PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 29/05

Governing the Interactions of an Agent-based Open Supply Chain Management System

Gustavo Robichez de Carvalho
Rodrigo de Barros Paes
Carlos José Pereira de Lucena
Ricardo Choren

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. 29/05 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Agosto/August, 2005

Governing the Interactions of an Agent-based Open Supply Chain Management System

Gustavo Robichez de Carvalho[1], Rodrigo de Barros Paes[1],
Carlos José Pereira de Lucena[1], Ricardo Choren[2]

[1] Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro – Brasil, 22453-900

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

[2] Departamento de Engenharia de Sistemas - IME
Praça General Tibúrcio 80, Rio de Janeiro, Brasil, 22290-270

choren@de9.ime.br

Abstract. In open multi-agent systems, in which components are autonomous and heterogeneous, trust is crucial. This paper presents a law-governed mechanism to ensure trust and augment reliability in open systems. The mechanism is based on governing the interactions within the system. This is a non-intrusive method, which allows free development of agents for open systems – they must only follow the protocols specified for the system. This paper gives a complete example of the law government approach, using a real open system environment for supply chain management. In this paper, the XMLaw description language was extended to specify the characteristics of the Trading Agent Competition – Supply Chain Management (TAC SCM) domain and these new features have been mapped to a mechanism that interprets the descriptions and analyzes the compliance of the software agents that inhabit the open software system to its laws. Using this specification, a supply chain management application based on TAC-SCM's description was developed.

Keywords: Multi-agent systems, interaction protocols, open systems, laws, software engineering.

Resumo. Sistemas multi-agentes podem apresentar componentes autônomos e heterogêneos, portanto garantir confiabilidade é fundamental. Este artigo apresenta um mecanismo de governança de leis para garantir e aumentar a confiabilidade em sistemas abertos. Este mecanismo se baseia na governança de interações do sistema. Este é um método não intrusivo, que permite o desenvolvimento independente de agentes – eles precisam somente respeitar a especificação do protocolo. Este artigo apresenta um exemplo completo de uma abordagem de governança de leis, utilizando um sistema aberto do domínio de aplicação de gestão de cadeias de suprimentos. Neste artigo, a linguagem de descrição XMLaw foi estendida para atender aos requisitos do domínio representado pela aplicação TAC SCM (Trading Agent Competition – Supply Chain Management) e estas novas funcionalidades foram mapeadas para um mecanismo que interpreta as descrições e analisa a conformidade dos agentes de software que habitam o sistema aberto, segundo as leis previamente definidas. Utilizando esta especificação, um sistema de gestão de cadeia de suprimentos baseado na descrição de TAC – SCM foi desenvolvido.

Palavras-chave: Sistemas multi-agentes, protocolos de interação, sistemas abertos, leis, engenharia de software.

In charge for publications:

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
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introduction

Nowadays, openness is a characteristic that is crucial for software. Open systems are environments where autonomous distributed components interact to fulfill their goals. These components may enter and leave the environment freely, and they may even have conflicting interests [9]. Openness has led to software systems that have no centralized control [1] and thus can present an emergent or even unpredictable behavior. Multi-agent auction systems are examples of such open and distributed applications [3, 16].

Software agent technology is considered a promising approach for the development of such open applications due to properties such as autonomy, adaptation, distribution and heterogeneity that they possess [15]. The specification of open multi-agent systems (open MAS) includes the definition of agent roles and any other restrictions that the environment imposes on an agent to allow it to enter and participate in conversations.

Open system components are often autonomous; thus, sometimes they may behave unpredictably and unforeseen situations may arise. Agents are therefore faced with significant degrees of uncertainty for making decisions since it is very hard to devise all the possible situations that may arise in the execution context. Taming this uncertainty is a key issue for open software development. In such circumstances, trust is a strategy for dealing with uncertainty associated with interactions in open systems. In this work, trust in open software systems is achieved by the development of interaction laws. Interaction laws are restrictions imposed by the environment to tame uncertainty and to promote open system dependability [11], thus defining what and when something can happen in an open system. These laws must be enforced to delimit tolerated autonomous behavior and are also used to foster the development of trusted systems. The bottom line is that laws are used to represent the valid interactions in open MAS applications.

There are many approaches that can be used to govern open multi-agent systems based on laws [8, 10, 12]. Governance means that laws and any other form of specification can be enforced dynamically at application runtime. One of these approaches is the e-Governance [7]. This work presents a law-governed approach based

on the application of interaction laws expressed in the XMLaw description language [11], which allows for the runtime verification of agents compliance based on a law-governed mechanism.

To illustrate the applicability of our law-governed approach, we have developed a prototype based on the specification of the Trading Agent Competition - Supply Chain Management (TAC SCM) [4]. The TAC SCM provides some requirements in the form of rules concerning the interactions in the environment. In this example, we discuss how open MAS specifications can be represented as laws that structurally define the boundary among software agents in an interaction protocol. The goal of this study is to approach the TAC SCM structure by considering it an open system, and through the analysis of its description. We aim to specify the interaction concerns and then provide a means to verify the compliance of agents with the rules that govern the open system. The main purpose of the current investigation is not to contribute to TAC SCM evolution as a realistic open system for B2B trading, but rather to show that it is possible to productively specify, analyze and develop open software systems using interaction laws.

The paper's contributions are twofold: the extension of XMLaw and the implementation of a relevant application using this approach. The XMLaw description language was extended to specify the characteristics of TAC SCM domain and these improvements were mapped to a mechanism that interprets those descriptions and analyzes the compliance of software agents that inhabit the open software system with the rules of the environment. In this paper, the extensions of XMLaw include a better explanation and definition of usage semantics of permission, prohibition and obligation norms; norms now can be indirectly verifiable by introducing extra resources in a context (e.g. data structures or constraints), for instance, restrictions can disable norms temporarily by checking elements in the norm's context; a norm can be required to be deactivated to enter a scene or even to fire a transition; and it is possible to specify the maximum number of participants of a specific role in a scene.

Finally, a supply chain management application based on TAC-SCM's description [4] was developed. TAC SCM was chosen as a case study because it is a relevant, well-known 1large scale open system application based on independent agents that

1 At least within the agent community

can benefit from the use of proper engineering concepts for its specification and construction.

The paper is organized as follows. Section 2 describes the law-governed approach, its architecture and the XMLaw description language. In Section 3, we discuss the TAC-SCM 2005's edition requirements and rules. Section 4 briefly describes the 2005´s TAC SCM edition using our approach. Related work is described in Section 5. Finally, we evaluate this approach and describe some future work and our conclusions in Section 6.

