

Towards a Risk Driven Method for Developing Law Enforcement Middleware

Gustavo Carvalho¹, Rodrigo Paes¹, Ricardo Choren² and Carlos Lucena¹

¹ Departamento de Informática – Laboratório de Engenharia de Software – PUC-Rio
Rua Marquês de São Vicente, 225 Rio de Janeiro, Brasil, 22453-900
e-mail: {guga, rbp, lucena}@inf.puc-rio.br

² Departamento de Engenharia de Sistemas - Instituto Militar de Engenharia - IME
Praça General Tibúrcio 80, Rio de Janeiro, Brasil, 22290-270
e-mail: choren@de9.ime.ub.br

Abstract. Nowadays, multi-agent system (MAS) analysis relies mostly on the specification of individual software agents, considering unpredictability as a consequence of the interaction among them. In this paper, we propose a law enforcement method that uses risks as guidelines for open MAS infrastructure specification and development. The method contributes to a better understanding of the problem characteristics and of the causes and consequences of specific behaviors in the MAS, thus contributing to the improvement of dependability attributes. By structuring the design decisions as risks, the method also aims to help on future evolutions and reengineering of the law enforcement middleware.

Keywords: software agents, open systems, risks, law-enforcement, methodologies, middleware

1. Introduction

Software systems have become larger and more complex than ever [23]. A system is a collection of elements related in a way that allows a common objective to be accomplished [21]. Aiming to provide a new generation or an improvement of existing tools and techniques to help the development of those complex systems, software agents have been proposed as a promising paradigm [22,25].

The causes of unexpected or undesirable side effects in multi-agent systems may be very complex. They may lie in the environment where software agents inhabit, in the multi-agent system infrastructure, in any particular software agent behavior, or in some interdependencies between those entities. Because of those interdependencies, it is necessary to understand not only the agents' individual behavior or the basic infrastructure characteristics, but also the effects of the interaction between software agents and the consequences of those interactions within the environment. Furthermore, the characteristics of multi-agents systems present many other sources of uncertainty; for instance, this kind of solution could be based on complex organizations

where participants collaborate in a harmonious or competitive way to achieve their individual goals and it could also present a systemic evolution or adaptation without having a centralized control on how or when those adaptations would happen. In multi-agent systems, this uncertainty about what exactly will happen is described as emergent behavior.

Generally, achieving sufficient dependability in system development and demonstrating this achievement in a convincing and rigorous manner is of crucial importance [21]. The term dependability was proposed by Laprie [13] to cover the related system attributes of availability, reliability, safety and security. In this sense, a developer should consider and assess dependability attributes while developing any system. In an open multi-agent system the experience of emergent behavior makes it even more important to consider those attributes. Currently, no approach for developing multi-agent system infrastructures provides means to effectively gauge those characteristics.

Basically, a systematic exposition of the concepts of dependability consists of three parts: the threats to, the attributes of, and the means by which the dependability is attained [2]. Besides threats, it is important to extend this definition including beneficial consequences or opportunities that could also arise from interactions among agents.

A mechanism to enforce behavioral rule (law enforcement mechanism) aims to provide a way to map the consequences that arise from the interactions to the qualitative or even quantitative criteria presented by the dependability attributes. An enforcement mechanism intercepts particular interactions between agents to control and audit the execution flow of conversations. The enforcement approach aims to contribute to a convergence from an unstable and totally unpredictable open multi-agent system under development to a dependable, more stable and less unpredictable one.

Enforcement rules are specified as laws and norms that are associated with well-known consequences that may be subjected to any participant of the multi-agent system. The laws and consequences can be identified through a risk-driven analysis of the system environment. The use of an enforcement mechanism and of a risk driven method permits to control the failures and to promote the benefits and also contributes to tame the uncertainty presented by open multi-agent systems.

In this paper, we propose a method that considers risk analysis concepts to provide a structured way to specify, implement, monitor and maintain systems requirements and the existent challenges, opportunities, threats and limitations identified through the development of the solution. Briefly, this analysis is based on the identification, control and assessment of relationships between cause and consequences states, events and their characteristics. With this approach, we propose to evolve and develop a law enforcement infrastructure using risks as guidelines. The goal of this method is to provide a structured way to develop the law enforcement middleware, i.e., an infrastructure that contributes to the fulfillment of open multi-agent systems specifications and laws. This approach considers that the software agents are developed by different people using those specifications and laws generated with this method.

