PUC

ISSN 0103-9741

Monografias em Ciência da Computação

n° 33/05

A Governance Framework for Instantiating Supply Chain Management Applications as Open Systems

Gustavo Robichez de Carvalho
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

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

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 33/05 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Outubro/ October, 2005

A Governance Framework for Instantiating Supply Chain Management Applications as Open Systems

Gustavo Robichez de Carvalho, Carlos José Pereira de Lucena
Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro – Brasil, 22453-900

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

Abstract. Governance means that specifications are enforced dynamically at application run-time. Governance framework is a technique to design open systems for extensions. We based this proposal on object-oriented framework concepts and adapted them for distributed agents and interactions. A governance framework reuses general specifications and some infrastructural services provided by open MAS. Governance frameworks structure the extensions of open system instances as variations in components, defined as roles, and variations in interactions among agents, defined as templates. Templates are used to gather a core implementation and extension points. Extension points are "hooks" that will be customized to implement an instance of the governance framework. During framework instantiation, roles are bound to external agents and templates are refined to concrete interaction specification. As a proof of concept experiment, in this paper we propose a framework for instantiating supply chain management applications as open systems.

Keywords: Reuse, frameworks, multi-agent systems, interaction protocols, open systems, laws, software engineering.

Resumo. Em um sistema aberto, governança significa que especificações são verificadas dinamicamente em tempo de execução. Frameworks de governança é uma técnica para projeto sistemas abertos para extensão. Baseamos esta proposta em conceitos de frameworks orientado a objetos e os adaptamos para desenvolver sistemas abertoc com agentes distribuídos. Um framework de governança reutiliza especificações gerais e alguns serviços de infra-estrutura oferecidos por um sistema multi-agente aberto. Frameworks de governança estruturam as extensões de uma instância de sistema aberto como variações em componentes, definidos com papéis, e variações em interações entre agentes, definidos como templates. Templates são utilizados para reunir o núcleo da solução e pontos de extensão. Pontos de extensão são ganchos que podem ser customizados para implementar uma instância de um framework de governança. Durante a instanciação do framework, papéis são associados a agentes e templates são refinados em especificações completas de interações. Como uma prova de conceito, neste artigo propusemos um framework para instanciar aplicações de cadeia de suprimento como sistemas abertos.

Palavras-chave: Reuso, frameworks, 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

Software permeates every aspect of our society, and it is increasingly becoming a distributed, open and ubiquitous asset. The greater the dependence of our society on open distributed applications, the greater will be the demand for new solutions that are variations of previously existing ones. One of the challenges of software development is to produce software that is designed to evolve, therefore reducing the maintenance efforts.

Nowadays, in many situations openness is a characteristic that is crucial for software. Open systems are environments where autonomous distributed components interact and may enter and leave the environment at their will [12]. Openness has led to software systems that can present an emergent or even unpredictable behavior [1]. Multi-agent auction systems are examples of such open and distributed applications [24].

Software agent technology is considered a promising approach for the development of open system applications [22]. 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 enter and participate in conversations. Agents are reused as they are required and as they conform to the specifications of the open MAS. Since open system components are often autonomous [24], sometimes they behave unpredictably and unforeseen situations arise. Taming this uncertainty is a key issue for open software development. The establishment of laws over interaction specification is a means to define what and when something can happen in an open system, representing the valid interactions in open MAS applications. When enforced, laws create a boundary of tolerated autonomous behavior and are used to foster the development of trusted systems.

Open MAS should be specified and developed to facilitate extensions. Since software systems need to be customized according to different purposes and peculiarities, the authors think that it is possible to express evolution as variations related to interactions of open systems and to components that inhabit the environment. Following this hypothesis, we propose to design open systems using extension points [8] to annotate interaction specification and using laws to customize the agents' expected behav-

ior. For component variation support, we propose to use specifications of agent roles [14][16][23]. One of the challenges of this paper is to argue that some specification elements can be reused and that some predefined "hooks" can be refined to develop a set of open MAS applications in a specific domain.