2 Governing Interactions in Open Systems

In open MAS, distributed software agents are independently implemented, i.e., the development takes place without a centralized control. In order to achieve a coherent system, we assume that every agent developer may have an a priori access to the open system specification, including the protocol description and the interaction laws. In this scenario, law-governed architectures can be designed to guarantee that the specifications will be obeyed.

In this kind of architecture, a mediator is needed to intercept messages and interpret the laws that were previously described. In fact, to provide a scalable solution, this approach needs to consider the existence of a pool of mediators. As more clients interact within the system, additional mediator's instances can be added to improve throughput. The core of a law-governed approach is the mechanism used by the mediators to monitor the conversations between components. We have developed a software support [12] that when necessary permits the extending of this basic infrastructure to fulfill any open system requirements or interoperability concerns.

Besides monitoring the conversation of agents, we need to specify the interaction rules that will govern this mediation. In this section, we explain the description language XMLaw [11]. XMLaw is used to represent the interaction rules of an open system specification. Those rules are interpreted by a mechanism that at runtime is used to analyze the compliance of open MAS with interaction laws. We have made some extensions to XMLaw during this experiment and we explain below the resulting language.

XMLaw represents the structure and the relationships of important law elements (Fig 1), describing a law specification. Law elements are interrelated in a way that makes it possible to specify interaction protocols using time restrictions, norms or even time sensitive norms. The composition and interrelationship among law elements is accomplished through events (Fig 2). One law element can generate events to signal something to other elements. Likewise, other elements can sense events for many purposes – for instance, activating or deactivating themselves.

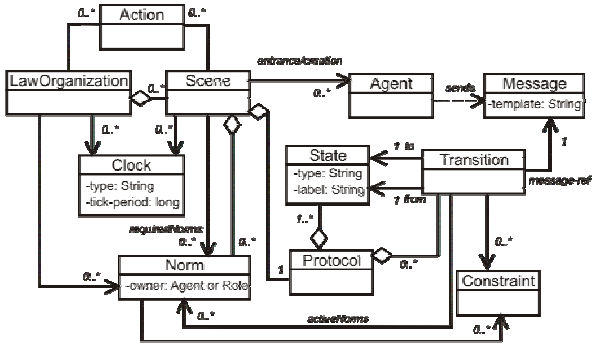

Fig 1 XMLaw Conceptual Model

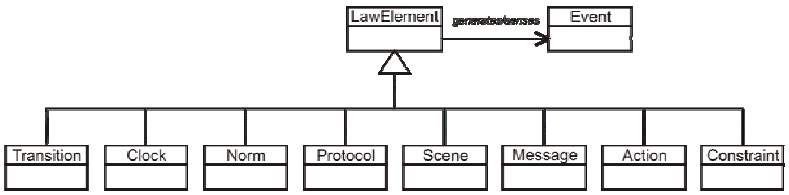

Fig 2 Law Elements and Events

Basically, we can understand the dynamic of law elements as a chain of events that are related as causes and consequences. An event can activate a law element; this law element can generate other events, and so on. For example (Fig 3), the arrival of a message activates a transition (transition activation event), which activates a clock (clock activation event), which generates a clock tick, and the clock tick activates a norm (norm activation event). Below, we give a detailed description of each law element in Fig 1 and its XMLaw structure to specify interactions in open MAS.

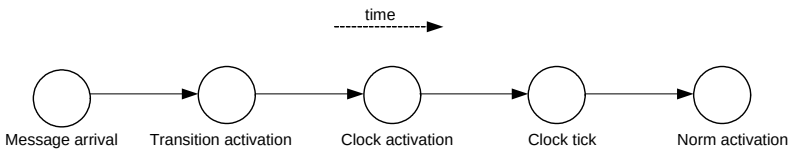

Fig 3 Chain of Events

2.1 Agents Communicating in a Scene

Inspired by electronic institutions [8], XMLaw uses scenes as abstractions to help organize interactions [11]. A scene defines a common interaction context to be shared by the agents. Scenes specify patterns of allowed interaction of agents. For instance, a scene may specify how agents must interact in an auction room. Scenes (Code Segment 1) are composed of the tags Creators, Messages, Protocol, Clocks, Actions, Norms, ActiveNorms and DeActivatedNorms.

A Scene has ActiveNorms and DeActivatedNorms elements representing all the norms that must be active or deactivated in order to enter into a scene. The tag Creators defines which agent roles can create an instance of this scene. The tag Entrance defines in which state of an interaction protocol an agent role can enter a scene execution. The tag Messages defines all possible message patterns. The tag Protocol specifies the interaction protocol for this scene. The tag Clocks defines all the clocks that are valid for this scene. The tag Norms contains the specification of the norms that can be triggered in this scene. Finally, the tag Actions brings the definition of all actions that can be triggered by events that occur in this scene.

```
<Scene id="anSceneID">
 <ActiveNorms>...</ActiveNorms>
 <DeActivatedNorms>...</DeActivatedNorms>
 <Creators>
      <Creator agent="any" role="any"/>
 </Creators>
 <Entrance>
      <Participant agent="any" role="anAgentRole">
         <States>...</States>
      </Participant>
 </Entrance>
 <Messages>...</Messages>
 <Protocol>...</Protocol>
 <Clocks>...</Clocks>
 <Norms>...</Norms>
 <Actions>...</Actions>
</Scene>
```

Code Segment 1 Scene Structure

Our law-governed approach considers an agent (Agent element) as any distributed entity able to communicate with other agents through the exchange of messages (Message element). In our model, an agent is described as attributes of scenes and it is pos-

sible to define the attribute role as an identification of an agent type in a law specification. An agent role represents how an agent is viewed at the organizational level. The Creators, Participant and Message elements make reference to agent roles (Code Segment 2). When a role attribute has the value "any", it means that any agent can be used in that context. Besides, it is possible to specify the maximum number of participants of a specific role in a scene using the attribute limit (Code Segment 2).