In the following section, we present a synthesis of the law enforcement approach and we discuss risks on open MAS development. After that, we present the method

for developing open multi-agent middleware in section 3. In section 4, we present the case study used to experiment the method. Finally, we present some related work and then conclude the article with some considerations and directions about future works.

2. Law Enforcement Approach and Risks in MAS Development

The explicit identification of organizational rules is an important issue in the context of open systems. With the arrival of new, previously unknown, and possibly self-interested agents, the overall organization must be able to enforce its internal coherency, despite the dynamic and untrustworthy environment [25]. Trying to regulate and to limit part of the unpredictable behavior that open systems can present, we propose to apply a law-enforcement approach [19].

In this sense, specification artifacts with rules could be elaborated to guide the development of these system components through the convergence into a desired behavior. At runtime, it is possible to enforce those rules. Rules could be seen as a boundary that stipulates the level of freedom that any unit has to act and they are used to maintain the acceptable levels of dependability. These rules or laws are applied on software agents' conversations and describe permissions, prohibitions and obligations that software agents must fulfill.

Rules specify restrictions and conditions concerning interaction protocols. A protocol is a definition of a set of legal actions that can be performed during agents' conversations. It specifies valid rules and the acceptable behavior concerning interactions that agents could participate and it also defines and restricts the context where the information exchanged must be interpreted [7]. A protocol is composed of transitions. A transition structures the evolution of the process from a previous state to the next one. It also defines the types of participants and represents the valid actions that can be performed by them.

Norms prescribe how the agents ought to behave, specify how they are permitted to behave and what their rights are [12]. A norm is composed by obligations, permissions and prohibitions. Each kind of norm has activation or deactivation conditions and consequences.

Dependable software includes extra, often redundant, code to perform the necessary checking for exceptional states and to recover from system faults. Because of this additional design, implementation and validation efforts, increasing the dependability of a system can significantly increase development costs [21]. In this sense, a criteria to establish an importance order using a risk analysis method should be applied to help the assessment of alternatives and the gauging of how much of these efforts are really necessary.

Risk is a possible future event that, if it occurs, will lead to an undesirable outcome or to unpredictable consequences and characteristics [14]. A dependability attribute can be modeled as risks, represented as a sequence or a chain of cause and consequence states. This sequence is helpful to understand the origins of consequences and how those characteristics evolve during the execution of the multi-agent system. By this arrangement it is possible to infer and analyze how we can predict and alter the consequences, understanding their causes and the links between intermediary states.

The techniques used to understand and analyze the chain of causes and consequences, navigate from both directions of a chain (Figure 1). Looking the chain from the consequences through the past, a technique tries to find out the causes and the origins of the problems or opportunities. This kind of technique is concerned with answering why the consequence could happen. In the other side, another approach could be concerned with answering how some causes would influence the system behavior and dependable attributes, this analysis begins with identifying the root causes and tries to derivate the consequences through the evaluations of future states in the chain.

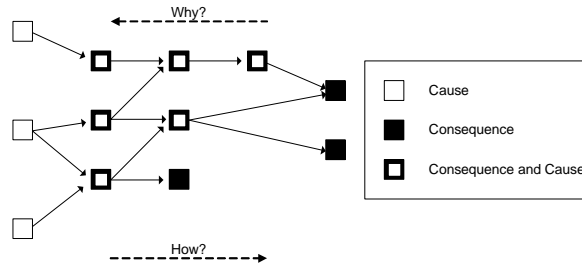


Figure 1 - Chain of cause and consequences instance

3. Risk Driven Method for Developing Law Enforcement Middleware

The risk driven method for developing the open multi-agent system infrastructure described here intends to guide the law enforcement approach basing the specification, implementation and monitoring of open multi-agent system on a risk analysis.