We are proposing governance frameworks based on some object-oriented framework concepts. An object-oriented framework [11] metaphor provides the necessary modeling capabilities for constructing reusable implementations of open systems. A framework is a set of abstract and concrete elements that embody a semi-complete solution. A framework instance is a set of concrete elements that specializes abstract elements to provide an executable system. The motivation for proposing the framework metaphor is to simplify the design and maintenance of a family of applications and to address the needs for highly customizable applications in an economical manner [5]. Governance frameworks may demonstrate in practice the ability to apply enforcement (or, when needed, to relax enforcement) for both complex and changing specifications. Besides customizations, the compliance of the system to the specification must continue to be analyzed by a mechanism that governs the laws of interactions in open MAS. We use the XMLaw description language [19] to map the specification of interaction rules into a governance mechanism.

A proof of concept prototype has been developed based on the specification of the Trading Agent Competition - Supply Chain Management (TAC SCM) [3][10][21]. In this example, we discuss how the changes to the laws of open MAS applications can be represented as templates that structurally "hook" the extension points into the 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 specifications, we aim to learn about how to extend the interaction specification and compliance verification in open system applications. 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 extension points.

The contributions of this paper are threefold. First, we propose to use variations and laws to specify, implement and maintain extension points in open systems. Second, we support the implementation of these variations using a law-governed mecha-

nism. Third, we specified and implemented a framework for supply chain management applications based on TAC-SCM's specifications.

The organization of this paper is as follows. Section 2 briefly describes the law-governed mechanism. In Section 3, we discuss the governance framework approach as a means to design open system for extensions. Section 4 maps the variations identified in TAC-SCM's editions into a governance framework for supply chain management. Section 5 partly describes two instances of the TAC SCM using our approach. Related work is described in Section 6. Finally, we describe our conclusions in Section 7.

2 GOVERNING INTERACTIONS IN OPEN SYSTEMS

We consider that distributed software agents are independently implemented, i.e., the development is done without a centralized control and the only restriction that we impose is that agents communicate using ACL. Furthermore, we assume that every agent developer may have an a priori access to the open system specification, including protocol descriptions and interaction laws.

Law governed architectures are designed to guarantee that the specifications will be obeyed. We developed an infrastructure that includes a modification of a basic communication infrastructure [6] that is provided to agent developers. This architecture intercepts messages and interprets the laws previously described. Whenever necessary, a software support [20] permits extending this basic infrastructure to fulfill open system requirements or interoperability concerns regarding law monitoring.

In this paper, we use the description language XMLaw [19] to represent the interaction rules of an open system specification. XMLaw (Figure 1) specifies interaction protocols using time restrictions, norms, or even time sensitive norms. The composition and interrelationship among elements is done by events. One law element can generate events to signal something to other elements. Other elements can sense events for many purposes — for instance, activating or deactivating themselves. Some enhancements on XMLaw proposed in [7][8] will be applied here, including the proposal of extension points. Those elements are represented in an XML structure like (Code 1).

