ISSN 0103-9741

Monografias em Ciência da Computação
n° 33/06

A Middleware for Governance in Open Multi-Agent Systems

Rodrigo de Barros Paes
Maíra Athanazio de Cerqueira Gatti
Gustavo Robichez de Carvalho
Luiz Fernando Chagas Rodrigues
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 33/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Agosto, 2006

A Middleware for Governance in Open Multi-Agent Systems *

Rodrigo de Barros Paes, Maíra Athanázio de Cerqueira Gatti,

Gustavo Robichez de Carvalho, Luiz Fernando Chagas Rodrigues,

Carlos José Pereira de Lucena

{rbp,mgatti,guga,lfrodrigues,lucena}@inf.puc-rio.br

Abstract. Interacting entities in many distributed systems have to follow certain protocols and interaction rules. This paper presents a middleware to regulate agent interactions in systems characterized mainly by the need of having control over the actions that an agent can perform. The solution is based on the ideas proposed in the field of governance of agent systems, where governance can be defined as the set of approaches that aim to establish and enforce some structure, set of norms or conventions, that articulate or restrain interactions in order to make agents more effective in attaining their goals or more predictable.

Keywords: Governance, Agents, Protocols, Electronic Institutions, Multi-Agent Systems, Open Systems.

Resumo. Em muitos sistemas distribuídos as entidades que interagem precisam seguir protocolos e regras de interação. Neste artigo apresenta-se um middleware para regular interações entre agentes em sistemas caracterizados principalmente pela necessidade de algum grau de controle sobre as ações que os agentes realizam. A solução é baseada em idéias de governança de sistemas multi-agentes, onde governança pode ser definido como o conjunto de abordagens que objetivam estabelecer e verificar um conjunto de regras que restringem a interação dos agentes.

Palavras-chave: Governança, Agentes, Protocolos, Instituições Eletrônicas, Sistemas Multi-Agentes, Sistemas Abertos.

---

* Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil (e agência de fomento e o número do processo, se aplicável). (Em Inglês: This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil)

Responsável por publicações (ou In charge for publications, se o texto for em inglês):

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

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. However, there are in general some rules that must be followed by all agents while interacting within the system and all agent developers should understand the consequences of rule violation. An Internet-based market system is an example of such system. In these systems, agents must comply with rules such as "if a buyer buys some good, it must pay for it in a period of two days," "sellers must answer buyer requests in at most 1 minute." Then, the problems are how to specify these rules in a way that they can be easily understood by agent developers, how to verify if the agents are effectively following the specified rules, and, in some cases, how to prevent rule violation. A new, exciting research area has attracted attention by dealing with just such problems. Governance for open multi-agent systems can be defined as the set of approaches that aim to establish and enforce some structure, set of norms or conventions that articulate or restrain interactions in order to make agents more effective in attaining their goals or more predictable.

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. This paper deals with the above mentioned problem through a design approach and application of a middleware for governance in multi-agent systems. The middleware can be used in conjunction with a specific agent platform (such as JADE [1]), 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. Three examples are described in this paper: one shows the initial steps for the construction of a prototype of a negotiation system for the Central Bank of Brazil; the second shows how the middleware can be used to support integration tests; and finally, the third example deals with the identification and monitoring of agent criticality.

This paper is organized as follows. Section II introduces Governance vis-à-vis the discipline of Software Engineering. Section III presents the middleware for governance developed as part of the present work. Section IV discusses how the middleware can be effectively used. Section V describes three experiments that have been performed by using the middleware. Section VI presents some related work and highlights the contributions of this paper. Finally, Section VII presents some final considerations and conclusions about this work..

2 Governance in Multi-Agent Systems: general overview

The problem of establishing governance in multi-agent systems has been addressed from different point of views by communities such as Social Science, Artificial Intelligence and Software Engineering. The social scientists are interested in using agents to model humans and human societies. The main idea is to develop organizational models and model them as computational models in order to conduct experiments and get the feedback from the experiments. This feedback is used to validate the organizational models. Social scientists need governance because governance is considered as a key element to human social intelligence, and therefore, governance embodies the social knowledge and culture of a society through its norms, protocols and laws. Therefore, it is natural that organizational models contain notions of governance. Artificial Intelligence is particularly concerned with the reuse of social models to develop smarter systems. The research in the area builds computational models to represent social models, and to develop a set of theories, models of reasoning, plans and even emotions to build more adaptable and evolvable software systems. In the above context, the role of Software Engineering is to systematize the construction of smarter systems in a way that the results will lead to systems that are not just smarter, but also are easier to maintain, to evolve and that are more dependable.