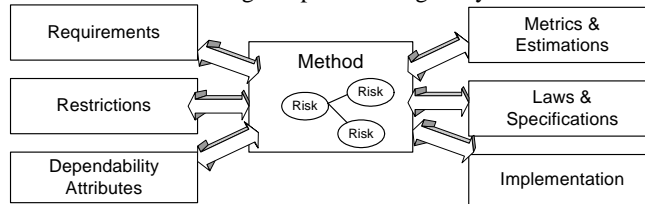


Figure 2 - Risk Driven Method for Developing Open MAS Overview

Analyzing carefully the requirements, restrictions, existing components and their relationships we have a base to specify the solution. Furthermore, we aim to implement a dependable multi-agent system. In this sense, dependability attributes are mapped to requirements, but due to their importance they have to be specially considered in this process. All the design decisions and important considerations are mapped to risks. Laws are elaborated and specified through the analysis of the risks. Consequences of not observing laws are described and implemented too. During the specification phase, metrics and estimations are identified. Metrics information is collected during the execution phase and it is compared to the estimations. Reengi-

neering decisions could be made based on the observation of the system resultant behavior and on the comparison with what was being expected.

The method for developing law enforcement middleware is composed by three phases: the specification phase, the implementation phase and the execution monitoring phase. These phases include the following steps: (A) identification of “desired” behavior specification; (B) protocol specification; (C) identification of software related risks and metrics; (D) analysis of software risks; (E) assessment of risks; (F) development of plans to address software risks, i.e. law specification and description, design decisions and metric values estimation; (G) law enforcement infrastructure implementation; (H) agents and services implementation; (I) tests of the infrastructure; (J) configuration of law enforcement infrastructure; (K) active monitoring and tracking of software behavior and risks using metrics; and (L) reevaluation of risks, laws and extensions, i.e., open system assessment.

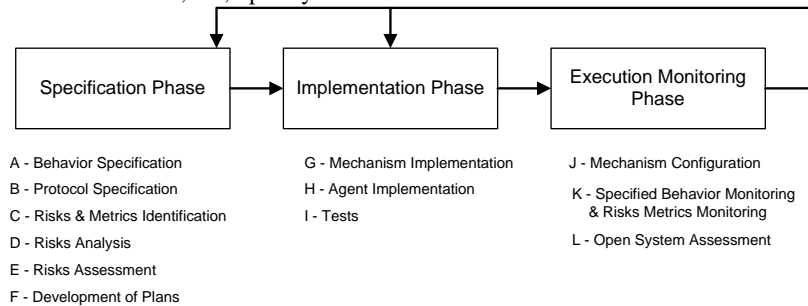


Figure 3 - Method Schematic Overview

Steps A, B, C, D, E and F comprehend the software specification phase. Steps G, H and I are mapped to the implementation phase and are concerned with the implementation of software agents and with the development of extension points of the open system infrastructure. Finally, steps J, K and L correspond to the execution monitoring phase where, at runtime, engineers have the opportunity to evaluate the behaviors and metrics that were previously specified.

3.1. The Specification Phase

Several requirements are gathered to specify system characteristics (A). Software requirements can be categorized as functional requirements, performance requirements, external interface requirements, design constraints and quality attributes [23]. Non functional requirements define the required reliability and availability of the system. “Shall not” requirements, like safety and security considerations, define the system behaviors that are not acceptable and can be decomposed into a more specific set of functional requirements [21]. Those requirements will describe the behavior of the system that is under development.

As mentioned before, we are trying to gauge system dependability through a risk driven analysis and a law based approach used to enforce norms and protocol specifications preserving the open system dependability attributes. Among other types of specification, agent interaction protocol and norms are specified focusing in detailing

and describing the requirements determined for the agents' conversations (B). This description can be used to identify problems or threats in the protocol specification, complementing this specification with restrictions, obligations and any other kind of constraints. A protocol comprehends the list of all events that arise during agent interaction. Only events specified in protocols are allowed. An important decision while specifying conversations is to choose between describing the global protocol or their different agent views [7]. In the former case, the protocol is specified considering all the messages exchanged among agents. In the latter one particular view of the protocol for each participant is specified. The whole protocol can be composed of the union of each particular view.

Risks are gathered during the specification phase (C) and they will guide the decisions that will be made during the other phases including the implementation and monitoring of open multi-agent systems. Through the understanding, identification and assessment of risks involved in the software development or operation, this method aims to gain insights into the whole by specifying the linkages, interactions and processes between the elements that comprise the system. Besides, wherever possible we have to consider some dependability metrics (C) to assess how dependable a system is [21]. For instance, among different existent metrics for reliability, we can mention the mean time to failure, probability to failure on demand, rate of failure occurrence and availability metrics.