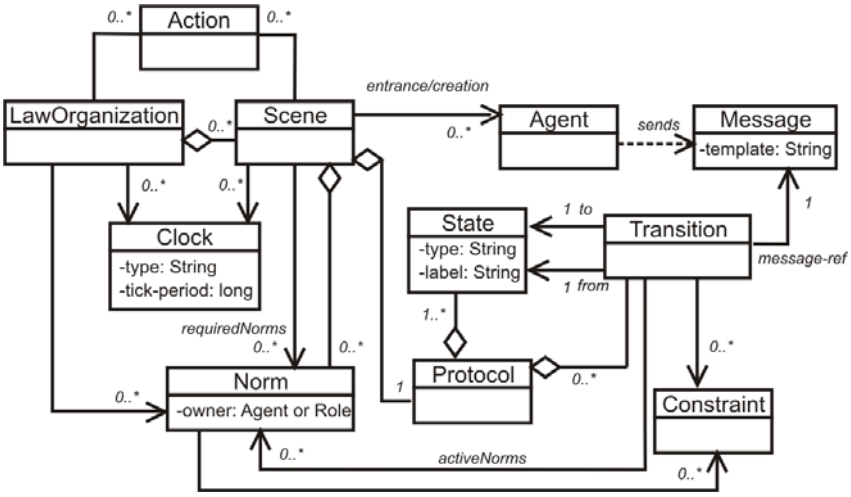

Figure 1 Conceptual Model

```
<Laws>
   <LawOrganization id="…" name="…">
      <Scene id="…" time-to-live="…">
         <Creators>…</Creators>
         <Entrance>
            <Participant role="…" limit="…"/>
         </Entrance>
         <Messages>…</Messages>
         <Protocol>
            <States> … </States>
            <Transitions>…</Transitions>
         </Protocol>
         <Norms>... </Norms>
         <Clocks>...</Clocks>
         <Actions>...</Actions>
      </Scene>
   </LawOrganization>
</Laws>
```

Code 1: XMLaw elements' structure

3 The Object-Oriented Framework Metaphor for Open Systems

An object-oriented framework is a reusable, semi-complete application that can be specialized to produce custom applications [11]. A framework is a collection of abstract entities that encapsulate common algorithms of a family of applications [12]. Abstract elements provide some "hooks" to other elements implemented or defined within the framework. A framework instance reuses the framework implementation but has only concrete elements. Some of the concrete elements are specialized from abstract ones to provide an executable system [11]. Hooks are a means of representing knowledge about the place in a framework that can be changed by application developers to produce an application from the framework [11].

Domain analysis is fundamental for reuse achievement [5]. In open software systems, the rules that enforce the relationships between agents are not always fully understood early in the framework life cycle. Still, many more rules are not applied because of the lack of systems support for changing specifications or the complexity of the specifications. Inspired by object-oriented frameworks [11], governance frameworks can deal with this complexity, reifying proven software designs and implementations in order to reduce the cost and improve the quality of software.

A governance framework is an extensible design for building open multi-agent systems. A solution for open system development is achieved by relaxing the boundary between a framework (the common part of the family of applications) and its instantiations (the application-specific part). In a governance framework, certain services and laws of the open system are abstract, because they are left unspecified or not completely specified because they would expose details that would vary among particular executable implementations.

A governance framework is flexible by design. Flexibility works in opposition to the concept of static interaction specification or enforcement and static bindings of components. Customizability ensures the framework may receive new constructs or adapt the existing ones. For this purpose, a governance framework provides "hooks" for its instances; we define abstract definitions for agents as roles and for interactions as templates. Governance support and agents' implementations that have specificities according to their applications are fully implemented later, but all common definition and implementation are present in general specifications or open system services. The realization of abstract interactions and abstract components are deferred to instantiation time and execution time, respectively.

3.1 Variations in Open Multi-Agent Systems

Software variability is the ability of a software system or artifact to be changed, customized or configured for its use in a particular context [4]. A high degree of variability allows the use of software in a broader range of contexts [4]. Besides reusability, we extend this idea, posing that the variability is also a means to specify the flexibility that a software system design has to adapt itself, preserving some previously specified characteristics.

A comparison with object-oriented frameworks can be made. The fixed part determines what definition is common to all instances and the flexible part defines different ways to configure an open MAS instance. The combination of the fixed part and the flexible part produces an instance that is equivalent to a specific definition of an application [11]. In open MAS, we have identified two categories of elements that can present variations: open system components and the interaction elements (Figure 2). Below, we detail extensions regarding both categories. Templates and roles are the definition of how extensions can be made; and they will be realized by interaction specifications and external agents, respectively.