The Message element specifies all the patterns of messages that are used in the protocol specification. Protocols use these patterns to specify transitions. The template attribute is a message pattern that specifies the structure of instances of a valid Message element (Code Segment 2).

```
<Scene id="SceneID" time-to-live="infinity">
   <Creators>
      <Creator role="any"/>
   </Creators>
   <Entrance>
      <Participant role="aRole" limit="1">...</Participant>
   </Entrance>
   <Messages>
      <Message id="aMessageId"
         template="message( performative,
                           sender(SenderName,SenderRole),
                           receiver(ReceiverName,ReceiverRole),
                           content(SomeContent) )."/>
   </Messages>
   ...
</Scene>
```

Code Segment 2 References to agent roles

2.2 An Approach Based on State Machines

Interaction protocols define the valid interactions in which software agents participate and the context in which the exchanged information must be interpreted. We represent protocols as non-deterministic automata, where states represent snapshots of the protocol execution and transitions are the link between the states (Code Segment 3).

```
<Protocol>
  <States> ... </States>
  <Transitions> ... </Transitions>
</Protocol>
```

Code Segment 3 Protocol Structure

Protocol transitions are triggered by events – for instance, the message arrival or a clock-tick. It is important to notice that the attributes from and to (Code Segment 4) are ref-

erences to states specified in the States element (Code Segment 3), and the message-ref is also a reference to a message template specified in the Message element (Code Segment 2). A transition activation event is fired only if the message that originated the message arrival event satisfies the pattern specified in the template attribute (Code Segment 2); and all the required enabled norms are active (ActiveNorms tag in Code Segment 4); and all the required disabled norms are not active (DeActivatedNorms tag in Code Segment 4).

```
<Transition id="anId"
            from="anStateId"
            to="anotherStateId"
            message-ref="aMessageId">
   <ActiveNorms>
      <Norm ref="aNormId"/>
   </ActiveNorms>
   <DeActivatedNorms>
      <Norm ref="anotherNormId"/>
   </DeActivatedNorms>
</Transition>
```

Code Segment 4 Transition Structure

2.3 Time Sensitive Laws

Laws may be time sensitive; that is, although an agent may be able to perform an action a1 at time t1, it might not be able to perform the same action at time t2 (t1 < t2). XMLaw provides the Clock element to take care of the timing aspect. Temporal clocks represent time restrictions or controls and they can be used to activate other law elements. Clocks indicate that a certain period has elapsed producing clock-tick events.

A new clock instance is created each time a clock is activated. It means that the clock will start counting the time from its activation and will be independent of other active clocks. Once activated, a clock can generate clock-tick events. Clocks can be of two types: regular or periodic. Regular means that it will generate only one clock-tick after a tick-period of milliseconds has elapsed after the clock activation. Periodic clocks generate clock-ticks at the interval specified by the tick-period attribute.

Clocks are activated and deactivated by law elements, represented through the Activation and Deactivation tags (Code Segment 5). Both are referenced to other law elements, and they have two attributes: ref and event-type. The ref is a reference to an identification string of a law element and the event-type is the type of event generated by the referenced element. The event type is necessary because some elements generate more

than one type of event. Then, it is possible to specify exactly what event should start the clock with the event type attribute.

```
<Clock id="aClockId" type="regular" tick-period="1000">
  <Activations>
    <Element ref="anId" event-type="anEventType"/>
  </Activations>
  <Deactivations>
    <Element ref="otherId" event-type="anEventType"/>
  </Deactivations>
</Clock>
```

Code Segment 5 Clock Element Structure

2.4 Normative Interactions

Norms prescribe how distributed software components ought to behave, and specify how they are permitted to behave and what their rights are. Each norm element has an activation and a deactivation condition (Code Segment 6). Norm instances carry information about the context where the instance was generated. A context keeps information about the set of activated norms, the set of deactivated norms and any other data regarding the interaction execution. As a consequence of the relationship between norms and transitions, it is possible to specify which norms must be active or deactivated in order to fire a transition, in this sense; a transition should only fire if the sender agent has a specific norm.

In XMLaw, there are three types of norms: obligations, permissions and prohibitions. An obligation defines a commitment that software agents acquire while interacting with other entities. For instance, the winner of an auction is obligated to pay the committed value and this commitment indicates that the norm must not be broken. Permissions define the rights of a software agent in a given moment, e.g. the winner of an auction has permission to interact with a bank provider through a payment protocol. Finally, prohibitions define forbidden actions of a software agent at a given moment, such as if an agent does not pay its debts it will not be allowed future participations in a scene.

The structure of the Permission, Obligation and Prohibition elements are equal. Each type of norm contains activation and deactivation conditions. These conditions have the same semantic of the clock's conditions. Moreover, norms define the agent role that owns it through the attribute owner.

```
<Norms>
   <Permission id="aPermissionId">
      <Owner>AgentRole</Owner>
      <Activations>
         <Element ref="anId" event-type="anEventType"/>
      </Activations>
      <Deactivations>
        <Element ref="anotherId" event-type="anEventType">
      </Deactivations>
      <Actions> ... </Actions>
      <Constraints> ... </Constraints>
   </Permission>
   <Obligation id="anObligationId"> ... </Obligation>
   <Prohibition id="aProhibitionId"> ... </Prohibition>
</Norms>
```

Code Segment 6 Norm Structure

Statically, an interaction protocol defines the set of states and transitions (activated by messages or any other kind of event) allowed for components in an open system. Norms are used together with the protocol specification, constraints, actions and also temporal elements to provide a dynamical configuration for the allowed behavior of components in an open system. Below, we describe the enforcement consequences of permissions, prohibitions and obligations to the execution of a protocol.

A permission norm identifies a new trajectory that agents can choose to execute if it wants. For instance, if a permission norm is granted to an assembler agent to submit a request for quotation at a specific moment, this entity may request it to the supplier while this condition is valid. In XMLaw, permissions are referenced in ActiveNorms tags (Code Segment 4). In the ActiveNorms tag (Code Segment 1) of scenes, permission instances indicate a condition for agents to enter or create scenes.

On the other side, a prohibition norm eliminates a possible trajectory reducing the options that an agent must execute. For instance, if a prohibition is granted to the agent to submit new quotations, it could not request this kind of information to the supplier agent while this condition is valid. In XMLaw, prohibitions are referenced in DeActivatedNorms tags (Code Segment 4). And in the DeActiveNorms tag (Code Segment 1) of scenes, prohibition instances indicate that an agent cannot enter or create scenes.

Composition of norms can be achieved using an obligation since obligations can force an agent to execute a specific trajectory. Suppose that in a specific instant an agent has three trajectories from which to choose. After some interactions an obligation is granted to him and this obligation can imply prohibitions that eliminate some of

agent's alternatives. In this sense, all other alternatives that do not contribute to this commitment are prohibited until this duty is done. In XMLaw, obligations are referenced in both ActiveNorms and DeActivatedNorms tags (Code Segment 4, Code Segment 1).

