Enforcing Interaction Protocols in Multi-Agent Systems[*]

Rodrigo de Barros Paes*
rbp@les.inf.puc-rio.br

Hyggo Oliveira de Almeida*
Universidade Federal de Campina Grande, UFCG, Brasil
hyggo@dee.ufcg.edu.br

Carlos J. P. de Lucena
lucena@inf.puc-rio.br

Paulo S. C. Alencar
University of Waterloo, Canadá
palencar@csg.uwaterloo.ca

PUC-RioInf.MCC 09/04 March, 2004

Abstract: Numerous multi-agent approaches have been used to develop large and complex systems in many application areas. Interactions among agents, which are commonly ruled by interaction protocols that ensure the correct system execution, constitute one of the main features of these approaches. However, in open, dynamic, and heterogeneous systems, where there are no guarantees about the trustfulness of the agents, mechanisms should be provided to ensure protocol fulfillment. In this paper we propose a conceptual framework and its implementation to enforce secure interaction protocols in multi-agent systems. We use the "organization law" abstraction as a generalization of other concepts found in the literature, such as constraints, policies, norms and protocols. Finally, the proposed conceptual framework allows laws to be defined as declarative specifications..

Keywords: multi-agent systems, organizations, interaction protocols, open systems, laws

Resumo: A abordagem multi-agentes vem sendo utilizada para a concepção de vários tipos de sistemas considerados complexos. Esta abordagem tem como uma de suas principais características as interações entre os agentes, sendo estas regidas por protocolos de interação que determinam o funcionamento correto do sistema. Porém, em sistemas abertos, dinâmicos e heterogêneos, onde não é possível garantir a confiabilidade dos agentes que o compõem, é necessária uma forma de garantir que estes protocolos sejam cumpridos. Neste trabalho propõe-se um arcabouço conceitual para garantir a confiabilidade nas interações entre os agentes de um sistema multi-agentes. Para isso, utiliza-se a abstração de "leis sobre organizações" que, como será apresentado neste artigo, pode ser visto como uma generalização dos conceitos de restrições, políticas, normas e protocolos de interação entre agentes. Além disso, apresenta-se um arcabouço de software implementado em Java que implementa as especificações do arcabouço conceitual e permite que estas leis sejam especificadas de forma declarativa.

Palavras-chave: sistemas multi-agentes, organizações, protocolos de interação, leis

[*] Sponsored by CAPES/Brasil.

1 Introduction

Multiagent approaches have been used as abstractions to develop large and complex systems in many application areas [1]. The interaction among agents, which is one of the main features of these approaches, is commonly ruled by protocols that determine the correct system execution. In addition, policies [2], protocols [3], norms [4] and constraints [5, 6] have been used as abstractions to define the interaction rules related to multiagent system interactions. On the other hand, the Organization concept also has been used as abstraction to guide the software development in agent-based systems. According to this concept, a system may be considered a set of agent organizations that interact to reach their goals and obey laws that restrict their interactions. In open, dynamic, and heterogeneous systems, where there are no guarantees about the trustfullness of the agents, mechanisms should be provided to ensure protocol fulfillment. In this paper we propose a conceptual framework and its implementation to enforce secure interaction protocols in multi-agent systems. We use the "organization law" abstraction as a generalization of other concepts found in the literature, such as constraints, policies, norms and protocols. The conceptual framework, which is called FROG, provides support for interaction-related law specification and enforcement in open and heterogeneous multiagent systems, where the laws are defined in a declarative way. Finally, we also present a framework called JFROG that is a Java implementation of the conceptual framework. Because the multiagent interaction laws are defined declaratively, their definition will be independent of the actual software implementation infrastructure. The rest of paper is organized as follows. In Section 2 we present law abstractions based on the notion of organizations. In Section 3 we introduce FROG, the conceptual framework that enforces secure interactions in multi-agent system. In Section 4 we present JFROG software framework, the Java-based framework implementation. At last, we describe work related to our contributions in Section 5, and we conclude the paper in Section 6 with a summary of our results and comments about future work.