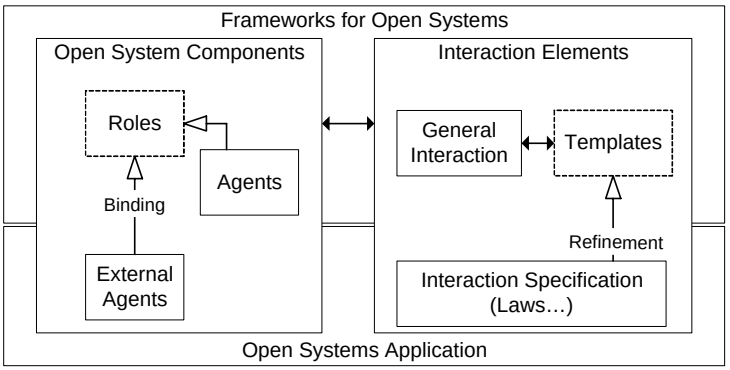

Figure 2 - Governance Framework Structure Overview

Open system components are software agents that participate in some collaboration in an open system. Agents are encapsulated entities that are rarely self-sufficient [24] and so present social skills. The semantics of an agent is largely defined by its relationship or interdependencies with other agents. Agent interdependencies can be expressed as collaborations. Collaboration is a collection of roles that encapsulates relationships across its corresponding agents. A role identifies the type of an agent and associates it with the set of characteristics that are expected from them in a collaboration. The collaboration structure defines the agent roles and their relationships. Roles can be specified as a general description for agents' responsibilities in an organization and they are bound with real software agents in open system execution.

Each component in the open MAS has a role associated with it. While playing roles, agents acquire the obligation of obeying the law that is specified for their responsibilities and it is possible to enforce the laws prescribed in the protocol. Roles specify a sort of interface that defines the responsibilities, characteristics and any other information that an agent playing this role has to fulfill. Our main purpose is not to discuss how to structure the component reuse as agent roles [14][16][23]. Rather, we intend to use an

agent role as a means to identify an extension point that describes agents' responsibilities in a collaboration and it will be realized by external software agents.

The definition of how the components interact is very important to understand the open MAS. The interaction elements comprehend the specification of dynamical concerns of an open system, including the protocol and law specification. The interaction specification is composed of interaction laws and interaction protocols. Interaction protocols define the context and the sequence of messages of a conversation between agent roles. The fixed part of interaction specifications is called general interaction. General interactions can be derived by analyzing the application domain. If any interaction element is common to all intended instances, this element is attached to the core definition of the framework.

Concerning interactions, the variability implies a more flexible protocol that specifies some alternatives and options to the execution of the open system components. Each interaction element in the open MAS is a potential extension point. The specification of interaction protocols can be made flexible enough to permit the inclusion of some norms, constraints and actions that define the desired behavior for the open MAS applications. Templates are part of the flexibility of the open MAS interactions [8]. In governance frameworks, templates are defined as "hooks" for elements of the interaction specification that will be refined during open system instantiation. Even with extension points, we still need to monitor the entire application; to gather information about its execution, and also to analyze the compliance of the system components with the previously specified desired behavior. This means that the governance mechanism must support this peculiarity.

4 Governance Framework for Open Supply Chain Management

An important characteristic of a good framework is that it provides mature runtime functionality and rules within the specific domain in which it is to be applied [11]. Hence, we based our proof of concept prototype on the specification of the Trading Agent Competition - Supply Chain Management (TAC SCM) [3][10][21]. 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 their consequences (e.g. interaction rules were defined to protect agents from malicious par-

ticipants). In our prototype, each set of rules can be used to configure a different instance of a framework for instantiating an open supply chain management system. We reuse this experience as a means to achieve the domain knowledge required for proposing a governance framework.

The TAC-SCM System [3] has been designed with a simple set of rules to capture the complexity of a dynamic supply chain. 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 agents. In the real TAC SCM architecture, there is a TAC Server that simulates the behavior of the suppliers, customers, and factories. We converted part of the simulation components present in TAC SCM to external agents or the open system's services of a prototypical version [7]. 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 evolution of TAC SCM's requirements, we can perceive evidences that interaction protocols have a core definition. We can also identify some extension points in this specification and then they can be customized to provide different instances of the supply chain. As mentioned before, extension points can specify templates that will be "hooked" into the "stable" conversation among agents. The results of this observation are presented below.

