

Applying the Governance Framework Technique to Promote Maintainability in Open Multi-Agent Systems

Gustavo Carvalho¹, Carlos J.P. de Lucena¹, Rodrigo Paes¹,
Ricardo Choren², and Jean-Pierre Briot³

¹ PUC-Rio - Marqus de So Vicente 225,
4 Andar RDC - Gvea RJ, Brazil
{guga,lucena,rbp}@inf.puc-rio.br

² SE/8 - IME
Pa Gen Tibrcio 80 - Praia Vermelha,
RJ, Brazil
choren@de9.ime.eb.br

³ LIP6, Université Pierre et Marie Curie (Paris 6)
8 rue du Capitaine Scott, 75015 Paris, France
Jean-Pierre.Briot@lip6.fr

Abstract. Governance means that specifications are enforced dynamically at application runtime. Governance framework is a technique to design and implement an extensible interaction specification for a family of open systems. This specification can be refined for particular applications. We based this proposal on object-oriented framework concepts and adapted them for distributed agents and interactions. A governance framework structures the extensions of open system instances as variations in interactions among agents, defined as templates. Templates are used to gather core implementation and extension points. Extension points are "hooks" that will be customized to implement an instance of the governance framework. During framework instantiation, 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: Interaction protocol, Reuse, Law-enforcement.

1 Introduction

Nowadays, software permeates every aspect of our society, and it is increasingly becoming a distributed and open asset. Distribution means that it is possible to integrate different software solutions from different sources or machines and they work cooperatively to achieve system requirements. Openness is crucial for software. Open systems are software systems in which autonomous distributed components interact and may enter and leave the environment at their will [10]. Auction systems and virtual enterprises are examples of such open and

distributed applications [19]. Software agent technology is considered a promising approach for the development of open system applications [19].

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 will only be permitted to interact if they conform to the specification of the open MAS. Since open system components are often autonomous, 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 and their enforcement over software agents create a boundary of tolerated autonomous behavior and can be used to foster the development of reliable systems.

In open software systems, rules that enforce the relationships between agents are not always fully understood early in the development life cycle. Still, many more rules are not applied because of the lack of system support for changing specifications or the complexity of the specifications. Inspired by object-oriented frameworks [9], governance frameworks are proposed to deal with this complexity, reifying proven software designs and implementations in order to reduce the cost and improve the quality of software.

Since software systems need to be customized according to different purposes and peculiarities, it should be 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 [5] to annotate interaction specification and using laws to customize the agents' expected behavior. We 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 [9]. This approach provides the necessary modelling capabilities for constructing reusable implementations of open systems. 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 XMLLaw description language [16] to map the specification of interaction rules into a governance mechanism. The purpose of governance frameworks is to provide an approach to support the development of governance mechanisms.

A proof of concept prototype has been developed based on the specification of the Trading Agent Competition - Supply Chain Management (TAC SCM) [2, 8, 18]. 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 specify and develop open software systems using extension points.

The contributions of this paper are threefold. First, we jointly apply 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 mechanism. Third, we specified and implemented a governance 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

Software agents in open MAS are heterogeneous, i.e., the development is done without a centralized control, possibly by different parties, with different purposes and preferences. The only restriction an open MAS imposes is that the agents communicate through a common language. In this work we assume that every agent developer may have an a priori access to the open system specification, including protocol descriptions and interaction laws and that agents communicate using ACL.

Law-governed architectures are designed to guarantee that the specifications will be obeyed. We developed an infrastructure that includes a modification