2.5 Constraining Events

Constraints are restrictions over norms or transitions and generally specify the allowed values for a specific attribute of an event. For instance, messages carry information and laws can be enforced on information in several ways. The message pattern enforces the "shape" of messages, constraining the message structure fields. However, message pattern does not describe what the allowed values for specific attributes are. In this scenario, constraints are used to verify a condition regarding attributes of a message.

Constraints are expressed using Java code. In this way, developers are free to build as complex constraints as needed for their applications. Constraints are defined inside the Norm (Code Segment 8) or Transition (Code Segment 7) tag. The Constraint element defines the class attribute that indicates the java class that implements the constraint. This class is called when a transition or a norm is supposed to fire, and basically the constraint analyzes if the message values or any other events' attributes are valid.

```
<Transition id=”a-transition-id”
            from=”from-state”
            to=”to-state”
            message-ref=”rfq”>
   <Constraint id=”constraintID” class=”ajavapackage.AConstraintClass”/>
</Transition>
```

Code Segment 7 Constraint Structure in Transition Tag

```
<Permission id="a-Permission-Id">
   <Owner>...</Owner>
   <Activations>...</Activations>
   <DeActivations>...</DeActivations>
   <Constraints>
      <Constraint id="constraintID" class="aClass"/>
   </Constraints>
   <Actions>...</Actions>
</Permission>
```

Code Segment 8 Constraint Structure in Norm Tag

2.6 Acting in the Environment

Environmental actions are domain-specific Java code that run integrated with XMLaw specifications. Since Actions are also law elements, they can be activated by any event such as transition activation, norm activation, clock activation, and even action activa-

tion. XMLaw specifies Action elements as shown in Code Segment 9. The class attribute specifies the java class in charge of the functionality implementation. The Element tag makes references for the events that activate this action, and many Element tags as needed can be defined in an action.

```
<Actions>
  <Action id="anActionId" class="apackage.ActionClass">
    <Element ref="generatorReference" event-type="type"/>
    <Element ref="anotherGeneratorReference" event-type="aType"/>
  </Action>
</Actions>
```

Code Segment 9 Action Structure

3 Trading Agent Competition – Supply Chain Management

We based our proof of concept prototype on the specification of the 2005's Trading Agent Competition - Supply Chain Management (TAC SCM). We reuse this description [5] as a means to obtain the supply chain knowledge required for proposing the interaction rules that govern an open system in this domain.

The TAC SCM System [3,5] has been designed with a simple set of rules to capture the complexity of a dynamic supply chain. SCM applications are composed of planning and coordination activities related to a supply chain. These activities involve different participants and organizations, and their coordination is a key factor for shippers, distribution centers, suppliers and clients. This chain is extremely dynamic and involves an important number of products, information and resources throughout their different stages.

Six assembler agents that produce Personal Computers (PCs) participate in each competition round of TAC SCM. These participants compete with each other and have to deal with the uncertainty of customer demands and also with a limited number of components that are offered by eight suppliers. Assembler agents need to negotiate with supplier agents to buy components to produce PCs. A bank agent is used to monitor the progress of the participants and to determine the most profitable agent at the end of the competition.

This competition represents an interesting experiment with some complexities concerning the interaction among those different types of agents. Every negotiation market available in this application has interdependencies and uncertainties. This sce-

nario permits the establishment of different strategies for each assembler to achieve better results and to tame the unpredictability of the interactions in the open system. The rules of the game have been updated over the last three years. This evolution was achieved by the observation of the behavior of different agents during the last editions and the consequences of their actions. For instance, some interaction rules aim to protect agents from malicious participants. In the real TAC SCM architecture, there is a TAC Server that simulates the behavior of the suppliers, customers and bank.

In our prototype, we converted part of the simulation components present in TAC SCM to external agents. We continue to have the TAC SCM Server, but this server aims to monitor and to analyze the compliance of agents' behavior to laws that were previously established. Analyzing the TAC SCM's requirements, we can perceive evidences that interaction protocols have a set of interaction rules, and in our prototype, interaction rules were converted into laws.

We chose the scenario of negotiation between the suppliers and assemblers to explain our approach. This negotiation process involves an assembler agent that buys components from suppliers. The bank is also present in this scenario because an assembler must pay the components for the supplier. In this scenario, an assembler may send RFQs to each supplier everyday to order components offered by the supplier. Each RFQ represents a request for a specified quantity of a particular component type to be delivered on a date in the future [5]. The supplier collects all RFQs received during the "day", and processes them. On the following "day", the supplier sends an offer for each RFQ back to each agent, containing the price, adjusted quantity, and due date [5]. Offers made by suppliers are valid only during the "day" they arrive. If the agent wishes to accept an offer, it must confirm it by issuing an order to the supplier.

4 An Agent-based Open Supply Chain Management System

There are six assembler agents that produce PCs participating in each TAC SCM instance (Fig 4). Fig 4 is based on ANote's agent class diagram [4]. These participants interact with suppliers and with a bank agent. Four different components are necessary to assemble a PC. Each component is available in two different models. Thus, we have eight supplier agents. Only one bank agent is responsible for managing payment ac-

counts. Assembler role interacts with supplier role. The bank role interacts with both suppliers and assemblers.

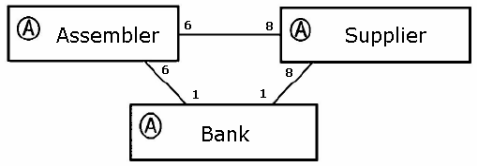

Fig 4 Agent roles for TAC SCM prototype

4.1 Scene Specification

The design decision was to organize the scenario into two Scenes, one for the negotiation process between assemblers and suppliers and the other for the payment process involving the assembler and the bank agent. Code Segment 10 details the initial specification of the scene that represents the negotiation between the supplier and the assembler. Each negotiation scene is valid over the duration of the competition, which is 3300000ms (220 days x 15000ms).