We focus on the negotiation between suppliers and assemblers to buy supplies to produce PCs. Besides these two roles, there is the bank role. Although not specified in the TAC SCM proposal, suppliers also have an account to manage their revenues and payments. There are six assembler agents that produce PCs participating in each TAC SCM instance. These participants interact with both suppliers and a bank agent. There are eight different supplier agents in each supply chain. Only one bank agent is responsible for managing payments accounts. The diagram (Figure 3) depicts the roles and their relationships. Figure 3 is based on ANote's agent class diagram [9].

We decided to organize this scenario into two scenes: one for the negotiation process between assemblers and suppliers, and the other for the payment involving the assembler and the bank agent. Code 2 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).

The Code 3 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.

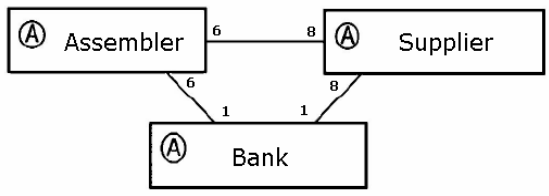

Figure 3 - Roles, relationships and cardinalities

```
<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 2: 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 3: Roles, relationships and cardinalities of payment scene

4.1 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's reply with a confirmation response, represented by the receipt message (Figure 4). The specification in XMLaw of this interaction is listed below (Code 4). Figure 4 is based on ANote's interaction diagram [9].

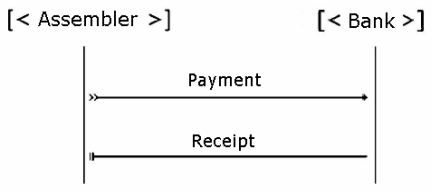

Figure 4 Payment Interaction

```
<Messages>
   <Message id="payment" template="..."/>
   <Message id="receipt" template="..."/>
</Messages>
<Protocol>
   <States>
    <State id="p1" type="initial"/>
    <State id="p2" type="execution"/>
    <State id="p3" type="success"/>
   </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 4: Payment interaction protocol description

The negotiation between assemblers and suppliers is carried out in five steps, four messages (Figure 5) and six transitions. Below (Code 5, Code 6, Code 7), we describe this scene in detail using XMLaw. Figure 5 is based on ANote's interaction diagram [9].

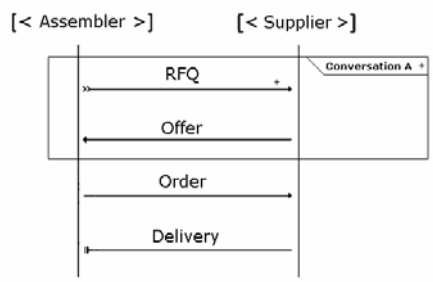

Figure 5 Negotiation interaction diagram

```
<Messages>
   <Message id="rfq" template="..."/>
   <Message id="offer" template="..."/>
   <Message id="order" template="..."/>
   <Message id="delivery" template="..."/>
</Messages>
```

Code 5: Negotiation interaction protocol description: Messages

```
<States>
    <State id="as1" type="initial"/>
    <State id="as2" type="execution"/>
    <State id="as3" type="execution"/>
    <State id="as4" type="execution">
    <State id="as5" type="success"/>
</States>
```

Code 6: Negotiation interaction protocol description: 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>
```

Code 7: Negotiation interaction protocol description

4.2 Specification Refinement

During the refinement of roles and services, it is possible to detail abstract elements or any information required by the open system specification. We need to represent the concrete relationships that are available, i.e., the relationships between domain specific or general implementations of agent roles that will be further implemented.

We specified the bank role as being realized by an agent that is common to all open system instances. The implementation of this agent is provided within the framework core implementation. The assembler and the supplier roles were left to be bound with external agents. During the execution of the open system, at most six agents will play the assembler role and eight agents will play the supplier role.

4.2.1 General Interaction Specification

To illustrate the use of general specifications, we identified the stable interactions in the last three editions of TAC SCM and we implemented it using XMLaw. This specification is reused in every instance of our governance framework. It defines 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 [3][10][21].

"On the following day 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. 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 with the quantity of items relaxed; or (ii) an earliest complete offer is generated with the 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 the Code 8 and Code 9. 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, while the other constraint verifies if a valid offer message was generated — that is, if the offer was sent one day after the RFQ. This permission is only valid if both of the constraints are true. Below, we illustrate the offerTransition (Code 8) and describe the permission RestrictOfferValues and its XMLaw specification (Code 9).

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

Code 8: General Transition Specification

XMLaw includes the context concept. Elements in the same context share the same local memory to share information, i.e., putting, getting and updating any value that is important for other law elements. Code 9 depicts one example of context usage. The keepRFQInfo Action preserves the information present in the rfq message to be later used by the checkAttributes and checkDates Contraints.