2 Organizational Laws

The notion of organization has been used as an abstraction by many researchers to guide the development of complex systems [7, 8]. In this line of research, an organization is defined as an entity that has the following features [9, 10, 11, 12, 13, 14, 15]: it has a purpose for its creation, i.e., it is not just a grouping of agents; it is composed by agents and other organizations that must play roles defined by the organization; it has a structure that may change during its life time, where this structure represents the elements that compose the organization and the way in which such elements interact with each other; it specifies behavioral rules, called laws, that must be fulfilled by the members that compose such organization. Among the organization features mentioned previously, we have chosen to focus mainly on the last one, the feature related to interaction laws. A law can be seen as a set of rules that govern a specific kind of activity and that is imposed by an authority [16]. According to this definition, a law can be seen as a generalization of other concepts commonly used by other researchers to represent constraint enforcement in multi-agent systems. Examples of such concepts include:

- **Protocol** - a set of possible actions, possibly ordered, that can be executed by the participants in an interaction, i.e., a set of interaction constraints that must hold among the participants.

- **Policy** - rights and duties that the system participants have on some resources or on their interactions. A duty can be seen as a constraint that has to be fulfilled. In the same way, a right of an entity can be considered equivalent to the negation of this right to all the other entities. In this way, policies can also be represented by constraints.

- **Norm** - rules that must be fulfilled by the entities affected by these rules. Therefore, norms can also be seen as constraints.

As protocols, policies and norms, the laws can be represented as a set of constraints. In order to represent in a general way the constraint-based enforcement of secure interactions among agent that compose a multi-agent organization, in this work we adopt the concept of laws. In this way, we use only one abstraction to represent many apparently unrelated concepts found in literature.

3 FROG: The conceptual framework

FROG is a conceptual framework that provides support for law specification and enforcement of secure interaction protocols in open and heterogeneous multi-agent systems. This framework defines a set of specifications that make possible to have compatibility among the framework instantiations even when different languages are used. FROG uses the agent abstraction as part of its specification.

Agents are able to communicate with other software components that represent agents using a high level language, probably based on speech acts [17]. Although these are not issues we focus on in this paper, agents can have some degree of autonomy and, in some cases, they may also have learning and mobility capabilities. To ensure law enforcement related to interaction protocols in multi-agent systems, the following elements are defined in FROG:

- Organization.def - This element should describe multi-agent organizations. Therefore, it must define agent roles, organization laws and an organization hierarchy. The notion of an organization hierarchy concept is especially useful when we are developing enterprise software systems. A typical enterprise is composed by many administrative units and defines some laws that must be followed by these units. Furthermore, these administrative units may define their own policies that are subordinate to the enterprise policies. In addition, the elements that belong to an administrative unit are subordinated to the entire law hierarchy. In the current FROG version, the Organization.def addresses only the law specifications, and we are leaving other features to be developed as future work.

- Controller - In our conceptual framework the agent communication is decentralized and, furthermore, the programming language that can be used for agent implementation may be different. Thus, it is necessary to include an intermediate communication level among agents communication to be able to verify whether the laws are followed. The Controller element was introduced to accomplish this task and act as an agent proxy in an organization.

- Certification Authority(CA) - This entity makes the system more secure by helping to provide guarantees about the trustfullness of the agents and, thus, enforcing more secure multi-agent interactions and protocol fulfillment. In this sense, the CA certifies the controllers that can be trusted and, among other tasks, provides authentication of assignments of roles to agents.