Code Segment 11 describes the payment process. We decided not to specify any time out to the payment scene and this is represented by the "infinity" value assigned by the attribute time-to-live. The assembler agent can create the negotiation scene and we do not specify any restriction regarding who can create the scene payment, so any agent can have the initiative to create it.

```
<Scene id="negotiation" time-to-live="3300000">
   <Creators>
      <Creator role="assembler"/>
   </Creators>
   <Entrance>
      <Participant role="assembler" limit="6"/>
      <Participant role="supplier" limit="8"/>
   </Entrance>
</Scene>
```

Code Segment 10 Roles, relationships and cardinalities of negotiation scene

```
<Scene id="payment" time-to-live="infinity">
   <Creators>
      <Creator role="any"/>
   </Creators>
   <Entrance>
      <Participant role="assembler" limit="1"/>
      <Participant role="bank" limit="1"/>
   </Entrance>
</Scene>
```

Code Segment 11 Roles, relationships and cardinalities of payment scene

4.2 Interaction Protocol Specification

The negotiation between assemblers and suppliers is related to the interaction between the assembler role and the bank role. Basically, a payment is made through a payment message sent by the assembler to the bank and the bank reply with confirmation response represented by the receipt message (Fig 5). The specification in XMLaw of this interaction is listed below (Code Segment 12, Code Segment 13). Fig 5 is based on ANote's interaction diagram [4] and Fig 6 represents the correspondent state machine of the payment interaction protocol.

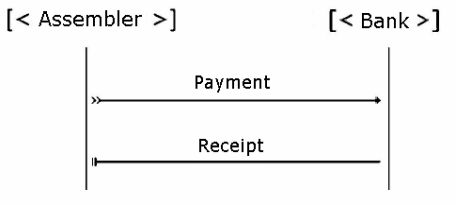

Fig 5 Payment interaction

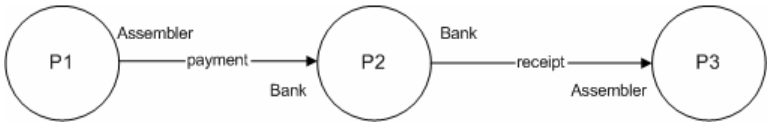

Fig 6 Payment (assembler, bank) Interaction Protocol Representation

```
<Messages>
   <Message id="payment" template="message(inform,
                           sender(RFQRequester,assembler),
                           receiver(TACBank,bank),
                           content(payment(
                                   order(Id),
                                   value(Value)))."/>
   <Message id="receipt" template="message(inform,
                           sender(TACBank,bank),
                           receiver(RFQRequester,assembler),
                           content(receipt(
                                   number(ReceiptNumber)))."/>
</Messages>
```

Code Segment 12 Payment (assembler, bank) messages

```
<Protocol>
  <States>
    <State id="p1" type="initial" label="Initial state"/>
    <State id="p2" type="execution"
                   label="Confirmation pending state"/>
    <State id="p3" type="success" label="Done state"/>
  </States>
  <Transitions>
     <Transition id="payingTransition" from="p1" to="p2"
                 message-ref="payment"/>
     <Transition id="paymentConcludedTransition" from="p2" to="p3"
                 message-ref="receipt"/>
  </Transitions>
</Protocol>
```

Code Segment 13 Payment (assembler, bank) interaction protocol description

The negotiation between assemblers and suppliers takes place in five steps (Code Segment 15), with four messages (Code Segment 14, Fig 7) and six transitions (Code Segment 16). In this scene, the assembler first sends a request for a quotation message to the supplier detailing the product information, including the component type, quantity and due date. An assembler can send other requests for quotation messages while deciding which one to buy. After analyzing the received requests, a supplier submits an offer message that includes the quantity, the price and the due date for delivering the components. After receiving the proposal, an assembler sends an order considering the offer and the assembler's actual storage and the demands of PCs. After receiving the order, a supplier commits to deliver the components. Below (Code Segment 15, Code Segment 14, Code Segment 16), we describe this scene in details using XMLaw. Fig 7 is based on ANote's interaction diagram [4] and Fig 8 represents the state machine of the payment interaction protocol.

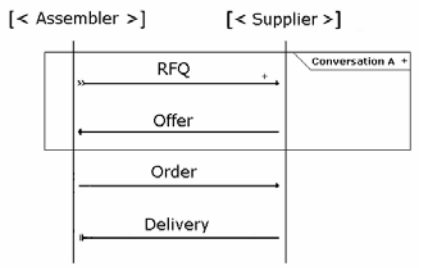

Fig 7 Negotiation interaction diagram

```
<Messages>
   <Message id="rfq" template="message(cfp,
            sender(Assembler,assembler),
            receiver(Supplier,supplier),
            content(product-details(rfqId(Id),
            component(Type),quantity(Quantity),dueDate(Date))))."/>
   <Message id="offer" template="message(propose,
            sender(Supplier,supplier),
            receiver(Assembler,assembler),
            content(product-details(offerId(Id),
                  rfqId(Rid),quantity(Quantity),
                  unitPrice(Price),dueDate(Date))))."/>
   <Message id="order" template="message(accept-proposal,
            sender(Assembler,assembler),
            receiver(Supplier,supplier),
            content(product-details(offerId(Id),rfqId(Rid))))."/>
   <Message id="delivery" template="message(inform,
            sender(Supplier,supplier),
            receiver(Assembler,assembler),
            content(product-details(orderId(Id))))."/>
</Messages>
```

Code Segment 14 Negotiation (assembler, supplier): Messages

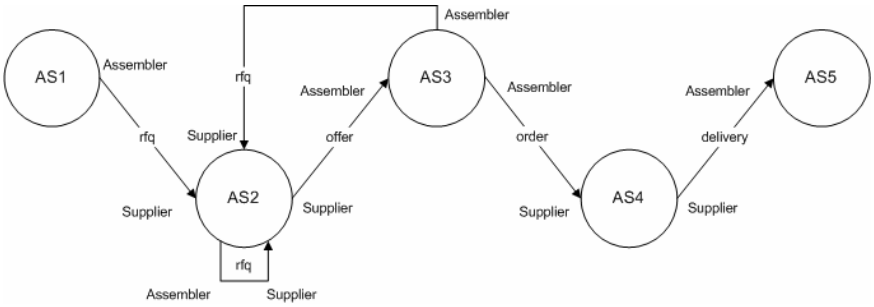

Fig 8 Negotiation (assembler, supplier) Interaction Protocol Representation

```
<States>
   <State id="as1" type="initial" label="Initial state"/>
   <State id="as2" type="execution" label="Waiting Quotations"/>
   <State id="as3" type="execution" label="Waiting Orders"/>
   <State id="as4" type="execution" label="Producing "/>
   <State id="as5" type="success" label="Done state"/>
</States>
```