```
<Norms>
 <Permission id="RestrictOfferValues">
 <Owner>Supplier</Owner>
  <Activations>
    <Element ref="rfqTransition"
             event-type="transition_activation"/>
  </Activations>
  <Deactivations>
   <Element ref="offerTransition"
            event-type="transition_activation"/>
  </Deactivations>
  <Actions>
   <Action id="keepRFQInfo"
           class="tacscm.norm.actions.KeepRFQAction">
    <Element ref="rfqTransition"
             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>
<Norms>
```

Code 9: General Norm specification

4.2.2 The Framework's Extension Points

Code 10 is an example of a template. This permission is about the maximum number of requests for quotation that an assembler can submit to a supplier. According to TAC SCM specifications [3][10][21], each day each agent may send up to a maximum number of RFQs. But the precise number of RFQs has changed over the last editions of TAC SCM, so it is possible to defer this specification to instantiation time. We use a template for this purpose; in the template some hooks will guide the specialization of an instance of this framework.

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

Code 10: Permission and Constraint over RFQ message Templates

In this example, we opted to keep the attribute class-id of the constraint checkDueDate not specified, that is, it will be set during framework instantiation. The constraint over the acceptable due date of an RFQ (checkCounter) regulates the same interaction point, the request for quote message. The constraint checkDueDate (Code 10) is associated with the transition rfqTransition. It means that if the verification is not true the transition will not be fired.

The scenario described above is the same for a constraint over the acceptable due date of an RFQ (checkCounter), and it has changed over the last editions of TAC SCM (Code 11). The constraint checkCounter 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 is defined under the permission AssemblerPermissionRFQ and it is triggered by a clock-tick every day, zeroing the value of the counter of the number of requests issued by the assembler during this day. The other action orderID is activated by every transition transitionRFQ and is used to count the number of RFQs issued by the assembler, updating a local counter. The class that implements this action was not specified because its implementation varies according to TAC SCM editions. Finally, a clock nextDay is used to mark the day period, and this mark is used to zero the counter of RFQs by the action ZeroCounter.

```
<Norms>
   <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>
</Norms>
```

Code 11: Norm description Template

Another example of template is used to specify the relationship between orders and offers of the negotiation protocol. According to [3], 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. But when they will receive the payment is not completely specified in this law. Another template was used to map variations on TAC SCM editions. This template only specifies the structure of the ObligationToPay obligation, 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 12). The assembler can only enter into the payment scene if it has an obligation to pay for the products (Code 13). An assembler cannot enter into another negotiation if it has obligations that were not fulfilled (Code 14).

```
<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 12: Negotiation Scene and the Payment Scene

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

Code 13: Agents must have the norm to enter the payment scene

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

Code 14: Agents must not have the norm to enter 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 15: Norms of the organization

5 TAC SCM editions as Framework's Instances

In this section, we present two examples of instantiations of the framework for open supply chain management. We were inspired by the TAC SCM 2004 and 2005 editions. In this prototypical version, we considered the source of assemblers and suppliers agents as unknown. Thus, these two roles will be fulfilled during the execution of the open system. Below, we present the refinements proposed to the templates described above.

5.1 TAC SCM 2004

According to [3], on each day each agent may send up to ten RFQs to each supplier. An RFQ with DueDate beyond the end of the negotiation will not considered by the supplier. For this purpose, we implemented the constraint class ValiDate (Code 16). The constraint class CounterLimit (Code 17) checks if the local attribute for controlling the number of RFQs is below the limit of 10. The RFQCounter action increments the same attribute when receiving new messages.