Risk analysis (D) comprehends analyzing the system and its operational environment; discovering potential results; discovering the root causes of these results; and identifying the risks and benefits associated with them [21]. Each requirement should be assessed for the risk it poses and the specification may either describe how the software should behave to minimize the risks, maximize the benefits or require that the result or characteristic should never or always exist. For each identified consequence, a detailed analysis should be carried out to discover the conditions which might have causes or originates the result. This analysis is the joint use of deductive and inductive techniques. Deductive techniques are generally easier to use and start with the consequences and work from that to the causes. On the other hand, inductive techniques start from the causes and identify which consequences might arise. For instance, with the fault tree analysis [15] you first identify the undesirable event or desirable transition and then work backwards from it to discover the possible causes. The consequence is at the root of the tree and the leaves represent potential causes.

Risk assessment (E) is done considering the severity of each consequence, the probability that it will arise, and the probability that a result will be originated from it [21]. For each item, the outcome of the risk assessment process is a statement of acceptability. A risk classification could be used to help the choice of which requirements would be specified as laws that would be enforced by a controller. For example, the classification presented in [4] proposes three levels including intolerable (not arise or not cause bad consequences), as low as reasonably practical (minimize the probability of bad consequences arise) and acceptable (reduce the probability of this cause arising). By analyzing those levels, we evaluate the impact of the consequences and we can propose a specific treatment and amount of attention that will be spent on each one.

Dependability represents non functional requirements that implies on new functional requirements that specifies how results may be avoided, given incentives or tolerated. In the proposed approach, functional and non functional requirements could be specified as design decisions or expressed as laws and consequences that help achieving the objectives regulated by these restrictions. More then only identifying the risks metrics, we have to estimate how those metrics should evolve during software execution. Making estimation involves making assumptions, observing constraints and dealing with uncertainty [8]. The estimation of metrics values, the law descriptions and the set of design decisions can be seen as results of the activity named development of plans to deal with risks (F).

3.2. The Implementation Phase

In this phase, any necessary service to provide the enforcement needs to be implemented. We proposed a law enforcement architecture to guarantee that the specifications will be obeyed (Figure 4) and developed an infrastructure which includes some communication components that will be provided to agent developers. This architecture is based on a mediator that intercepts every message and interprets the laws previously described. The main goal of this phase is to provide the infrastructure for the mediation of conversations between agents and it is responsible for developing the basic communication components of agents and the interoperability concerns.

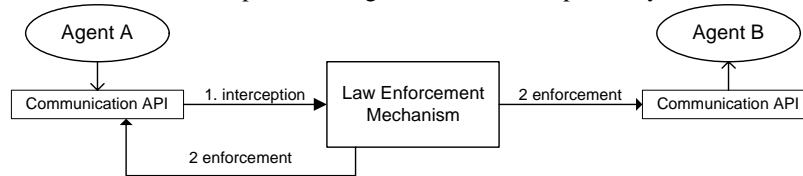


Figure 4 - Law Enforcement Architecture

Depending on the solution domain, it could be necessary to extend this basic infrastructure to attend system requirements. Those extensions have to be made in this phase. To develop dependable software this infrastructure must implement some dependability techniques like fault tolerance, error handling and redundancy.

In this phase, agents are implemented, but this development can be done without a centralized control. The developers may have an a priori access to every specification, protocol descriptions or laws generated during the specification phase. Finally, some tests are executed to verify the functionalities of the infrastructure.

3.3. The Execution Monitoring Phase

Among the different activities that deserve the analysis of causes and consequences, it is important to observe that dependability attributes must also be managed during the operation of the system [21]. In this sense, the monitoring phase determines the activities for observing and controlling the open system execution. It also measures the progress of interactions, reviews intermediate results and provides means to be taken corrective actions when necessary using configuration alternatives. Open system

control corresponds to a monitor and measures performance and results against plans, notes deviations, and takes corrective actions to ensure conformance between plans and actual results. It is a tool that could act like an auditor that guarantees that what is happening conforms to what have been specified.

This control can be seen a feedback system for how well the open system is going, helping the software engineer to understand the complexity of the dynamic of the open system. It can also present a resumed list of metrics and others information that could guide reengineering decisions. Those decisions could be the modification of a specific rule, the redefinition of the interaction protocol or even the reimplementation of a basic service.