In a general way, the interaction among the elements previously described happens as follows: when an agent A needs to send a message to an agent B, the message must first be sent to the controller of the agent A. Then, this controller verifies whether in Organization.def there exists any rule that must be dispatched because of the message that was received. If such a rule exists, this rule will be fulfilled. After that, the controller of the agent A sends the message to controller of the

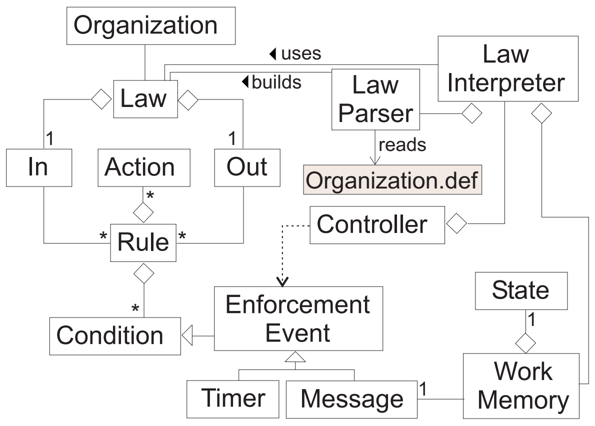

Figure 1: Conceptual diagram of FROG elements relationship

agent B, which also verifies whether there exists any rule that must be dispatched because of the message that has been received. Finally, the controller of agent B sends the message to agent B.

Figure 1 shows a conceptual view of the relationships among FROG elements. According to this figure, each organization has a law and each law is composed mainly by the elements *in* and *out*. These elements have an associated set of rules and each rule has an activation condition and actions that will be dispatched when these conditions are satisfied. The activation condition can refer to any event including message or temporal events. We also note that the controller is composed by a law interpreter. This interpreter uses a syntactic analyzer that reads the Organization.def element and builds a structure that can be used by the interpreter. Furthermore, this interpreter has a memory consisting of two objects (state and message) that can be used in the law specifications.

3.1 Organization.def

This element defines a language to represent the main Organization features as laws, and should be instantiated through a text file. The language is composed by two main blocks, the block *in* and the block *out*. The block *in* is where the rules, which should be fulfilled by a controller when it receives an event, should be defined. The block *out* has a similar semantics, except that the event that is received must be has to be originated by the agent that this controller represents. Events can be specialized in two kinds: message or temporal events. Message events occur in the context of agent communications and temporal events can be

```
IF message.get(”type”).equals(”x”){
    IF state.get(”counter”).value == 1{
        message.consume();
    }ELSE{
        state.get(”counter”).set(1);
    }
}
```

Figure 2: Example of law specification

used to specify constraints such as “at each 15 minutes verify some condition”, or “every day at 19:00hs do some action”. The rules related to organization laws mainly provide support to constraint the multi-agent behavior. These rules are specified using an *IF_THEN* format. This allows a large number of constraints to be specified. Furthermore, the Organization.def should describe objects representing the events and the historical information related to organization-based interactions. Using these objects, the person that is writing the Organization rules will be able to have rules that involve both the events and information related to past agent interactions. The *message* and *state* objects are examples of two of these objects. The *message* object allows the manipulation of the messages that an agent is receiving or sending. This object is composed by pairs of fields and contents and, in this way, any message based on a field-value structure can be mapped to this object. For example, a FIPA-ACL message [3] has some fields such as *conversation-id* and *sender*, and these fields has content values. For each specific language, it is necessary build a translator that receives a domain specific kind of message and translates this message into the structure used in the *message* object. The *state* object is like a memory that can store the information that is generated during the law enforcement. This object allows contextual rules to be written, e.g., “a message of a certain type could be sent only once”. We could implement this constraint by writing a rule that verifies whether it is the first time that the specific message is being received. If this is the case, we can dispatch this rule and increase an auxiliary counter by 1 in the *state* object. Thus, when another message arrives, the rule checks the value of this counter and the rule will not be dispatched again. Figure 2 illustrates how this law can be implemented.

3.2 Controller

The controller is the element responsible for interpret the Organization.def. It is the controller that ensure the fulfilling of the laws specified in Organization.def.

