A Governance Framework Implementation for Supply Chain Management Applications as Open Multi-Agent Systems

Gustavo Carvalho[1], Carlos Lucena[1], Rodrigo Paes[1], Jean-Pierre Briot[2], Ricardo Choren[3]

[1] PUC-Rio – Marquês de São Vicente 225
4º Andar RDC – Gávea RJ, Brazil
`{guga,lucena}@inf.puc-rio.br`
[2] LIP6 – Univ. Paris 6 - CNRS
8 rue du Capitaine Scott - 75015 PARIS , France
`Jean-Pierre.Briot@lip6.fr`
[3] SE/8 – IME – Pça Gen Tibúrcio 80 - Urca RJ, Brazil
`choren@de9.ime.eb.br`

Abstract. Governance means that specifications are enforced dynamically at application run-time. 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 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, 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.

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 where autonomous distributed components interact and may enter and leave the environment at their will [11]. Auction systems and virtual enterprises are examples of such open and distributed applications [21].

The greater the dependence of our society on open distributed systems, the greater will be the demand for dependable applications and also for new solutions that are variations of previously existing ones. Firstly, we should guarantee that errors in specific components will not be propagated to others. An error occurs when any component is not conforming to an existent specification. Secondly, one of the challenges of software development is to produce software that is designed to evolve, and so be extended, therefore reducing the maintenance efforts.

Software agent technology is considered a promising approach for the development of open system applications [20]. We believe that 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 specifications of the open MAS. Since open system components are often autonomous [22], 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 might create a boundary of tolerated autonomous behavior and can be used to foster the development of trusted systems.

In open software systems, the 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 systems support for changing specifications or the complexity of the specifications. Inspired by object-oriented frameworks [10], 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.

We believe that 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 [7] to annotate interaction specification and using laws to customize the agents' expected behavior. 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 [10] metaphor provides the necessary modeling 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 XMLaw description language [17] 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][9][19]. 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, analyze 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 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 the 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 this work imposes is that the agents communicate using ACL. We also 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 [5] that is provided to agent developers. This architecture intercepts messages and interprets the laws previously described. Whenever necessary, a software support [18] 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 [17] 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 [6][7] will be applied here, including the proposal of extension points. Those elements are represented in an XML structure like (Code 1).

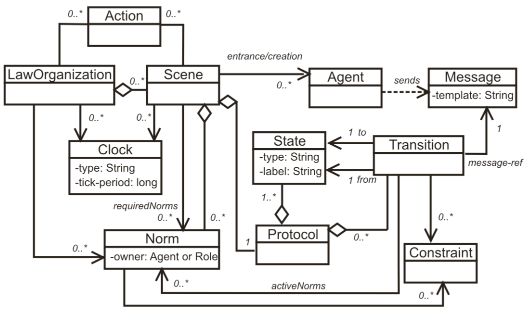

Figure 1 Conceptual Model

```
<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

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

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

XMLaw has two elements that can be easily plugged into the specification of interaction laws: actions and constraints. Actions are used to plug services in open systems. Services are domain specific functionalities. The first attempt to define extension points was deferring the definition of the class implementation [7]. We enhanced this notion with the proposal of refinements operators in XMLaw (ref). Below, we explain how the interaction specification with extension points can be prepared to further refinements. The examples below details laws applied on sales that are customized according to the period of the year (e.g. summer and winter). Discounts are given according to the period of the year. The actions will calculate and apply the discount and the constraint badClient identifies this kind of clients.

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

Listing 1: Action hook

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

Listing 2: Constraint hook

It is useful to indicate in XMLaw code when we have "hooks" or even when the existing laws must be better defined to be used. 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 can not 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. It is a means to achieve 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.

As laws can be defined as abstract, with some elements to be further detailed, we still need instruments to describe at implementation time the modifications to turn laws concrete. 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. 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 can not leave any element unspecified unless it also redefines this element as an abstract one. The completes operator can not include any new element to the abstract law, it is limited to the definition of class implementations.

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 can not leave any element unspecified unless it redefines this element as an abstract one.

```
<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 3: abstract operator

```
<Permission id="SummerSale"
            completes="Sale">

   <Constraint id="badClient"
               class="BadCustomers"/>
   <Action id="giveDiscount"
           class="Percentage10"/>

</Permission>
```

Listing 4: completes operator

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

Listing 5: extends operator

3 The Object-Oriented Framework Metaphor for Open Systems

Suppose that it is possible to specify and implement the kernel of a general solution and this kernel can be customized to different purposes (**Figure 2**). 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. Firstly, different governance mechanisms can be derived for different purposes and application scenarios. Secondly, different versions of the same governance mechanism can be instantiated during its lifecycle. For this reason, we reuse some object-oriented framework concepts. An object-oriented framework is a reusable, semi-complete application that can be specialized to produce custom applications [10], i.e., a framework is a collection of abstract entities that encapsulate common algorithms of a family of applications [11].

In this paper, 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 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 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. 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. The realization of abstract interactions is deferred to instantiation time. Governance functionalities that have specificities according to their applications are fully implemented later, but all common definition and implementation are present in general specifications.

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 [22] 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 [13] [21], which will be realized by external software agents. Rather, we intend to use an agent role at the design level as a means to describe agents' responsibilities in a collaboration. The definition of how roles 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 (**Figure 3**) 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.