Code Segment 15 Negotiation (assembler, supplier): States

```
<Protocol>
   <States>...</States>
   <Transitions>
      <Transition id="rfqTransition" from="as1" to="as2"
                  message-ref="rfq">...</Transition>
      <Transition id="newRFQTransition" from="as2" to="as2"
                  message-ref="rfq">...</Transition>
      <Transition id="otherRFQTransition" from="as3" to="as2"
                  message-ref="rfq">...</Transition>
      <Transition id="offerTransition" from="as2" to="as3"
                  message-ref="offer">...</Transition>
      <Transition id="orderTransition" from="as3" to="as4"
                  message-ref="order"/>
      <Transition id="deliveryTransition"
                  from="as4" to="as5"
                  message-ref="delivery">...</Transition>
   </Transitions>
</Protocol>
```

Code Segment 16 Negotiation (assembler, supplier):Transitions

After specifying the main elements and the structure of the scenes, it is possible to define in which state the agent roles will enter the open MAS. In this example, the assembler will enter the process in the state as1 of scene negotiation and p1 of scene payment, the supplier in the state as2 of scene negotiation and the bank in the state p2 of scene payment (Code Segment 17, Code Segment 18).

```
<Scene id="payment" time-to-live="infinity">
   <Creators>
      <Creator role="any"/>
   </Creators>
   <Entrance>
      <Participant role="assembler" limit="1">
         <State ref="p1"/>
      </Participant>
      <Participant role="bank" limit="1">
         <State ref="p2"/>
      </Participant>
   </Entrance> ...
</Scene>
```

Code Segment 17 Refinement of payment scene: Participant Entrance

```
<Scene id="negotiation" time-to-live="3300000">
   <Creators>
      <Creator role="assembler"/>
   </Creators>
   <Entrance>
      <Participant role="assembler" limit="6">
         <State ref="as1"/>
      </Participant>
      <Participant role="supplier" limit="8">
         <State ref="as2"/>
      </Participant>
   </Entrance>
   ...
</Scene>
```

Code Segment 18 Refinement of negotiation scene: Participant Entrance

4.3 Norm Specification

To illustrate the use of norms in TAC SCM, we implemented using XMLaw the relation between a request for quote (RFQ) sent by an assembler and an offer that will be sent by a supplier. Below, we briefly describe the specification according to [5]:

On the day following of the arrival of a request for quotation, the supplier sends back to each agent an offer for each RFQ, containing the price, adjusted quantity and due date. It is possible that the supplier will not be able to supply the entire quantity requested in the RFQ by the due date, even if the reserve price does not constrain the quantity. In this situation, the supplier may respond by issuing up to two amended offers, each of which relaxes one of the two constraints, quantity and due date: (i) a partial offer is generated if the supplier can deliver only part of the requested quantity on the due date specified in the RFQ (quantity relaxed); or (ii) an earliest complete offer is generated to reject the earliest day that the supplier can deliver the entire quantity requested (due date relaxed).

Offers are received the day following the submission of RFQs, and the assembler must choose whether to accept them. In the case an agent attempts to order both the partial offer and the earliest complete offer, only the order that arrives earlier will be considered and the others will be ignored

The implementation of this rule in XMLaw is illustrated in Code Segment 19, Code Segment 20 and Code Segment 21. A permission was created to define a context in the conversation that is used to control when the offer message is valid considering the information sent by an RFQ. For this purpose, two constraints were defined into the permission context, one determining the possible configurations of offer attributes that a

supplier can send to an assembler. Furthermore, a verification is made by a constraint to guarantee that a valid offer message was generated; that is, the offer is sent one day after the RFQ. This permission is only valid if both the constraints are true. There are three transitions that have the same structure: rfqTransition, newRFQTransition and otherRFQTransition. Below, we only illustrate the rfqTransition and describe its XMLaw specification (Code Segment 19).

```
<Transition id="rfqTransition" from="as1" to="as2"
            message-ref="rfq">
   <ActiveNorms>
      <Norm ref="AssemblerPermissionRFQ"/>
   </ActiveNorms>
</Transition>
<Transition id="offerTransition" from="as2" to="as3"
            message-ref="offer">
   <ActiveNorms>
      <Norm ref="RestrictOfferValues"/>
   </ActiveNorms>
</Transition>
```

Code Segment 19 Transition Specification

In this experiment, we have extended the XMLaw to include the context concept. A context is the locus where the law element will act. Elements in the same context use the same local memory to share information, i.e., putting, getting and updating any value that is important for other law elements. For a better presentation, we split the one example of context usage in Code Segment 20 and Code Segment 21. The keepRFQInfo Action preserves the information present in the rfq message to be later used by the checkAttributes and checkDates Contraints.

```
<Permission id="RestrictOfferValues">
 <Owner>Supplier</Owner>
  <Activations>
  <Element ref="rfqTransition" event-type="transition_activation"/>
  <Element ref="otherRFQTransition"
           event-type="transition_activation"/>
  <Element ref="newRFQTransition" event-type="transition_activation"/>
  </Activations>
  <Deactivations>
   <Element ref="offerTransition" event-type="transition_activation"/>
  </Deactivations>
  <Actions> ...</Actions>
  <Constraints> ...</Constraints>
</Permission>
```

Code Segment 20 Norm structure specification

```
<Permission id="RestrictOfferValues">
       ...
<Actions>
 <Action id="keepRFQInfo" class="tacscm.norm.actions.KeepRFQAction">
  <Element ref="rfqTransition" event-type="transition_activation"/>
  <Element ref="otherRFQTransition"
           event-type="transition_activation"/>
  <Element ref="newRFQTransition" event-type="transition_activation"/>
 </Action>
</Actions>
<Constraints>
  <Constraint id="checkDates"
              class="tacscm.norm.constraints.CheckValidDay"/>
 <Constraint id="checkAttributes"
             class="tacscm.norm.constraints.CheckValidMessage"/>
</Constraints>
</Permission>
```

Code Segment 21 Actions and Constraints specification

Now, we detail the permission on the maximum number of requests for quotations that an assembler can submit to a supplier. According to TAC SCM specification [4], each day, each agent may send up to a maximum number of RFQs. Besides this permission, the constraint over the acceptable due date of a RFQ (checkCounter) regulates the same interaction point, the request for quote message. The constraint checkDueDate (Code Segment 22) is associated with the transition rfqTransition. It means that if the verification is not true the transition will not be fired.

```
<Transition id="rfqTransition" from="as1" to="as2"
            message-ref="rfq">
   <Constraints>
      <Constraint id="checkDueDate"/>
   </Constraints>
   <ActiveNorms>
      <Norm ref="AssemblerPermissionRFQ"/>
   </ActiveNorms>
</Transition>
```