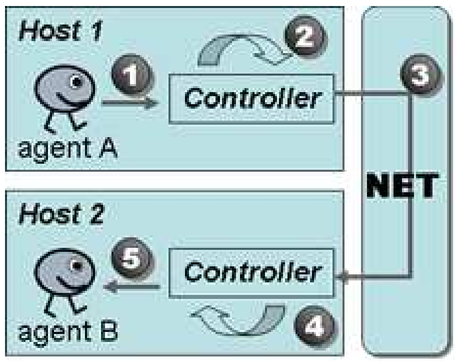

Figure 3: Agent A sending a message to agent B

The process's flows can be seen in Figure 3, where an agent A send a message to an agent B, then, this message is received by the controller of agent A that interpret the law according with the message received and, then, this controller sends the message to the controller of agent B. This controller also interprets the law and, finally, sends the message to the agent B. When the design of the controller undertaken, two kinds of strategies were analyzed. A first one would allow us to implement only one controller that would receive all the messages exchanged in the multi-agent interactions. This approach is intrinsic non-scalable because it would introduce a single point through which all the messages would have to pass. Therefore, to avoid this problem, we have adopted a second strategy, in which the controller would be designed in a decentralized way and where each agent would have its own controller.

3.2.1 The Controller's communication mechanism

One of the significant goals related to the conception of FROG is to develop a multi-agent system that is technology independent. So, the FROG implementations in different languages such as C++ or JAVA are able to communicate with each other. To reach this goal, the communication among the FROG implementations had to be programming-language independent. The FROG communication mechanism follows this requirement and a diagram of this model is shown in Figure 4. In addition, using these components in this model, the addition, deletion and update in the communication and transport protocols leads to more flexible interactions. Each of these components are described as follows:

- Communication - This component manages the agent communication and

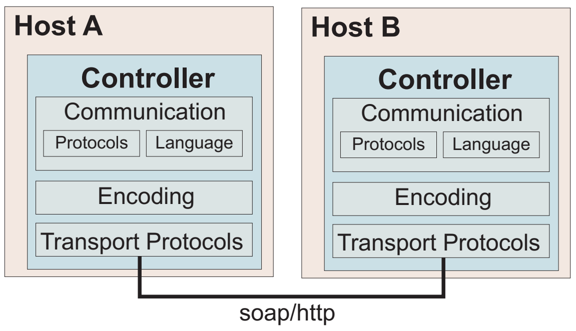

Figure 4: Controller architecture

it is composed essentially by communication languages and protocols. The design of this component should possibility addition of new communication languages and protocols.

  – Communication Language - In this work, the communication languages we have adopted are languages such as SOAP [18] and FIPA-ACL. One of the important advantages of these languages is that they provide programming-language independence. They also have other advantages as, for example, in FIPA-ACL, the communication among agents is based on speech acts theory and, for this reason, this language provides additional semantics that can be used by the agents. Furthermore, each application can implement its specific languages, but, in despite of this fact, we must choose at least one default language that all controllers must implement to allow communication among the implementation of different controllers. The SOAP language was chosen for this purpose. SOAP was chosen because it is programming-language independent and is based on open source specifications. Furthermore, it has a large industrial acceptance that is gradually growing more and more mainly due to Web Services [19]. As result, integration with existing systems is facilitated.

  – Communication Protocols - These protocols are high-level protocols often identified as a sequence of interactions that are in agreement with the protocol specification. Examples of such protocols are the specifications needed to ensure the trustfulness of a controller or to acquire an Organization.def element from Internet.

- Encoding - In general, the communication languages are represented as objects that allow a more intuitive manipulation of the messages expressed one needs to describe. However, these messages must be encoded to be sent across the network and decoded in objects when they are received. The Encoding component defines the possible encoding formats. Example of such formats are string, XML and serializable objects.