4. Mobile Ticket Trader Case Study

Suppose an airport where flight companies and passengers have an immersive environment for negotiating flight tickets. Immersive in the sense that the goal of this environment is to enhance computer use by making many computers available throughout a physical environment, and by also making them effectively visible to as many users as possible, and that the users could have access to the airport services using systems like PDAs and mobile phones. Flight companies and passengers are represented by software agents and they can enter or leave the environment at their own will.

Frequently, flight companies offer tickets for commercial flights. The goal of a flight company is to sell the maximum number of tickets, to increase the user satisfaction and to charge them as much as possible. Passengers use palmtops when they arrive at the airport to buy flight tickets. Each passenger has a specific profile that defines his/her preferences concerning the destination, flight types, maximum acceptable ticket cost, and any other characteristics.

In this context, imagine a scenario where persons could bargain discounts because they are buying more tickets in a same negotiation process. This group should be formed considering common preference attributes of the participants specified in their personal interests, e.g. the same destination, price, comfort or time of departure.

In the whole negotiation process a specific step exists for forming groups of interests, where the participant personal profiles are combined aiming to inform other participants that have close interests. Using this information, it is possible to form a group with close preferences and this group can bargain discounts with the sellers.

4.1. The Specification Phase

Understanding the risks, it is possible to specify interaction protocols and laws that will regulate the multi-agent system. Below (Figure 5), we have the steps of the case study protocol specification. The user arrives physically at the airport; it tries to form groups to bargain discounts; it sends messages to the flight companies' agents (FCA) to negotiate their values and attributes; it must pay the ticket to the air flight company and the FCA emits the electronic ticket; it can check in using the electronic ticket, and it leaves the environment.

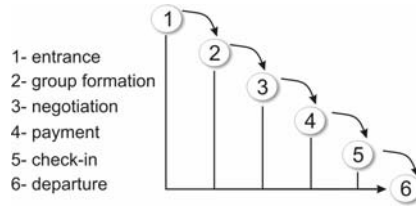


Figure 5 - Agents Execution Scenario

To show an example of a law description, we will further specify the payment step. This step starts if there is success at the end of the negotiation process. The negotiation process is based on the fipa contract net protocol [9] specification. If the negotiation process is done by a group, the whole group must complete the payment step and honor their commitments. If one member of the group does not fulfill its obligations, there is a risk of a failure in the whole negotiation process. Then, it is necessary to specify a norm to deal with this situation and protect buyers and ticket sellers of known “bad” agents. The norm can be informally specified in the following way:

Norm 1: After the success of the negotiation all members of the group acquire the obligation of paying the ticket. If one of them doesn't pay the ticket, it will be prohibited of future interactions in the airport.

Another example of norm is the specification of time-outs for preserving availability conditions. For instance, suppose that the permission of canceling a confirmation was granted to the buyer when the seller has spent more than a specific amount of time to answer to him. The norm can be informally specified in the following way:

Norm 11: After the period of 60 seconds subsequently to the confirmation of the purchase by the buyer, it is granted to him the permission to cancel the commitment.

Furthermore, many other laws can be specified aiming to deal with the unpredictable behavior of the participant agents. This informal specification should be certainly specified in a formal manner. However, focus on the law specification is out of scope of this paper. Another example of result from the specification phase was the identification of some metrics considered useful for understanding the process. For example, it is important to monitor how many participants disobey the *Norm 1*.

4.2. The Implementation Phase

We implemented the enforcement mechanism as a mediator of agents' conversations and a component that extend the Jade [3] communication API to provide the redirections of messages. This mediator agent (Figure 6) has the implementation of a state machine that corresponds to the protocol and to the norms previously specified.

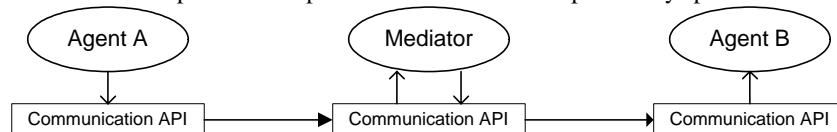


Figure 6 - Enforcement Mechanism Implemented as a Mediator Agent

4.3. The Execution Monitoring Phase

We also developed an application to monitor the process execution, for example, this application shows the norm activation and deactivation and it collected the metrics that were previously specified. If the collected metric breaks a previously estimated threshold, for example the percentage number of agents that disobey the *Norm 1*, this information could indicate that some reengineering of the process should be promoted.