Code Segment 22 Permission and Constraint over RFQ message

The constraint checkCounter (Code Segment 23) is associated with the permission AssemblerPermissionRFQ. It means that if the verification is not true the norm will not be valid, even if it is activated. The action ZeroCounter (Code Segment 23) is defined under the permission AssemblerPermissionRFQ and it is triggered by a clock-tick everyday, zeroing the value of the counter of the number of requests issued by the assembler during this day. The other action orderID (Code Segment 23) is activated by every transition transitionRFQ and is used to count the number of RFQs issued by the assembler, updating a local counter.

```
<Permission id="AssemblerPermissionRFQ">
   <Owner>Assembler</Owner>
   <Activations>
      <Element ref="negotiation" event-type="scene_creation"/>
   </Activations>
   <Deactivations>
       <Element ref="orderTransition"
                event-type="transition_activation"/>
   </Deactivations>
   <Constraints>
      <Constraint id="checkCounter"/>
   </Constraints>
  <Actions>
   <Action id="permissionRenew"
           class="tacscm.norm.actions.ZeroCounter">
       <Element ref="nextDay" event-type="clock_tick"/>
   </Action>
   <Action id="orderID">
     <Element ref="rfqTransition" event-type="transition_activation"/>
   </Action>
  </Actions>
</Permission>
```

Code Segment 23 Norm description

Finally, a clock nextDay (Code Segment 24) is defined to mark the day period, and this mark is used to zero the counter of RFQs by the action ZeroCounter (Code Segment 23).

```
<Clocks>
 <Clock id="nextDay" type="periodic" tick-period="100000000">
  <Activations>
   <Element ref="negotiation" event-type="scene_creation"/>
  </Activations>
  <Deactivations>
    <Element ref="negotiation"
             event-type="sucessful_scene_completion"/>
  </Deactivations>
 </Clock>
</Clocks>
```

Code Segment 24 Clock Description

Another example of law is used to specify the relationship between orders and offers of the negotiation protocol. According to [5], agents confirm supplier offers by issuing orders. After that, an assembler has a commitment with a supplier, and this commitment is expressed as an obligation. It is expected that suppliers receive a payment for its components. This requirement specifies the structure of the ObligationToPay obligation (Code Segment 28), defining that it will be activated by an order message and that it will be deactivated with the delivery of the components and also with the payment.

A supplier will only deliver the product if the assembler has the obligation to pay for them (Code Segment 25). The assembler can only enter into the payment scene if it has an obligation to pay for the products (Code Segment 26). An assembler cannot enter into another negotiation scene if it has obligations that were not fulfilled (Code Segment 27).

```
<Transition id="orderTransition" from="as3" to="as4"
            message-ref="order"/>
<Transition id="deliveryTransition" from="as4" to="as5"
            message-ref="delivery">
   <ActiveNorms>
      <Norm ref="ObligationToPay"/>
   </ActiveNorms>
</Transition>
```

Code Segment 25 Law that impacts the Negotiation Scene and the Payment Scene

```
<Scene id="payment" time-to-live="infinity">
   <ActiveNorms>
      <Norm ref="ObligationToPay"/>
   </ActiveNorms>
   ...
</Scene>
```

Code Segment 26 Agents must have the norm to create the payment scene

```
<Scene id="negotiation" time-to-live="3300000">
   <DeActivatedNorms>
      <Norm ref="ObligationToPay"/>
   </DeActivatedNorms>
   ...
</Scene>
```

Code Segment 27 Agents must not have the norm to create the negotiation scene

```
<Norms>
   <Obligation id="ObligationToPay">
      <Owner>Assembler</Owner>
      <Activations>
         <Element ref="orderTransition"
                  event-type="transition_activation"/>
      </Activations>
      <Deactivations>
          <Element ref="payingTransition"
                   event-type="transition_activation"/>
      </Deactivations>
   </Obligation>
</Norms>
```

Code Segment 28 Norms of the organization

4.4 Actions and Constraints – Norms' Refinement

In this prototypical version, we considered the source of assemblers and suppliers agents as unknown. So these two roles will only be fulfilled during the execution of the

open system. Below, we present the refinements proposed to the law described above. According to [5]:

1. each day, each agent may send up to five RFQs to each supplier for each of the products offered by that supplier, for a total of ten RFQs per supplier. Another action component named RFQCounter2005 is plugged-in (Code Segment 30). It counts the number of RFQs according to the type of component. The CounterLimit2005 also chooses a specific counter for each type of component that a supplier provides;

2. an RFQ with DueDate beyond the end of the game will not be considered by the supplier. RFQs with due dates beyond the end of the game, or with due dates earlier than 2 days in the future, will not be considered. It is implemented by the constraint ValiDate2005 (Code Segment 30).

```
<Transition id="rfqTransition"
            from="as1" to="as2"
            message-ref="rfq">
   <Constraints>
      <Constraint id="checkDueDate"
           class="tacscm.constraints.ValiDate2005"/>
   </Constraints>
   ...
</Transition>
```

Code Segment 29 Constraint checkDueDate instance for TAC SCM 2005

```
<Permission id="AssemblerPermissionRFQ">
 <Constraints>
   <Constraint id="checkCounter"
              class="tacscm.norm.constraints.CounterLimit2005"/>
 </Constraints>
 <Actions>
   <Action id="orderID"
          class="tacscm.norm.actions.RFQCounter2005">...</Action>
 </Actions>
</Permission>
```

Code Segment 30 Permission AssemblerPermissionRFQ instance for TAC SCM 2005

According to [5], suppliers wishing to protect themselves from defaults will bill agents immediately for a portion of the cost of each order placed. The remainder of the value of the order will be billed when the order is shipped. In TAC SCM 2005, the down payment ratio is 10%. This down payment is implemented by the action SupplierPayment (Code Segment 31).

```
<Obligation id="ObligationToPay">
  <Owner>Assembler</Owner>
  <Activations>
  <Element ref="orderTransition"
           event-type="transition_activation"/>
  </Activations>
  <Deactivations>
   <Element ref="payingTransition"
            event-type="transition_activation"/>
  </Deactivations>
  <Actions>
  <Action id="supplierPayment"
          class="tacscm.norm.actions.SupplierPayment">
    <Element ref="orderTransition"
             event-type="transition_activation"/>
  </Action>
 </Actions>
</Obligation>
```

Code Segment 31 ObligationToPay instance for TAC SCM 2005

5 Related Work

It is possible to cite at least three important research works with goals very similar to the conceptual model used here. In Esteva approach [8], scenes are similar to the protocol elements proposed in this paper. Both Esteva scenes and protocol elements specify the interaction protocol using a global view of the interaction. It means that all the interaction among the agents is specified in only one protocol as opposed to individual agent views, where many partial views of the protocol (one of each agent) are specified. The time aspect is represented in 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 this paper could 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, we also include the concept of actions, which allows execution of java code in response to some interaction situation.