- Transport Protocol - The main goal of this component is to allow the use and management by the controller of many network protocols such HTTP, UDP and IIOP. For the same reason we have given previously in the case of the communication language specification, we also needed to choose a default transport protocol. The HTTP was chosen because of its flexibility and widespread use in the Internet.

However, besides defining such elements, we need to ensure that the communication between agents and their controllers is based on open standards and provide programming-language independence. So, we have chosen SOAP as the default communication language and UDP as the default transport protocol. This combination is the default combination, a combination that is mandatory in all controller implementations. But this requirement does not impede the existence of other combinations such as FIPA-ACL and IIOP, because this kind of controller also implements the default combination. As result of our proposed specification for these elements, different FROG implementations can communicate with each other and an organization law that has been used in one FROG implementation can also be used in another implementation without modification.

3.3 Certification Authority

One significant feature of controllers is that they must be trustworthy, in the sense that they must be able to interpret the Organization.def. In order to guarantee that a controller can trust another controller, we need ensure the authenticity of the controllers involved in a communication process. For this reason, we introduce an entity that is responsible to certify the trusted controllers, and each controller, through this entity, can verify whether the controller with which it is interacting is trustworthy. The entities that can provide certificates of authenticity are called *Certification Authority*. FROG provides two kinds of certification authorities: one of them provides authentication for controllers and the other one provides authentication for agent roles by certifying the agents that can play a specific role. The last entity was introduced as an open system feature because, for example, an agent who says that it is a manager has to be certified to give assurance that this is really the case. However, all these authentication features discussed previously are defined as optional features of an organization and, in this way, an organization may exist both in a free and not so trusted environment, or in a rigid and trustworthy one.

4 JFROG

In this Section we discuss some issues related to the design and implementation of JFROG. JFROG is a Java implementation of the FROG conceptual framework. Currently, JFROG is in the final phase of its development, and we expect that it will soon be available in the Internet as a free software license. The main element of FROG is the Controller that will be described in this section. As it was mentioned in Section 3.2, the controller is responsible for interpreting the Organization.def. To perform this task, JFROG uses both a syntactic and lexical generator called javacc [20] and the JJTree, which is a javacc pre-processor that can be used to build an abstract syntactic tree during the syntactic analysis phase [21]. Then, the Backus-Naur Form (BNF) of Organization.def is specified in a JJTree compatible format. The JJTree generates a file that is processed by javacc and that will generate a syntactic analyzer, a collection of auxiliary classes and a set of classes that represent the nodes of the syntactic tree. This process is illustrated in Figure 5. The adopted mechanism allow us minimize the costs related to the evolution of an Organization.def element, because most of the work related to changes can be automated using the framework and its related tools.

The controller has a component called Law Interpreter that is responsible for interpreting the Organization.def. This is illustrated in Figure 5. When an event is received by the controller, this component uses the syntactic tree node classes to interpret the Organization.def. FROG also defines that controllers must have the *Communication*, *Encoding* and *Transport Protocol* elements. More specifically, we notice that the encoding element is dependent on the communication language representation used in an application. For example, we can have an application that uses the FIPA-ACL communication language and messages are sent to the network in string format, as specified in [22]. However, to manipulate the messages in easier way, in this application it was written a translator so that when a message is received encoded in FIPA-ACL string format, this translator transforms the message into an ACLMessage object. Then, we see that if we change the structure of ACLMessage object, we also need change the algorithm that transforms a