5. Related Work

Law Governed Interaction (LGI) [18] proposes a mechanism for coordinate and control heterogeneous distributed systems. It is based on four basic principles: coordination policies need to be enforced; the enforcement needs to be decentralized; and policies need to be formulated explicitly rather than being implicit in the code of the agents involved and they should be enforced by means of a generic, broad spectrum mechanism; and it should be possible to deploy and enforce a policy incrementally, without exacting any cost from agents and activities not subject to it. This approach does not provide an explicit method to develop and evolve its law enforcement infrastructure.

One of the most interesting related works is the electronic institutions approach [7, 10, 20]. In this approach some concepts related to law enforcement were formalized, software tools were developed to facilitate the institution's design, a textual specification language called ISLANDER was defined, and an infrastructure that mediates agent interactions and enforces the institutional rules on participating agents was developed. Another contribution of this group is the method for rapidly building prototypes of large multi-agent systems using logic programming [24]. This method advocates the use of all permitted interactions among the components of the systems. Our work is more concerned with structuring the whole development process of open MAS middleware, providing during this process some guidelines, structured as risks and dependability attributes.

Cole [6] proposes a way to identify laws in real world problems. However, his result does not deal with issues related to law enforcement and specification. In addition, Mineau [17] 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 but does not propose any enforcement mechanism.

6. Conclusions

Complexity and uncertainty are issues that must be analyzed during the development and evolution of open multi-agent systems. The concept of an open system is considered in terms of a dynamic set of interacting entities, where no single individual or organization is in control of the construction or, consequently, behavior of the set as a whole [11]. Thus, an open environment is uncertain, i.e., the same component that provided an answer to an earlier request may not be available when called upon again [22]. Moreover, open systems consist of many distributed, asynchronous components

that are open to interaction with their environment. The functionality of an open system is not defined by the result of evaluating an expression; instead, the relative state of components, the relative timing of actions, the locality and distribution of the computation, among others, are all critical to the correctness of the system [1]. These systems are populated by heterogeneous components, normally developed by different people using different languages and architectures [10].

This paper presented a method for open system infrastructure development. This method considers risk analysis concepts to provide a structured way to specify, implement, monitor and maintain systems requirements. Briefly, this analysis is based on the identification, control and assessment of relationships between cause and consequence states, events and their characteristics. Besides, our approach provides a mechanism to enforce laws that were previously specified through all the software agents that participate in the open system. In law enforcement architectures, there exists a mediator that monitors every interaction between agents and enforces the laws applying the specified consequences, whenever needed. Basically, the mediator works like an interceptor of all messages exchanged by the agents during their conversations.

As a future work, we intend to extend Anote notation [5] to represent the design decisions, including the specification of system requirements considering information about risks, interaction protocols, norms activation and deactivation, and any other adaptation that could provide a better understanding of the solution. Furthermore, we also intend to provide a conceptual framework to aid the assessment of existing alternatives of law specification and enforcement mechanisms considering functionality, performance, cost and dependability system properties using for this purpose a risk driven approach.

7. References

1. Agha, G.A. Abstracting Interaction Patterns: A Programming Paradigm for Open Distributed Systems, In (Eds) E. Najm and J.-B. Stefani, Formal Methods for Open Object-based Distributed Systems IFIP Transactions, Chapman & Hall, 1997.
2. Avizienis, A., Laprie, J.C., Randell, B. Fundamental Concepts of Dependability, Research Report N01145, LAAS-CNRS, April 2001
3. Bellifemine, F. Poggi, A., Rimassa, G. "JADE: a FIPA2000 compliant agent development environment", Proceedings of the fifth international conference on Autonomous agents, ACM Press : Montreal, Quebec, Canada, pp. 216-217, 2001.
4. Brazendale, J., Bell, R. (1994) Safety-related control and protection systems: standards update. IEE Computing and Control Engineering, 16(2), 238-247.
5. Choren, R., Lucena, C. Agent-Oriented Modeling Using ANote. Proceedings of the Third International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, SELMAS 2004, Edinburgh, Scotland, May 2004, p. 74-80.
6. Cole, J., Derrick, J., Milosevic, Z., Raymond, K. Policies in an Enterprise Specification, In Policies for Distributed Systems and Networks, Springer-Verlag: Lecture Notes in Computer Science, v. 1995, pp.1-17, 2001.
7. Esteva, M. PhD Thesis, Electronic Institutions: from specification to development, Institut d'Investigació en Intel·ligència Artificial, October 2003, Catalonia – SPAIN.
8. Fairley, R. Risk-based software estimation. Encyclopedia of Software Engineering. (ed. John J. Marciniak). New York, John Wiley & Sons. Vol 2, 2002, p.1227-1233.