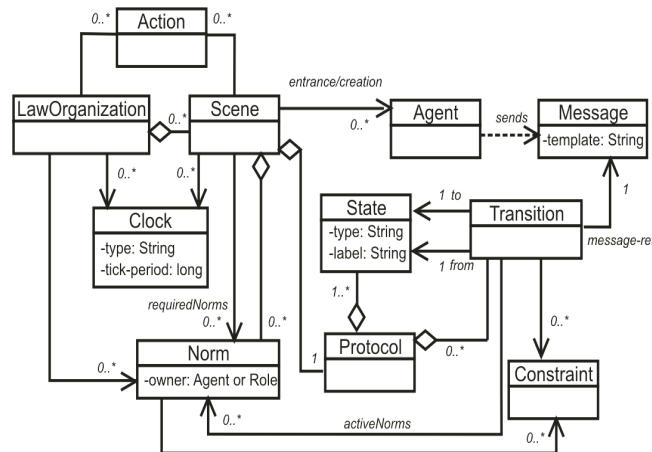


Fig. 1. Conceptual Model of XMLaw

of a basic communication infrastructure [4] that is provided to agent developers. This architecture intercepts messages and interprets the laws previously described. Whenever necessary, a software support [17] 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 [16] to represent the interaction rules of an open system specification. XMLaw (Fig.1) specifies interaction protocols using time restrictions, norms, or even time sensitive norms.

Those elements are represented in an XML structure like (Listing 1.1). 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.

```

<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>

```

Listing 1.1. XMLaw structure

2.1 Refinement Operators to Specify Laws in Open Multi-Agent Systems

To design the interaction laws of open MAS to facilitate extensions to deal with changing requirements, it is necessary to have an instrument to specify which law elements can be customized and so described as extension points. The extension points are a means of representing knowledge about the place where modifications and enhancements in laws can be made. In our context, it is useful to permit the inclusion of norms, constraints and actions into a pre-defined law specification.

XMLaw [16] has two elements that can be easily plugged into the specification of interaction laws: actions and constraints. Actions are used to plug services into open systems. Services are domain specific functionalities that can be triggered while the mediator monitors agents' interactions. The first attempt to define extension points was deferring the definition of the class implementation [5]. We enhanced this notion with the proposal of refinements operators in XMLaw [6]. Below, we explain how the interaction specification with extension points can be

prepared to further refinements. The examples below detail laws applied on sales that are customized according to the period of the year (e.g. discounts are given in summer and winter). The action `giveDiscount` (Listing 1.2) will calculate and apply the discount and the constraint `badClient` (Listing 1.3) restricts discounts to this kind of client.

```
<Actions>
  <Action id="giveDiscount">
    <Element ref="payment"
      event-type="transition_activation"/>
  </Action>
</Actions>
```

Listing 1.2. Action hook

```
<Constraints>
  <Constraint id="badClient"/>
</Constraints>
```

Listing 1.3. Constraint hook

The `abstract` attribute defines when a law element is not completely implemented. If no value for the `abstract` attribute is determined, the element is a concrete one (default `abstract = "false"`). If the law designer wants to specify that a law element needs some refinements to be used he has to explicitly specify the attribute `abstract` with the value `true` (`abstract = "true"`). If a law is defined as concrete, it cannot leave any element to be further refined; all elements must be fully implemented, otherwise, the interpreter will indicate an error. An abstract operator can define law elements with some gaps to be filled further (Listing 1.4). This is the extension point idea, defining clearly the context where the extensions are expected. Until now, we can defer the definition of the implementation of actions and constraints classes or the inclusion of any law element. Below, we structure the usage of the `badClient` constraint and `giveDiscount` action inside the permission `Sale`. If this permission active during the enforcement process, this discount action and verification constraint can be triggered.

```
<Permission id="Sale" abstract="true">
  <Owner>...</Owner>
  <Activations> ... </Activations>
  <Deactivations> ... </Deactivations>
  <Constraints>
    <Constraint id="badClient"/>
  </Constraints>
  <Actions>
    <Action id="ad" class="...">...</Action>
    <Action id="giveDiscount">...</Action>
  </Actions>
</Permission>
```

Listing 1.4. Abstract operator

The completes attribute is an operator that is useful to fill the elements that were left unspecified when a law element was defined as abstract (Listing 1.5). It is a simple operator to realize extensions as it can just be used to define action and constraints class implementations. The completes operator turns an abstract element into a complete one and cannot leave any element unspecified unless it also redefines this element as an abstract one. The completes operator is limited to the definition of class implementations.

```
<Permission id="SummerSale" completes="Sale">
  <Constraint id="badClient" class="BadCustomers"/>
  <Action id="giveDiscount" class="Percentage10"/>
</Permission>
```