Thus, there is a clear complementariness among the three referred areas, and the role of Software Engineering is crucial for establishing governance as a mature technology by providing methodologies and tools to build industrial quality systems. The work presented in this paper is concerned with providing a design approach and tools to enable agent developers to build better quality open multi-agent systems through a middleware that supports a strong notion of governance.

3 M-Law: The Middleware

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 [2].

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 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. The architecture is shown in **Figure 1**. Agents may be running either at the same host or at different hosts. The mediator can also run at any host — i.e., there is no restriction to where any agent is deployed.

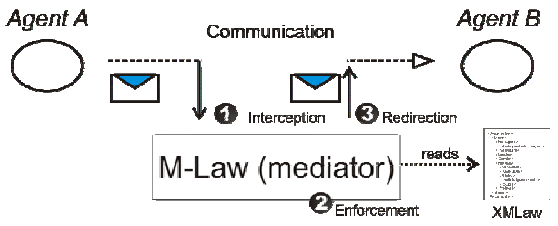

Figure 1 - Architecture

M-Law was built to support law specification using a specific purpose language named XMLaw [3]. XMLaw allows the system developer to specify how the interactions among agents of the whole system should occur. This language was chosen because it was designed to express ideas considered to be very important while regulating interactions, such as social norms and temporal constraints. In this way, before introducing the framework we are going to briefly present the main features of XMLaw. If the reader needs further information, please read [3].

3.1 XMLaw

The main elements of XMLaw are scenes, protocols, norms, clocks, actions and constraints. To better understand the rationale behind those elements, consider a standard shopping scenario. First, a buyer arrives at the shop, searches for the goods he wants to buy, negotiates the price, and finally makes the payment. In this scenario, we can say we have at least four scenes: arrival, searching, negotiation and payment.

The main purpose of the scene element is to compose elements such as protocols, norms, clocks, actions and constraints in a modular way, defining specific interaction contexts (**Figure 2**). Each scene has its own peculiarities, such as its own set of norms and its own interaction protocol. Scenes also specify which agents may participate in a scene and the moment in which they may participate.

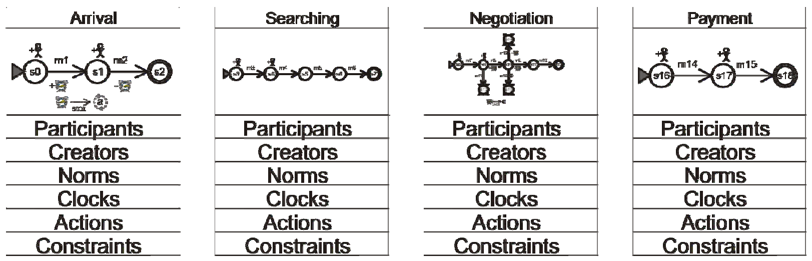

Figure 2 - Scenes

Protocols, or interaction protocols, are non-deterministic state-based machines where transitions are fired by event occurrences. Events form the basic communication model between the elements of XMLaw. We define XMLaw elements to be every element used in XMLaw, and they are: scenes, protocols, norms, clocks, actions and constraints. Elements in XMLaw can sense and generate events. For example, transitions may sense message arrival events and, once it perceives one, it may generate a transition activation event; this transition activation event may be sensed by a norm, which can generate other events and so on. This chain of causes and consequences makes flexible specification possible. Since XMLaw is just a specification language, in fact the control of generation and propagation of events is done by the mediator; however, whoever writes XMLaw specifications must be aware of this dynamic in order to specify the laws. For example, the following XMLaw code fragment shows a clock that will be activated when the transition "t1" is activated, or in other words, when the event "transition_activation" generated by the transition "t1" is sensed by the clock "c1".

```
<Protocol id="contract-net">
        ...
        <Transitions>
                <Transition id="t1" from="s0" to="s1" ref="m1" event-
type="message_arrival" />
        </Transitions>
</Protocol>
<Clock id="c1" type="periodic" tick-period="2000">
```

```
                        <Activations>
                              <Element          ref="t1"          event-
type="transition_activation" />
                        </Activations>
                        ...
</Clock>
```