9. FIPA, FIPA Contract Net Interaction Protocol Specification, <http://www.fipa.org/specs/fipa00029/>, December, 2002.
10. Fredriksson et al. (2003) First international workshop on theory and practice of open computational systems. In Proceedings of twelfth international workshop on Enabling technologies: Infrastructure for collaborative enterprises (WETICE), Workshop on Theory and practice of open computational systems (TAPOCS), pp. 355 - 358, IEEE Press.
11. Fredriksson, M., Gustavsson, R. A methodological perspective on engineering of agent societies. (Eds) A. Omicini and F. Zambonelli and R. Tolksdorf . In Engineering societies in the agents' world, Springer Verlag v. 2203, pp. 10-24, 2002.
12. Jones, A.J.I., Sergot M. "On the Characterisation of Law and Computer Systems: The Normative Systems Perspective". In Eds J.-J.Ch. Meyer and R.J. Wieringa, Deontic Logic in Computer Science: Normative System Specification, John Wiley and Sons, chapter 12, pp. 275-307, 1993
13. Laprie, J.C. et al. Architectural issues in fault tolerance. In Software Fault Tolerance (M. R. Lyu, ed.) Chichester: John and Sons, pp. 47-80, 1995.
14. Leishman, T., VanBuren, J. "The Risk of Not Being Risk Councious: Software Risk Managment Basic", STSC Seminar Series, Hill AFB, UT, 2003.
15. Leveson, N.G., Harvey, P.R. (1983). Analysing Software Safety. IEEE Trans. On Software Engineering, SE-9(5), 569-579.
16. Martín, F.J., Plaza, E., Rodriguez-Aguilar, J.A. An Infrastructure for Agent-Based Systems: an Interagent Approach, International Journal of Intelligent Systems (IJIS, 1999).
17. Mineau, G.W. Representing and Enforcing Interaction Protocols in Multi-Agent Systems: an Approach Based on Conceptual Graphs, IEEE/WIC International Conference on Intelligent Agent Technology, 2003.
18. Minsky, N.H., Ungureanu, V. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems, ACM Press, ACM Trans. Softw. Eng. Methodol., v.9, n.3, 2000, pp. 273-305.
19. Paes, R.B., Carvalho, G., Almeida, H.O, Lucena, C., Alencar, P.C.S. A conceptual architecture for law-governed open multi-agent systems. ASSE 2004 - Argentine Symposium on Software Engineering. Córdoba, Argentina. September, 2004
20. Rodriguez-Aguilar, J.A.. On the Design and Construction of Agent-mediated Electronic Institutions. PhDThesis. Institut d'Investigació en Intel.ligència Artificial. 2001, Catalonia - SPAIN.
21. Sommerville, I. Software Engineering. 7.ed. New York: Addison-Wesley, 2004. 759p.
22. Sycara, K., Giampapa, J.A., Langley, B.K., Paolucci, M. The RETSINA MAS, a Case Study, In (Eds) A. Garcia and et al; Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications. Springer-Verlag, v. 2603, pp. 232-250, 2003.
23. Thayer, R. H. Software System Engineering: A tutorial, IEEE Computer, April 2002
24. Vasconcelos W.; Robertson D.; Sierra C.; Esteva M.; Sabater J.; Wooldridge M. Rapid Prototyping of Large Multi-Agent Systems Through Logic Programming Annals of Mathematics and Artificial Intelligence. August 2004, vol. 41, no. 2-4, pp. 135-169(35).
25. Zambonelli, F., Jennings, N., Wooldridge, M. Developing multiagent systems: The Gaia methodology, In ACM Trans. Softw. Eng. Methodol., ACM Press, v. 12, n. 3, pp. 317-370, 2003