Listing 1.5. Completes operator

The extends attribute is a more powerful operator and it is similar to the specialization operation in object-oriented languages. Basically, an extends operator reuses the description of law elements and includes any modifications that are necessary to customize the law element to users' needs, including the redefinition of law elements. For example, this operator can include new activation references, new action elements, and new norm elements and can also superpose any element that was previously specified. The extends operator also turns an abstract element into a complete one and cannot leave any element unspecified unless it redefines this element as an abstract one (Listing 1.6).

```
<Permission id="WinterSale" extends="Sale">
  <Constraints>
    <Constraint id="badClient" class="BadPayers"/>
  </Constraints>
  <Actions>
    <Action id="giveDiscount" class="Percentage15"/>
    <Action id="giveSuperDiscount"
      class="ChristmasDiscount">
      <Element ref="christimas"
        event-type="clock_activation"/>
    </Action>
  </Actions>
</Permission>
```

Listing 1.6. Extends operator

3 Improving Governance Mechanism Maintainability

Suppose that it is possible to specify and implement the kernel of a general solution and this kernel can be customized to different purposes. In this kernel, you have the exact points that can be modified and enhanced. In our context, this approach can be used to derive a family of governance mechanisms that share a core

specification and implementation. The customization is used for two purposes. First, different governance mechanisms can be derived for different purposes and application scenarios. Second, different versions of the same governance mechanism can be instantiated during its lifecycle. To realize this scenario, we propose to use some object-oriented framework concepts. An object-oriented framework is a reusable; semi-complete application that can be specialized to produce custom applications [9], i.e., it is a collection of abstract entities that encapsulate common algorithms of a family of applications [3].

In this work, we focus on understanding how interaction specification enhanced by laws can be designed to support extensions. A governance framework is an extensible design for building governance mechanisms for open MAS. 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 laws of the open system are abstract, because they are left either unspecified or incompletely 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. In design time, customizability ensures the framework may receive new law elements or adapt the existing ones. For this purpose, a governance framework provides "hooks" for its instances; we define abstract definitions for interactions as templates. Governance functionalities that have specificities according to their applications are fully implemented later, but all common definitions and implementations are present in general specifications. The realization of abstract interactions is deferred to instantiation time.

In MAS, the collaboration structure defines the agent roles and their relationships. Roles are useful to specify general descriptions for agents' responsibilities in an organization [20] and they are bound to real software agents in open system execution. While playing roles, agents acquire the obligation to obey the law that is specified for their responsibilities and it is possible to enforce the laws prescribed in the protocol. Our main purpose is not to discuss how to structure the component reuse as agent roles [12, 20], which will be realized by external software agents. We intend to use an agent role at the design level as a means to describe agents' responsibilities.

Our main concern is over how agents interact in the open MAS. The interaction elements comprehend the specification of dynamical concerns of an open system. 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 (Fig. 2) 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 specification to include some alternatives and options to the design of a family of similar open MAS. Each interaction element in the open MAS is a potential extension point. The specification of interaction protocols can be made flexible enough to include new elements like norms, constraints and actions that define the desired behavior for the open MAS applications. Templates (Fig. 2) are part of the flexibility of the open MAS interactions [5]. In governance frameworks, templates are defined as "hooks" for elements of the interaction specification that will be refined during the governance mechanism instantiation.

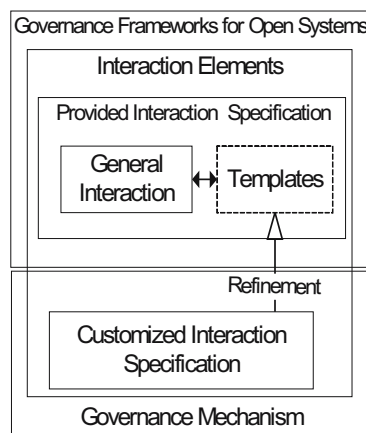


Fig. 2. Governance Framework Overview

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 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 [9]. Hence, we based our proof of concept prototype on the specification of the TAC SCM [2, 8, 18]. The game rules have been updated over the last three years. This evolution was achieved by observing the behavior of different agents during the last editions and their consequences (e.g. interaction rules were defined to protect agents from malicious participants). In our prototype,

each set of rules can be used to configure a different instance of a framework for instantiating governance mechanisms in open supply chain management domain.