Code Fragment 1 – A transition that activates a clock

Clocks are elements that permit specification of temporal aspects in the law. There are basically two types of clock: one generates just one event after a certain amount of time has elapsed, and the other cyclically generates events at a specific interval of time. Norms are viewed as commitments given a priori or made a posteriori by agents. For example, when a buyer that participates in an auction wins a bid, he acquires an obligation to pay for the good. Norms can be defined as obligations, permissions or prohibitions. Lastly, XMLaw provides two hook mechanisms where java code can be used: constraints and actions. Constraints allow using java code to perform complex validations on the messages exchanged by the agents. Actions allow java coding to be activated by any event of the underlying communication model. This means that actions are an effective way to extend the basic XMLaw functionality.

3.2 M-Law

M-Law middleware has to provide the means to effectively support XMLaw and its evolutions. A big picture of M-Law is showed in **Figure 3**. This figure shows the four main modules of the middleware. 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.

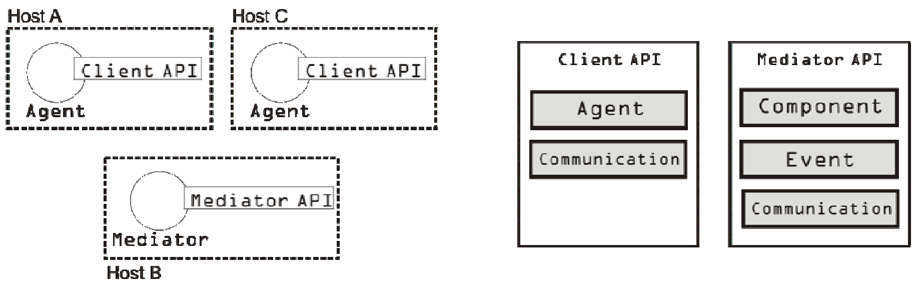

Figure 3 – M-Law: Big Picture

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 Section 5 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 [4] 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. Components are the set of classes placed in the mediator that implements the behavior of the XMLaw language elements; for example, Figure 4 shows the scene element in XMLaw and its set of classes that implements its behavior. In Figure 4, the scene Element in XMLaw is mapped to a descriptor and execution hierarchies. 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.

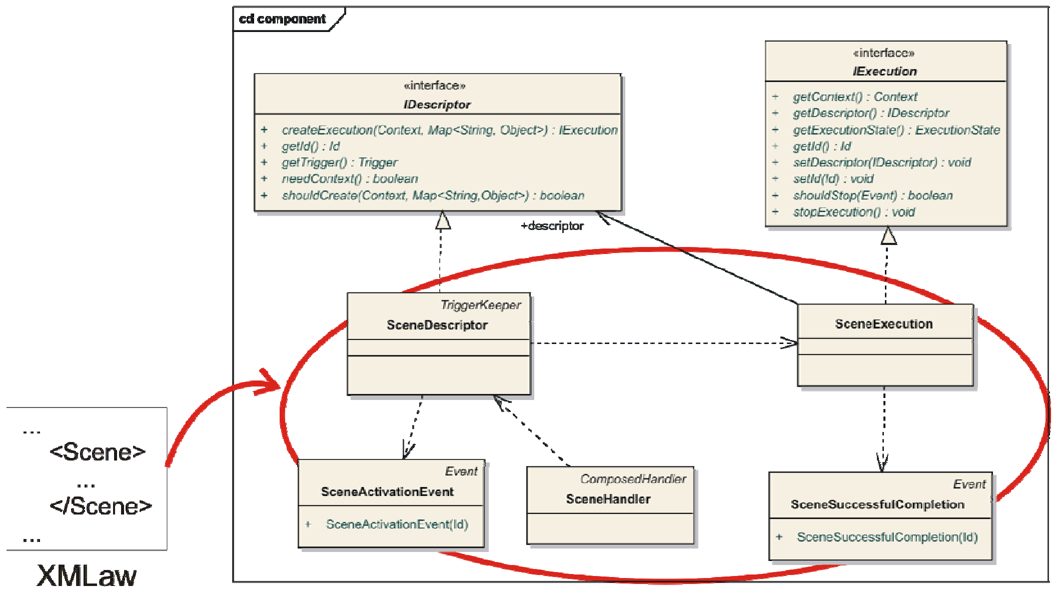
cd component
«interface»
IDescriptor
+ createExecution(Context, Map<String, Object>) : IExecution
+ getId() : Id
+ getTrigger() : Trigger
+ needContext() : boolean
+ shouldCreate(Context, Map<String,Object>) : boolean
«interface»
IExecution
+ getContext() : Context
+ getDescriptor() : IDescriptor
+ getExecutionState() : ExecutionState
+ getId() : Id
+ setDescriptor(IDescriptor) : void
+ setId(Id) : void
+ shouldStop(Event) : boolean
+ stopExecution() : void
+descriptor
TriggerKeeper
SceneDescriptor
SceneExecution
Event
SceneActivationEvent
+ SceneActivationEvent(Id)
ComposedHandler
SceneHandler
Event
SceneSuccessfulCompletion
+ SceneSuccessfulCompletion(Id)
...
<Scene>
...
</Scene>
...
XMLaw