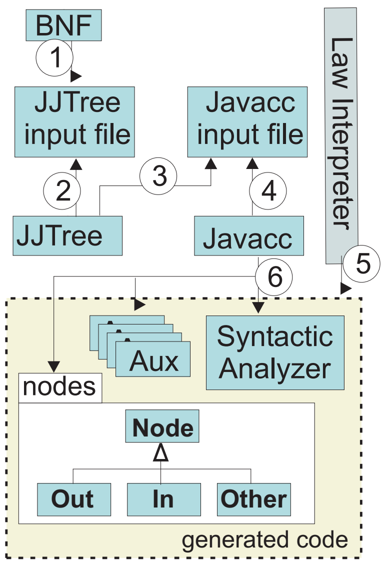

Figure 5: Interpreter generation

FIPA-ACL string message into an ACLMessage object. Thus, the encoding process depends of the way that the messages are represented in a specific application. For this reason, the Encoding component is implemented as an interface called *ObjectMessage*. In this way, all the objects that represent a message expressed in a certain communication language must implement this interface, which declares the methods *encode*() and *decode*(). Therefore, the messages need to be able to encode and decode themselves into a particular format such as a string, XML, or serializable objects. The interface ObjectMessage is illustrated in Figure 6. To use multiple transport protocols and to minimize the impact due to changes in such protocols, we have created the classes TransportService, TransportDescription, TransportFactory and an interface called Transport, which are illustrated in Figure 6. The class TransportService is a Façade [23] to the services provided by the transport layer, providing methods such as *send*() and *receive*(). This class uses the TransportFactory class to retrieve an instance of a specific transport protocol, e.g., HTTP or UDP. Furthermore, this class also has access to information describing which are the protocols that can be used to interact with a specific agent. In this way, an agent can interact in many different ways, using, for example, SMTP or HTTP. Such descriptions are implemented by the TransportDescription objects that are used by an instance of TransportFactory class. An architectural decision that we also need to address is whether the controllers can be accessed

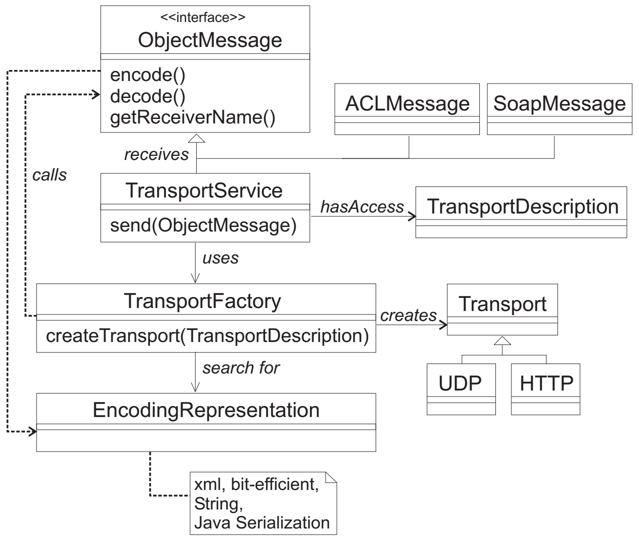

Figure 6: Class diagram: Communication, Encoding and Transport Protocol components

remotely. In order to allow them to be accessed remotely, one more node would have to be implemented to enforce a secure interaction protocol between an agent and its controller. In this case, when an agent sends a message to another agent, this message would have to cross the network for three times: from the sender agent to its controller; from sender's controller to the addressee's controller; and, finally, from addressee's controller to addressee agent. And in each of these three cases, the security protocols would have to be executed. To avoid this communication overload, the controller should be running in the same host of its agent. In this way, when an agent sends a message, the security protocols will be executed only once [1], and only to enforce the controller-controller communication. JFROG provides a client API that implements the Communication, Encoding and Transport Protocol components. In this way, the developer can focus in his business problems, and the details related to the law enforcement mechanisms can be abstracted. Figure 7 shows the JFROG architecture. This figure has an element that has not been described so far, namely the *Translator*. This elements implements the translation from specific communication languages such as FIPA-ACL into the *Message* object defined by FROG. Thus, the Law Interpreter can work independently of the communication language used in a specific application. For