In TAC SCM, 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. 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 variability of the negotiation between suppliers and assemblers over TAC SCM editions we depict the architecture below. The kernel of this framework is composed of a scene for negotiation, a scene for payment, the definition of interaction steps (transitions), states and messages, and a permission to restrict the offer values of an RFQ. The extension points that will be described here include the permission granted to assemblers to issue requests during one day (number of permitted requests and how to count them), the constraint to verify the date in which a request is valid, and the payment method implemented by actions inside the obligation.

4.1 Kernel Description

In the negotiation, assemblers buy supplies from suppliers to produce PCs. Besides these two roles, there is the bank role. 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 agent class diagram [7] (Fig. 3) depicts the roles, their relationships and their cardinalities.

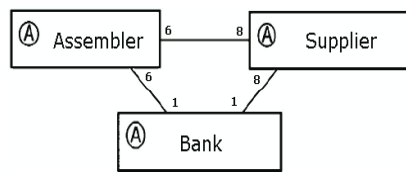


Fig. 3. Roles, relationships and cardinalities

We decided to organize this scenario in two scenes: one for the negotiation process between assemblers and suppliers, and the other for the payment involving the assembler and the bank agent. Listing 1.7 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). Listing 1.8 describes the payment process. We decided not to specify any time out to the payment scene and this is represented by the "infinity" value of the attribute time-to-live.

```
<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>
```

Listing 1.7. Negotiation scene structure

```
<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>
```

Listing 1.8. Payment scene structure

Analyzing the evolution of TAC SCM's requirements, we can observe evidences that interaction protocols have a core definition. In this specification we can also identify some extension points, which can be customized to provide different instances of the supply chain. 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 ((Listing 1.9 and 1.10, Fig. 4). Fig. 4 is based on the interaction diagram [7].

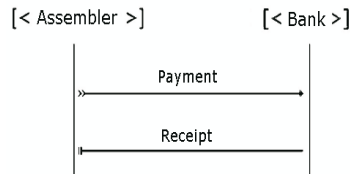


Fig. 4. Payment interaction

```

<Messages>
  <Message id="payment" template="..." />
  <Message id="receipt" template="..." />
</Messages>

```

Listing 1.9. Payment messages description

```

<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="paymentConcludedTrans"
      from="p2" to="p3" message-ref="receipt" />
  </Transitions>
</Protocol>

```

Listing 1.10. Payment interaction protocol description

The negotiation between assemblers and suppliers is carried out in five steps, four messages (Fig. 5, (Listing 1.11) and six transitions. Below ((Listing 1.11 and 1.12), this scene is described in detail using XMLaw. Fig. 5 is based on the interaction diagram [7].

```

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

```

Listing 1.11. Negotiation interaction protocol: messages

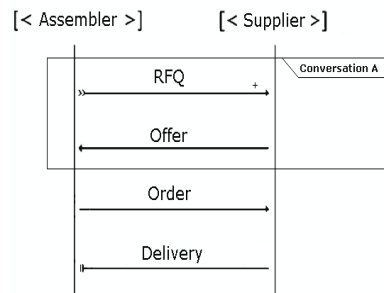


Fig. 5. Negotiation protocol diagram

```

<Protocol>
  <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>
  <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>

```

Listing 1.12. Negotiation interaction protocol description

To further illustrate the use of general specifications, we identified the stable interaction laws in the last three editions of TAC SCM and we implemented it using XMLLaw. This specification is reused in every instance of our governance framework. This law 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 [2,8, 18].

"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. 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 XMLLaw is illustrated in Listings 1.13 and 1.14. 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 (Listing 1.13) and describe the permission RestrictOfferValues and its XMLaw specification (Listing 1.14).

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

Listing 1.13. General Transition Specification

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

Listing 1.14. General Norm specification

XMLaw includes the notion of context. 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. Listing 1.14 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 constraints.

4.2 Extension Point Descriptions

The constraint checkDueDate verifies if the date attribute is according to the restrictions imposed by the edition of the environment. It means that if the verification is not true the transition will not be fired. This constraint is associated with the transition rfqTransition and this transition is specified as abstract to clearly document the extension point. Listing 1.15 is an example of a template. In this example, we opted to keep the attribute class of the constraint checkDueDate not specified, that is, it will be set during framework instantiation.

```

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

```

Listing 1.15. Permission and Constraint over RFQ message Templates

According to TAC SCM specifications [2, 8, 18], every 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 and we have created a permission to encapsulate this requirement (Listing 1.16); in the template some hooks will guide the specialization of an instance of this framework.