According to [3], supplier will receive assembler's payment after the delivery of components and at this time the cost of the order placed before will be fully charged. We implemented the payment as an action where the system forces the agent to pay the entire debit at the end of the negotiation (Code 18).

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

Code 16: Constraint checkDueDate instance for TAC SCM 2004

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

Code 17: AssemblerPermissionRFQ instance for TAC SCM 2004

```
<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.SupplierPayment100">
   <Element ref="deliveryTransition" event-type="transition_activation"/>
  </Action>
 </Actions>
</Obligation>
```

Code 18: ObligationToPay instance for TAC SCM 2004

5.2 TAC SCM 2005

According to [10]: (i) 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 named RFQCounter2005 is provided (Code 19). It counts the number of RFQs according to the type of component. The CounterLimit2005 was also updated to consider a specific counter for each type of component that a supplier provides;

(ii) 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 two days in the future, will not be considered. It is implemented by the constraint ValiDate2005 (Code 20).

```
<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 19: AssemblerPermissionRFQ instance for TAC SCM 2005

According to [10], suppliers wishing perhaps to protect themselves from defaults will bill agents immediately for a down payment on 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%. We implemented the payment process as two actions, one for the down payment and the other for the remainder of the debit at the end of the negotiation (Code 21).

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

Code 20: Constraint checkDueDate instance for TAC SCM 2005