Figure 4 – Scene component

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 4**, 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. We are working on alternative, more decentralized, solutions to this problem. We are performing experiments on both a network of decentralized mediators, such as LGI [5], and hierarchal organized mediators, such as Internet DNS. On the other hand, although a centralized solution poses scalability questions, it allows the specification of global laws with no need for state synchronization. Furthermore, temporal problems are also avoided, once there is just one host controlling clocks.

- 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.

4 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.

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 5** shows the details of those classes.

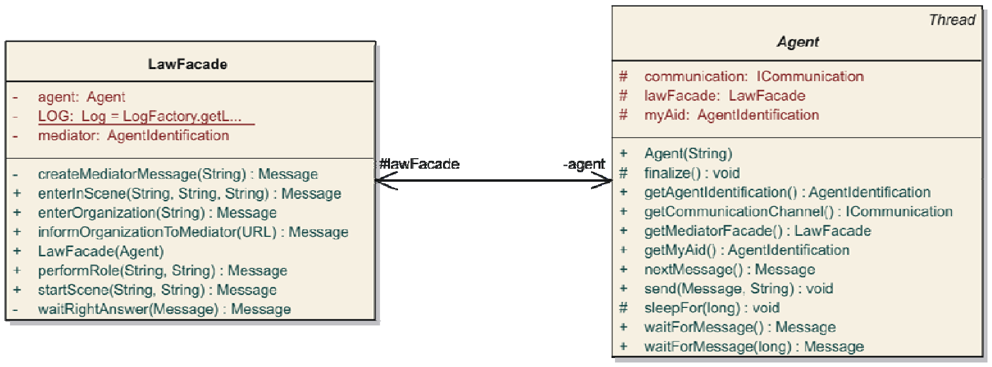
LawFacade
- agent: Agent
- LOG: Log = LogFactory.getL...
- mediator: AgentIdentification
- createMediatorMessage(String) : Message
+ enterInScene(String, String, String) : Message
+ enterOrganization(String) : Message
+ informOrganizationToMediator(URL) : Message
+ LawFacade(Agent)
+ performRole(String, String) : Message
+ startScene(String, String) : Message
- waitRightAnswer(Message) : Message
#lawFacade
-agent
Thread
Agent
# communication: ICommunication
# lawFacade: LawFacade
# myAid: AgentIdentification
+ Agent(String)
# finalize() : void
+ getAgentIdentification() : AgentIdentification
+ getCommunicationChannel() : ICommunication
+ getMediatorFacade() : LawFacade
+ getMyAid() : AgentIdentification
+ nextMessage() : Message
+ send(Message, String) : void
# sleepFor(long) : void
+ waitForMessage() : Message
+ waitForMessage(long) : Message

Figure 5 – Client API: LawFacade and Agent classes

We encourage agent developers to use the agent class either by inheritance or delegation. However, 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[1] and which messages the mediator expects.

5 Experience report

This section shows how M-Law has been used and it is useful to illustrate both the relevance of a governance framework and the flexibility of the M-Law design, which can be observed by its applications in different domains.

5.1 Experience 1: Central Bank of Brazil

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

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

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. 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.