This permission is about the maximum number of requests for quotation that an assembler can submit to a supplier. To implement this sort of verification, 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, turning to zero 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. 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`. In this paper, we do not describe the clock `nextDay` specification.

```

<Permission id="AssemblerPermissionRFQ" abstract="true">
  <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>

```

Listing 1.16. Norm description Template

Another extension point is used to specify the relationship between orders and offers of the negotiation protocol. According to [2], 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 specifies the structure of the `ObligationToPay` obligation (Listing 1.17), defining that it will be activated by an order message and 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 (Listing 1.18). The assembler can only enter into the payment scene if it has an obligation to pay for the products (Listing 1.19). An assembler cannot enter into another negotiation if it has obligations that were not fulfilled (Listing 1.20).

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

Listing 1.17. Obligation to pay

```
<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>
```

Listing 1.18. Negotiation and Payment Scene

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

Listing 1.19. Payment scene and `ObligationToPay` norm

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

Listing 1.20. Negotiation scene and `ObligationToPay` norm

5 TAC SCM Editions as Framework's Instances

In this section, we present two examples of instantiations of the framework for open SCM, explaining the refinements proposed to the templates described above.

In TAC SCM 2004 and according to [2], a supplier will receive an 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 (Listing 1.23). According to [2], 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 be considered by the supplier. For this purpose, we implemented the constraint class ValidDate (Listing 1.21). The constraint class CounterLimit (Listing 1.22) 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 of RFQ.

```
<Transition id="rfq2004" completes="rfqTransition">
  <Constraint id="checkDueDate"
    class="constraints.ValidDate2004"/>
</Transition>
```

Listing 1.21. checkDueDate instance for TAC SCM 2004

```
<Permission id="AssemblerPermissionRFQ2004"
  completes="AssemblerPermissionRFQ">
  <Constraint id="checkCounter"
    class="constraint.CounterLimit"/>
  <Action id="orderID" class="norm.actions.RFQCounter">
</Permission>
```

Listing 1.22. Permission instance for TAC SCM 2004

```
<Obligation id="ObligationToPay2004" extends="ObligationToPay">
  <Actions>
    <Action id="supplierPayment"
      class="actions.SupplierPayment100">
      <Element ref="deliveryTransition"
        event-type="transition_activation"/>
    </Action>
  </Actions>
</Obligation>
```

Listing 1.23. Obligation instance for TAC SCM 2004

In TAC SCM 2005, suppliers wishing perhaps to protect themselves from defaults will bill agents immediately for a down payment on the cost of each order placed [8]. The remainder of the value of the order will be billed when the order is shipped. In 2005, the down payment ratio is 10.


```

<Transition id="rfq2005" completes="rfqTransition">
  <Constraint id="checkDueDate"
    class="constraints.ValidDate2005"/>
</Transition>

```

Listing 1.24. checkDueDate instance for TAC SCM 2005

```

<Permission id="AssemblerPermissionRFQ2005"
  completes="AssemblerPermissionRFQ">
  <Constraint id="checkCounter"
    class="constraint.CounterLimit2005"/>
  <Action id="orderID"
    class="norm.actions.RFQCounter2005"> ... </Action>
</Permission>

```

Listing 1.25. Permission instance for TAC SCM 2005

```

<Obligation id="ObligationToPay2005" extends="ObligationToPay">
  <Actions>
    <Action id="supplierDownPayment"
      class="actions.SupplierPayment10">
      <Element ref="orderTransition"
        event-type="transition_activation"/>
    </Action>
    <Action id="supplierPayment"
      class="actions.SupplierPayment90">
      <Element ref="deliveryTransition"
        event-type="transition_activation"/>
    </Action>
  </Actions>
</Obligation>

```

Listing 1.26. Obligation instance for TAC SCM 2005

6 Related Work

Ao and Minsky [1] propose an approach that enhances their Law Governed Interaction (LGI) with the concept of policy-hierarchy to support that different internal policies or laws are formulated independently of each other, achieving a flexibility support by this means. [1] consider confidentiality as a requirement for their solution. However, the extensions presented here intend to support the maintenance of governance mechanisms, rather than flexibility for the purpose of confidentiality.

