of Brazil. We hope that this example could illustrate some benefits brought by the use of M-Law, and more generally speaking, the merits of MAS governance for future applications.

Acknowledgments. This work was partially funded by CNPq through the ESSMA Project (552068/2002-0) and through individual grants. The work was also supported by CAPES, in the CAPES/Cofecub International Cooperation Program, through the EMACA Project (0981-04-4).

References

1. Arunachalam, R., Sadeh, N., Eriksson, J., Finne, N., Janson, S.: The Supply Chain Management Game for the Trading Agent Competition 2004. CMU-CS-04-107, (2004)
2. Bellifemine, F., Poggi, A., Rimassa G.: JADE: a FIPA2000 Compliant Agent Development Environment. In: Fifth International Conference on Autonomous Agents (2001)
3. Carvalho, G., Brandão, A., Paes, R., Lucena, C.: Interaction Laws Verification Using Knowledge-based Reasoning. In: Workshop on Agent-Oriented Information Systems (AOIS-2006) at AAMAS 2006.
4. Carvalho, G., Lucena, C., Paes, R., Briot, J.: Refinement operators to facilitate the reuse of interaction laws in open multi-agent systems. In Proceedings of the 2006 International Workshop on Software Engineering For Large-Scale Multi-Agent Systems (2006)
5. Carvalho G., Almeida H., Gatti, M., Vinicius, G., Paes, R., Perkusich, A., Lucena, C.: Dynamic Law Evolution in Governance Mechanisms for Open Multi-Agent Systems. Second Workshop on Software Engineering for Agent-oriented Systems (2006)
6. Collins, J., Arunachala, R., Sadeh, N., Eriksson, J., Finne, N., Janson, S.: The Supply Chain Management Game for the 2005 Trading Agent Competition. CMU-ISRI-04-139. http://www.sics.se/tac/tac05scmspec_v157.pdf (2005)
7. Esteva, M.: Electronic institutions: from specification to development, Ph.D. thesis, Institut d'Investigació en Intelligència Artificial, Catalonia - Spain. (2003)
8. Fayad, M., Schmidt, D., Johnson, R.E.: Building application frameworks: object-oriented foundations of framework design. John Wiley & Sons (1999)
9. Ferber, J., Gutknecht, O., Michel, F. From Agents to Organizations: an Organizational View of Multi-Agent Systems. In Agent-Oriented Software Engineering (AOSE) IV, P. Giorgini, Jörg Müller, James Odell, eds, Melbourne, July 2003, LNCS 2935, pp. 214-230 (2004)

10. Gamma, E., Johnson, R., Helm, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, (1995)
11. Gatti M., Lucena C., Briot J.P.: On Fault Tolerance in Law-Governed Multi-Agent Systems. 5th International Workshop on Software Engineering for Large-scale Multi-Agent Systems (SELMAS) at ICSE (2006)
12. Guessoum, Z., Faci, N., Briot, J-P.: Adaptive Replication of Large-Scale Multi-Agent Systems - Towards a Fault-Tolerant Multi-Agent Platform. In: International Workshop on Software Engineering for Large-scale Multi-Agent Systems - SELMAS at ICSE (2005)
13. Huhns, M.N., Stephens, L.M.: Multi-agent Systems and Societies of Agents. G. Weiss (ed.), Multi-agent Systems, ISBN 0-262-23203-0, MIT press (1999)
14. Lindermann, G., Ossowski, S., Padget, J., Vázquez Salceda, J.: International Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems (ANIREM 2005), http://platon.escet.urjc.es/ANIREM2005/ accessed in December, 2006.
15. Minsky, N.H., Ungureanu V.: Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems, ACMTrans. Software Engineering Methodology 9(3) 273–305. (2000)
16. Odell, J., Parunak, H.V.D., Fleischer, M., Breuckner, S.: Modeling Agents and their Environment. Agent-Oriented Software Engineering III, Giunchiglia, F., Odell, J., Weiss, G. (eds.) Lecture Notes in Computer Science, Vol. 2585. Spriner-Verlag, Berlin Heidelberg New York (2002)
17. Paes, R.B., Carvalho G.R., Lucena, C.J.P., Alencar, P.S.C., Almeida H.O., Silva, V.T.: Specifying Laws in Open Multi-Agent Systems. In: Agents, Norms and Institutions for Regulated Multi-agent Systems (ANIREM), AAMAS2005. (2005)
18. Paes, R.B, Lucena, C.J.P, Alencar, P.S.C.: A Mechanism for Governing Agent Interaction in Open Multi-Agent Systems. http://wiki.les.inf.puc-rio.br/index.php/Publications (2005)
19. Paes, R.B., Gatti, M.A.C., Carvalho, G.R., Rodrigues, L.F.C., Lucena, C.J.P.: A Middleware for Governance in Open Multi-Agent Systems. Technical Report 33/06, PUC-Rio, 14 p. (2006)
20. Rodrigues, L., Carvalho, G., Paes, R., Lucena, C.: Towards an Integration Test Architecture for Open MAS. In: Software Engineering for Agent-oriented Systems - SEAS05 , (2005)
21. Rubino, R., Omicini, A., Denti, E.: Computational Institutions for Modeling Norm-Regulated {MAS}: An Approach Based on Coordination Artifacts. In: Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems, Springer, vol. 3913, 127—141 (2006)
22. Sadeh, N., Arunachalam, R., Eriksson, J., Finne, N., Janson, S.: TAC-03: a supply-chain trading competition, AI Mag. 24 (1) 92–94 (2003)
23. Schumacher, M., Ossowski, S.: The Governing Environment, E4MAS (2005)
24. Weyns, D., Parunak, H.V.D., Michel, F., Holvoet, T., Ferber, J.: Environments for Multiagent Systems: State-of-the-Art and Research Challenges. Lecture Notes in Computer Science Vol. 3374, Springer (2004)
25. Weyns, D., Michel, F., Parunak, H.V.D.: The Third International Workshop on Environments for Multi-Agent Systems - http://www.cs.kuleuven.ac.be/~distrinet/events/e4mas/2006/print.php - accessed in October (2006)
26. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. In: Autonomous Agents and Multi-Agent Systems 14(1), 5-30 (2007)