```
<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="supplierDownPayment" class="law.tacscm.norm.actions.SupplierPayment10">
         <Element ref="orderTransition" event-type="transition_activation"/>
     </Action>
     <Action id="supplierPayment" class="law.tacscm.norm.actions.SupplierPayment90">
         <Element ref="deliveryTransition" event-type="transition_activation"/>
     </Action>
   </Actions>
</Obligation>
```

Code 21: ObligationToPay instance for TAC SCM 2005

6 Related Work

We address the problem of constructing a family of governance mechanisms that ensure that agents will conform to a well defined customizable specification. Our main

goal was to contribute on the engineering on how we can productively use and reuse laws. Below we discuss some related work.

Ao and Minksy [2] propose an approach that enhances LGI with the concept of policy-hierarchy to support that different internal policies are formulated independently of each other, achieving a flexibility support by this means. Different from our approach, Ao and Minsky consider confidentiality as a requirement for their solution. The goal of the extensions that we have presented until now is to support open system law maintenance, rather than flexibility for the purpose of confidentiality.

COSY [13] views a protocol as an aggregation of primitive protocols. Each primitive protocol can be represented by a tree where each node corresponds to a particular situation and transitions correspond to possible messages an agent can either receive or send, i.e., the various interaction alternatives. In AgenTalk's [17], protocols inherit from one another. They are described as scripts containing the various steps of a possible sequence of interactions. Beliefs also are embedded into scripts. Koning and Huget [15] deal with the modeling of interaction protocols for multi-agent systems, outlining a component-based approach that improves flexibility, abstraction and protocol reuse. All of these approaches are useful instruments to promote reuse, they can be seen as instruments for specifying laws in governance frameworks, and their effectiveness will be evaluated in future experiments.

Singh [18] proposes a customizable governance service, based on skeletons. His approach formally introduces traditional scheduling ideas into an environment of autonomous agents without requiring unnecessary control over their actions, or detailed knowledge of their designs. Skeletons are equivalent to state based machines and we could try to reuse their formal model focusing on the implementation of a family of applications. But [18] has few implementation details and examples where allowing us to understand how his proposal was implemented.

7 Conclusions

In open multi-agent systems, in which components are autonomous and heterogeneous, trust is crucial. This paper presented an approach to ensure trust and augment reliability on customizable open systems. The approach 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.

The purpose of this paper was to derive an approach that could be useful to facilitate extensions on governance mechanisms for open systems. Interaction and roles are first order abstractions in open system specification reuse. Here, we illustrated how interaction could be easily designed for reuse. We can also conclude that while analyzing the open software system domain, it is possible to distinguish two kinds of interaction specification: fixed (stable) and flexible (extensible). The challenge to developers is to deliver a specification that identifies the aspects of the open MAS that will not change and cater the software to those areas. Stability is characterized by the interaction protocol and some general rules that are common to all open MAS instances. Extensions on interaction rules will impact the open MAS and the agents and extensions are specified.

The experiment showed that this is an interesting and promising approach; it improves the open system design by incorporating reliability aspects that can be customized according to application requirements and it improves maintainability. 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).

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

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

4. Bachmann, F and Bass, L. "Managing variability in software architectures," presented at Proceedings of the 2001 symposium on Software reusability: putting software reuse in context, Toronto, Canada.

5. Batory, D; Cardone, R. and Smaragdakis, Y. "Object-Oriented Frameworks and ProductLines", 1st Software Product-Line Conference, Denver, Colorado, August 2000.

6. Bellifemine, F; Poggi, A; Rimassa, G. (2001) Jade: a fipa2000 compliant agent development environment, in: Proceedings of the fifth international conference on Autonomous agents, ACM Press, 2001, pp. 216–217

7. Carvalho, Gustavo; Paes, Rodrigo; Lucena, Carlos. Governing the Interactions of an Agent-based Open Supply Chain Management System. MCC nº 29/05, Dpto de Informática, PUC-Rio, 27 p., 2005.

8. Carvalho, Gustavo; Paes, Rodrigo; Lucena, Carlos. Extensions on Interaction Laws in Open Multi-Agent Systems. In: First Workshop on Software Engineering for Agent-oriented Systems (SEAS 05), 19th Brazilian Symposium on Software Engineering. Uberlândia, Brasil

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

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

11. Fayad, M; Schmidt, D.C.; Johnson, R.E. Building application frameworks : object-oriented foundations of framework design. ISBN 0471248754, New York: Wiley, 1999.

12. Fredriksson M. 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), pp. 355 - 358, IEEE Press, 2003.

13. Haddadi, A. Communication and Cooperation in Agent Systems: A Pragmatic Theory, volume 1056 of Lecture Notes in Computer Science. Springer Verlag, 1996.

14. Kendall, E. "Role Modelling for Agent Systems Analysis, Design and Implementation", IEEE Concurrency, 8(2):34-41, April-June 2000.

15. Koning, J.L. and Huget, M.P.. A component-based approach for modeling interaction protocols. In H. Kangassalo and E. Kawaguchi, editors, 10th European-Japanese Conference on Information Modelling and Knowledge Bases, Frontiers in Artificial Intelligence and Applications.IOS Press, 2000

16. Kristensen, B. B., Østerbye, K. "Roles: Conceptual Abstraction Theory & Practical Language Issues", Special Issue of Theory and Practice of Object Systems on Subjectivity in Object-Oriented Systems, Vol. 2, No. 3, pp. 143-160, 1996.

17. Kuwabara, K; Ishida, T; and Osato, N. AgenTalk: Coordination protocol description for multiagent systems. In First International Conference on MultiAgent Systems (ICMAS-95), San Francisco, June 1995. AAAI Press. Poster.

18. Singh, M. P., "A Customizable Coordination Service for Autonomous Agents," Intelligent Agents IV: Agent Theories, Architectures, and Languages, Munindar P. Singh et al. ed., Springer, Berlin, 1998, pp. 93-106.

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

20. Paes, R.B; Lucena, C.J.P; Alencar, P.S.C. A Mechanism for Governing Agent Interaction in Open Multi-Agent Systems MCC nº 30/05, Depto de Informática, PUC-Rio, 31 p., 2005

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

22. Wooldridge, M; Weiss, G; Ciancarini, P. (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, 2002.

23. Yu, L; Schmid, B.F. "A conceptual framework for agent-oriented and role-based workflow modelling", the 1st International Workshop on Agent-Oriented Information Systems, Heidelberg, June 1999.

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