5.2 Integration Tests

This work [6] was concerned with how to write test cases and receive reports on their results. The system is considered to be composed of several distributed subsystems, and each subsystem is viewed as an agent. In this context, XMLaw was used to specify and the M-Law to monitor the system behavior. M-Law was integrated with other software that provides the ability to write test cases and to generate test reports. Results of this experiment show the ability of M-Law to integrate itself with other software solutions.

5.3 Criticality

The Agent Replication Technique, which is the act of creating one or more duplicates of one or more agents in a multi-agent system, is said to be a way to achieve fault tolerance in Multi-agent Systems. This experiment [7] sought to increase the dependability of multi-agent systems as much as possible in order to guarantee a high degree of reliability of such systems; a platform that integrated these two approaches (dependability achieved by fault tolerance and by governance of agents' interaction) would be an efficient way of doing it. So, it is possible to have a platform with a high degree of dependability and a law-governed approach to improve the criticality analysis that defines the number of agent's replicas and at the same time could provide the ability of monitoring and enforcing the behavior of agents through the enforcement of laws. The goal is to discover how the elements of the XMLaw could improve the agent criticality analysis that is done by DimaX [8] (an agent replication framework).

Thus, it was proposed to use new elements — Role and Criticality Analysis — that help in specifying the attributes concerning the agent criticality during its interaction with other agents.. The new elements have an impact on both XMLaw language and in the component module of M-law. New classes were added and integrated with the component module.

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 [9]. He uses a set of concepts that have points of intersection with those used in XMLaw. 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 [5] 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. Also, it does not provide information about either the integration of LGI with existent solutions or the support provided for extension of the LGI middleware. It is possible to think of LGI by having more elementary elements to specify and enforce interactions. It may be possible to combine and compose the elements provide by LGI to construct more sophisticated and higher level elements such norms. 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 the 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

This paper has presented some of the main ideas behind research into governance of agent systems, identifying the role of Software Engineering and its relationships with other related subjects such as Social Science and Artificial Intelligence. 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. The 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, and also due to its design concerns, which makes use of techniques such as frameworks and components.

We have shown the use of the middleware in three different contexts, where the central bank example shows the middleware in a complex real world scenario, highlighting the benefits brought by the use of M-Law. The second example, concerned with integration tests, shows the middleware being used in conjunction with other software solutions. And, the criticality example has explored the evolutionary ability of both XMLaw language and the M-Law in the sense that a criticality component was added.

To conclude, the expected contribution of this paper is to report a solution that preliminary experiments have led us to believe can be applied to a broad range of important problems present in software development.

Referências Bibliográficas

[1] F. Bellifemine, A. Poggi, G. Rimassa, "JADE: a FIPA2000 Compliant Agent Development Environment," Fifth International Conference on Autonomous Agents, 2001.

[2] M. Fayad, D. Schmidt, R. E. Johnson, "Building application frameworks: object-oriented foundations of framework design," John Wiley & Sons, 1999.

[3] R. Paes, G. Carvalho, C. Lucena, P. Alencar, H. Almeida, V. Silva, "Specifying Laws in Open Multi-Agent Systems," Agents, Norms and Institutions for Regulated Multiagent Systems, 2005.

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: elements of reusable object-oriented software," Addison-Wesley, 1995.

[5] N. H. Minsky, V. Ungureanu, "Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems." ACM Trans. Softw. Eng. Methodol., 2000.

[6] L. Rodrigues, G. Carvalho, R. Paes, C. Lucena, "Towards an Integration Test Architecture for Open MAS," Software Engineering for Agent-oriented Systems (SEAS05), 2005.

[7] M. Gatti, C. Lucena, J. Briot, "On Fault Tolerance in Law-Governed Multi-Agent Systems," International Workshop on Software Engineering for Large-scale Multi-Agent Systems (SELMAS) at ICSE 2006

[8] Z. Guessoum, N. Faci, J. Briot, "Adaptive Replication of Large-Scale Multi-Agent Systems - Towards a Fault-Tolerant Multi-Agent Platform," International

Workshop on Software Engineering for Large-scale Multi-Agent Systems (SELMAS) at ICSE 2005.

[9]	Marc Esteva. Electronic Institutions: from specification to development. PhD thesis, Institut d'Investigació en Intel.ligència Artificial, Catalonia - Spain, October 2003.