OMNI [14] is a framework for modeling agent organizations. This framework is composed of three dimensions: normative, organizational and ontological. These dimensions aim to cover from analysis to implementation of agent organizations. In the normative dimension, developers specify the mechanisms of social order, in terms of common norms and rules, to which members are expected to adhere. The organizational dimension describes the structure of an organization. The ontological dimension

defines environment and contextual relations and communication aspects in organizations. In addition, each one of these dimensions can be viewed in three abstraction levels: abstract, concrete and implementation. In the abstract level, the general organization goals are defined in a high level of abstraction. It also contains the definition of the ontology of the model itself. Based on the abstract level, norms, rules, roles, interaction protocols and concrete ontological concepts are defined. The implementation level assumes a given multi-agent architecture as basis for the implementation of the organizational model, and also mechanisms for role enactment and norm enforcement.

Minsky [2, 10] proposes a coordination and control mechanism called law-governed interaction (LGI). This mechanism is based upon two basic principles: the local nature of the LGI laws and a 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 **α** can depend only on **α** 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. Furthermore, it provides a language to specify laws and it is concerned with architectural decisions to achieve a high degree of robustness. In contrast, our approach provides an explicit conceptual model and focuses on different concepts such as Scenes, Norms and Clocks.

6 Conclusions

Trust is a belief an agent has that the other party will act as expected [6]. In open multi-agent systems, in which components are autonomous and heterogeneous, trust is crucial. This paper presented a law-governed mechanism to ensure trust and augment reliability on open systems and the implementation of an agent-based open supply chain management system using this approach. The mechanism is based on governing the interactions in the system. This is a non-intrusive method, which allows the independent development of the agents of the open system – they are only required to follow the protocols specified for the system.

We presented a conceptual model of law elements and a language to specify these laws, the XMLaw. Law elements are the constituent parts of the laws and can

comprise actions, clocks, constraints, norms, protocols, scenes and transitions. The contributions of this paper include the extension of XMLaw including a better explanation and definition of permission, prohibition and obligation norms; norms can be indirectly verifiable by introducing extra resources in a context (e.g. data structures or constraints); restrictions can temporarily disable norms; a norm can be required to be deactivated to enter into a scene or even to fire a transition; and it is possible to specify the maximum number of participants of a specific role in a scene.

This paper offered a complete example of the law government approach, using a real open system environment for supply chain management, the TAC SCM. We showed some scene specifications, which includes the specification of every element agents, protocols, constraints, norms, and so on. This case study intends to illustrate how our approach can be used. This first real size case study development using law-government showed that this is an interesting and promising approach; it improves the open system design, by incorporating reliability aspects. The application development experience showed us that it is possible to obtain benefits from the use of proper engineering concepts for its specification and construction. However, more experiments with real-life MAS applications are needed to evaluate and validate the proposed approach.

Acknowledgments

We gratefully acknowledge the financial support provided by the CNPq as part of individual grants and of the ESSMA project (552068/2002-0).

7 References

1. G. A. Agha. 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. X. Ao and N. Minsky. Flexible Regulation of Distributed Coalitions. In Proc. of the 8th European Symposium on Research in Computer Security (ESORICS). Gjøvik Norway, October, 2003.

3. R Arunachalam, N. Sadeh, J. Eriksson, N. Finne, and S. Janson, S. The Supply Chain Management Game for the Trading Agent Competition 2004. CMU-CS-04-107, July 2004

4. R. Choren and C.J.P. Lucena. Modeling Multi-agent systems with ANote. Software and Systems Modeling 4(2), 2005, p. 199 - 208.

5. J. Collins, R. Arunachala, N. Sadeh, J. Eriksson, N. Finne and S. Janson. The Supply Chain Management Game for the 2005 Trading Agent Competition. CMU-ISRI-04-139. http://www.sics.se/tac/tac05scmspec_v157.pdf , 2005.

6. P. Dasgupta. Trust as a commodity. In D. Gambetta, editor, Trust: Making and Breaking Cooperative Relations, pages 49–72. Blackwell, 1998.

7. e-Governance homepage . http://www.les.inf.puc-rio.br/governance

8. M. Esteva. Electronic institutions: from specification to development, Ph.D. thesis, Institut d'Investigació en Intelligència Artificial, Catalonia – Spain, 2003.

9. M. Fredriksson et al. 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), 2003, pp. 355 - 358, IEEE Press.

10. N. H. Minsky and V. Ungureanu Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems, ACMTrans. Softw. Eng. Methodol. 9 (3) 273–305, 2000.

11. R. Paes, G. R. Carvalho, C.J.P. Lucena, P. S. C. Alencar, H.O. Almeida; and V. T. Silva. Specifying Laws in Open Multi-Agent Systems. In: Agents, Norms and Institutions for Regulated Multi-agent Systems (ANIREM), AAMAS2005, 2005.

12. R. Paes, C.J.P. Lucena and P.S.C. Alencar. A Mechanism for Governing Agent Interaction in Open Multi-Agent Systems, 2005. http://www.les.inf.puc-rio.br/governance/pubs.html

13. N. Sadeh, R. Arunachalam, J. Eriksson, N. Finne and S. Janson. TAC-03: a supply-chain trading competition, AI Mag. 24 (1) 92–94, 2003.

14. J. Vázquez-Salceda, V. Dignum and F. Dignum. Organizing multiagent systems, Tech. rep., Institute of Information & Computing Sciences (March 2004).

15. M. Wooldridge, G. Weiss, P. Ciancarini (Eds.) Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001, Revised Papers and Invited Contributions, Vol. 2222 of Lecture Notes in Computer Science, Springer.

16. F. Zambonelli, N. Jennings and M. Wooldridge. Developing multiagent systems: The gaia methodology, ACM Trans. Softw. Eng. Methodol. 12 (3) 317–370, 2003.