[1]In this special case, we consider the the agent host to be a trusted host

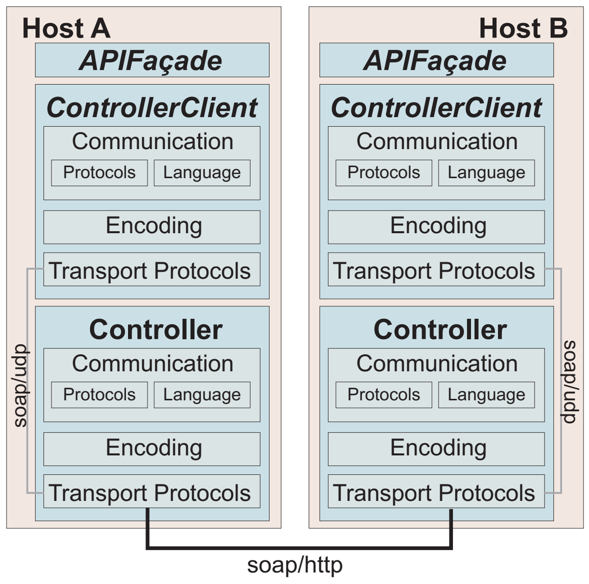

Figure 7: JFrog architecture

instance, to use SOAP as a communication language we just need to write a class that translates a SOAP message into the Message object.

4.1 Tools

To help manage virtual organizations that are defined using the Organization.def element, we are developing two tools:

- the JFROG-Organization tool, a graphical tool that helps with the creation and management of organization files. As its main features, this tool provides a law editor, a syntactic checker, and wizards that help the publication of the organization files.
- the JFROG-Administration tool that is used by the runtime environment administrators and that basically provides a friendly front-end to the configuration XML files.

5 Related work

In this paper we have described a framework called FROG to enforce secure interaction protocols in multi-agent systems, and its associated implementation, JFROG. The framework uses laws as a general abstraction to describe interaction constraints and these laws are specified in a declarative language based on IF_THEN rules. The declarative nature of the laws and of the specifications we have adopted allows our approach and framework to be language-independent in the sense that the declarations are not based on any specific implementation language. In his work, Nicklisch [24] has also used rules to define policies in domain of network management. However, his approach is specialized for this domain and the language proposed in our work is not, being therefore more general. Another work on laws related to our paper is the work done by Cole et al. [25], in which the authors propose a way to identify laws in real world problems. However, their result does not deal with issues related to law enforcement and specification. In addition, Mineau in [26] has proposed that laws should be specified using a conceptual graphs approach. This approach supports the validation of rules and uses a very rich and expressive language. However, the approach also leads to an increase in the complexity of the rule specifications and one of the main goals of FROG is to have a simple language, one that has a fast learning curve. Minsky et al. also uses the notion of laws to ensure law enforcement in agent organizations. His approach was reported in a number of publications referred as Law-Governed Interactions (LGI) [27, 28, 29, 30]. In contrast, FROG also has as one of its goals the law enforcement in agent organizations. However, FROG addresses issues related to the enforcement of secure interaction protocols in the context of heterogeneous systems and, through its conceptual and flexible declarative framework, it leads to different FROG implementations using different languages. Furthermore, FROG allows the rules to be specified independently of the interaction language used.

6 Final remarks

In this paper we have presented a conceptual framework called FROG to ensure secure interaction protocols in multi-agent systems, and its implementation, JFROG. This framework was conceived using the concept of laws as a generalization of other concepts such as policies, protocols and norms. FROG provides mechanisms to support law implementation and to enforce laws in open and heterogeneous multi-agent systems. As part of our future work, we will continue experimenting with the approach and will develop further case studies and tests involving JFROG.