Singh [15] 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 adapt and reuse their formal model focusing on the implementation of a family of applications. But [15] has its focus on building multi-agent systems instead of providing support for monitoring and enforcement purpose.

Below we describe some useful instruments to promote reuse; they can be seen as instruments for specifying extendable laws in governance frameworks. COSY

[11] 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 [14], protocols inherit from one another. They are described as scripts containing the various steps of a possible sequence of interactions. Koning and Huget [13] deal with the modeling of interaction protocols for multi-agent systems, outlining a component-based approach that improves flexibility, abstraction and protocol reuse.

7 Conclusions

In open MAS, in which components are autonomous and heterogeneous, governance is crucial. This paper presented an approach to augment reliability on customizable open systems. The approach is based on governing the interactions in the system. This is a non-intrusive method, allowing 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 governance frameworks is to facilitate extensions on governance mechanisms for open systems. Interaction and roles are first order abstractions in open system specification reuse. Besides, 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 main contribution of this work is to provide a technique to design software that evolves, therefore reducing maintenance efforts.

With this proposal we aim to improve the engineering of distributed systems, providing a customizable conformance verification mechanism. We are also targeting improvement in the quality of governance mechanisms of open systems; this will be achieved by facilitating the extension of governance mechanisms. We propose to use variations and laws to specify, implement and maintain extension points. We also support the implementation of these variations using a law-governed mechanism. 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) and by CAPES as part of the EMACA Project (CAPES/COFECUB 482/05 PP 016/04).

References

1. 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.
2. 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
3. Batory, D; Cardone, R. and Smaragdakis, Y. "Object-Oriented Frameworks and ProductLines", 1st Software Product-Line Conference, Denver, Colorado, August 2000.
4. Bellifemine, F; Poggi, A; Rimassa, G. Jade: a fipa2000 compliant agent development environment, in: Proceedings 5th international conference on Autonomous agents, ACM Press, 2001, pp. 216-217
5. 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
6. Carvalho, G.; Lucena, C.; Paes, R.; Briot, J.P.; Refinement Operators to Facilitate the Reuse of Interaction Laws in Open Multi-Agent Systems, International Workshop on Software Engineering for Large-scale Multi-Agent Systems (SELMAS'06), 5th, at ICSE 2006, Shanghai, China. In: Proceedings of the Fifth International Workshop on Software Engineering for Large-scale Multi-agent Systems, p. 75-82, May 21-22, 2006.
7. Choren, R. and Lucena, C.J.P. Modeling Multi-agent systems with ANote. Software and Systems Modeling 4(2), 2005, p. 199 - 208.
8. 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.
9. Fayad, M; Schmidt, D.C.; Johnson, R.E. Building application frameworks : object-oriented foundations of framework design. ISBN 0471248754, New York: Wiley, 1999.
10. 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.
11. Haddadi, A. Communication and Cooperation in Agent Systems: A Pragmatic Theory, volume 1056 of Lecture Notes in Computer Science. Springer Verlag, 1996.
12. Kendall, E. "Role Modeling for Agent Systems Analysis, Design and Implementation", IEEE Concurrency, 8(2):34-41, April-June 2000.
13. Koning, J.L. and Huget, M.P.. A component-based approach for modeling interaction protocols. In H. Kangassalo and E. Kawaguchi (eds) 10th European-Japanese Conference on Information Modeling and Knowledge Bases, Frontiers in Artificial Intelligence and Applications. IOS Press, 2000
14. 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.
15. Singh, M. P., "A Customizable Coordination Service for Autonomous Agents," Intelligent Agents IV: Agent Theories, Architectures, and Languages, Springer, Berlin, 1998, pp. 93-106.

16. 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.
17. Paes, R.B; Lucena, C.J.P; Alencar, P.S.C. A Mechanism for Governing Agent Interaction in Open Multi-Agent Systems MCC n 30/05, Depto de Informtica, PUC-Rio, 31 p., 2005
18. Sadeh, N; Arunachalam, R; Eriksson, J; Finne, N; Janson, S. TAC-03: a supply-chain trading competition, AI Mag. 24 (1) 92-94, 2003.
19. 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.
20. 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, 1999.