References

[1] Jennings, N.R.: Agent-Oriented Software Engineering. In: Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World. Volume 1647., Springer-Verlag (1999) 1–7

[2] Linington, P.F., Neal, S.: Using policies in the checking of business to business contracts. In: IEEE 4th International Workshop on Policies for Distributed Systems and Networks. (2003)

[3] : Foundation for intelligent physical agents (2003)

[4] Lopez, F., y Lopez, Luck, M., d'Inverno, M.: A framework for norm-based inter-agent dependence. In: Third Mexican International Conference on Computer Science, SMCC-INEGI (2001) 31–40

[5] Gupta, V., Jagadeesan, L.J., Jagadeesan, R., Jiang, X., Laufer, K.: A constraint-based framework for prototyping distributed virtual applications. In: Principles and Practice of Constraint Programming. (2000) 202–217

[6] Singh, G.: Constraint-based structuring of distributed protocols. In: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing, ACM Press (1995) 266

[7] Ferber, J.: Multi-Agent System: An Introduction to Distributed Artificial Intelligence. Addison-Wesley (1999)

[8] Zambonelli, F., Jennings, N.R., Wooldridge, M.: Organisational rules as an abstraction for the analysis and design of multi-agent systems. International Journal of Software Engineering and Knowledge Engineering **11** (2001) 303–328

[9] Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Third International Conference on Multi-Agent Systems (ICMAS98), Paris, France. (1998) 128–135

[10] Galbraith, J.R.: organization Design. Addison-Wesley Pub. Co. (1977)

[11] March, J.G., Simon, H.A.: Organizations. John Wiley Sons, Inc. (1958)

[12] M.S., F.: An organizational view of distributed systems. In: IEEE Trans. on System, Man and Cybernetics. Volume SMC-11. (1981) 70–80

[13] So, Y., Durfee, E.: An organizational self-design model for organizational change (1993)

[14] Wooldridge, M., Jennings, N.R., Kinny, D.: The gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems **3** (2000) 285–312

[15] Zambonelli, F., Jennings, N.R., Omicini, A., Wooldridge, M.: Agent-Oriented Software Engineering for Internet Applications. In Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R., eds.: Coordination of Internet Agents: Models, Technologies, and Applications. Springer-Verlag: Heidelberg, Germany (2000) 326–346

[16] Dictionary.com, T.F.: Law definition (2003)

[17] Searle, J.R.: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press (1969)

[18] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D.: Simple object access protocol (soap) 1.1 (2000)

[19] W3C: Web services activity (2002)

[20] Group, J.: Java compiler compiler [tm] (javacc [tm]) - the java parser generator. https://javacc.dev.java.net/ (2003)

[21] Aho, A.V., Sethi, R., Ullman, J.D.: Compilers. Addison-Wesley Pub Co (1986)

[22] for Intelligent Physical Agents, F.F.: Fipa acl message representation in string specification (2002)

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

[24] Nicklisch, J.: A rule language for network policies. Policy Workshop 1999 (1999)

[25] Cole, J., Derrick, J., Milosevic, Z., Raymond, K.: Author Obliged to Submit Paper before 4 July: Policies in an Enterprise Specification. In: Policies for Distributed Systems and Networks. Volume 1995 of Lecture Notes in Computer Science., Springer-Verlag (2001) 1–17

[26] Mineau, G.W.: Representing and enforcing interaction protocols in multi-agent systems: an approach based on conceptual graphs. In: IEEE/WIC International Conference on Intelligent Agent Technology. (2003)

[27] Ao, X., Minsky, N., Nguyen, T.D.: Hierarchical policy specification language, and enforcement mechanism, for governing digital enterprises. In: 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02). (2002)

[28] Ao, X., Minsky, N.H., Nguyen, T.D., Ungureanu, V.: Law-governed internet communities. In: Coordination Models and Languages. (2000) 133–147

[29] Minsky, N.: The imposition of protocols over open distributed systems. In: IEEE Transactions on Software Engeneering. Volume 17. (1991) 183–195

[30] Murata, T., Minsky, N.H.: On monitoring and steering in large-scale multi-agent systems. In: Selmas'03 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, Portland, Oregon (2003)