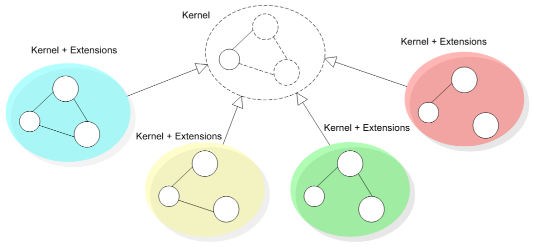

Figure 2 Hypothesis Scenario Illustration

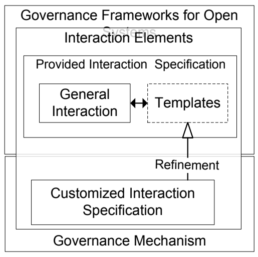
Governance Frameworks for Open
Interaction Elements
Provided Interaction Specification
General Interaction
Templates
Refinement
Customized Interaction Specification
Governance Mechanism

Figure 3 Governance Framework Structure Overview

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 (**Figure 3**) are part of the flexibility of the open MAS interactions [7]. 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 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 [10]. Hence, we based our proof of concept prototype on the specification of the Trading Agent Competition - Supply Chain Management (TAC SCM) [3][9][19]. The TAC SCM System [3] has been designed with a simple set of rules to capture the complexity of a dynamic supply chain. 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 participants). 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.

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 [6]. 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 a RFQ. The extension points that will be described here includes 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 ANote's agent class diagram [8] (**Figure 4**) depicts the roles and their relationships.

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 of the attribute time-to-live.

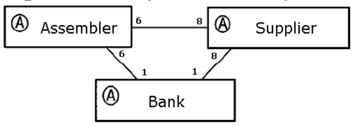

Figure 4 - Roles, relationships and cardinalities

```
<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: 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>
```

Code 3: Payment scene structure

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.

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 (Code 4, Code 5, **Figure 5**). **Figure 5** is based on ANote's interaction diagram [8].

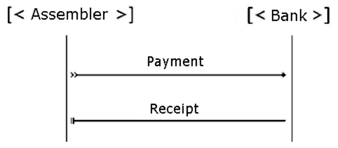

Figure 5 Payment Interaction

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

Code 4: 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="paymentConcludedTransition" from="p2" to="p3" message-ref="receipt"/>
   </Transitions>
```

```
</Protocol>
```

Code 5: Payment interaction protocol description

The negotiation between assemblers and suppliers is carried out in five steps, four messages (**Figure 6**, Code 6) and six transitions. Below (Code 6, Code 7), we describe this scene in detail using XMLaw. **Figure 6** is based on ANote's interaction diagram [8].

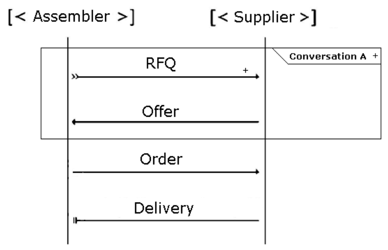

Figure 6 Negotiation diagram

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

Code 6: Negotiation interaction protocol: Messages

```
<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>
```

Code 7: Negotiation interaction protocol description

4.1.1 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. 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 [3][9][19].

"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 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

```
<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>
```

Code 9: 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. 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.

4.2 Extension Point Descriptions

Code 10 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. 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.

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

Code 10: Permission and Constraint over RFQ message Templates

According to TAC SCM specifications [3][9][19], 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 and we have created a permission to encapsulate this requirement (Code 11); 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. 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. In this paper, we do not describe the clock nextDay specification.

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

Code 11: Norm description Template

Another extension point 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 13). The assembler can only enter into the payment scene if it has an obligation to pay for the products (Code 14). An assembler cannot enter into another negotiation if it has obligations that were not fulfilled (Code 15).

```
<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>
```

Code 12: 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>
```

Code 13: Negotiation Scene and the Payment Scene

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

Code 14: Payment scene and ObligationToPay norm

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

Code 15: 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 supply chain management. In this section, we instantiated the TAC SCM 2004 and 2005 editions. Below, we explain the refinements proposed to the templates described above.

5.1 TAC SCM 2004

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 20). 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 ValidDate (Code 16). The constraint class CounterLimit (Code 18) 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.

5.2 TAC SCM 2005

According to [9], 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). On 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 [9]. 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 ValidDate2005 (Code 17).

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

Code 16: checkDueDate instance for TAC SCM 2004

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

Code 17: checkDueDate instance for TAC SCM 2005

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

Code 18: AssemblerPermissionRFQ instance for TAC SCM 2004

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

Code 19: AssemblerPermissionRFQ instance for TAC SCM 2005

```
<Obligation id="ObligationToPay2004"
            extends="ObligationToPay">

 <Actions>
  <Action id="supplierPayment"
  class="tacscm.actions.SupplierPayment100">
   <Element ref="deliveryTransition"
      event-type="transition_activation"/>
  </Action>
 </Actions>

</Obligation>
```

Code 20: ObligationToPay instance for TAC SCM 2004

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

Code 21: ObligationToPay instance for TAC SCM 2005

6 Related Work

Ao and Minsky [2] 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. 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 the maintenance of governance mechanisms, rather than flexibility for the purpose of confidentiality.

Singh [16] 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 [16] has its focus on building multi-agent systems instead of monitoring and enforcement purpose.

All approaches presented below are useful instruments to promote reuse; they can be seen as instruments for specifying extendable laws in governance frameworks. COSY [12] 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 [15], protocols inherit from one another. They are described as scripts containing the various steps of a possible sequence of interactions. Koning and Huget [14] 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 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 governance frameworks is to facilitate extensions on governance mechanisms for open systems. Interaction and roles are first order abstractions in open system specification 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 main contribution of this work is to provide a technique to design software to evolve, therefore reducing the maintenance efforts.

With this proposal we aim to improve the engineering of distributed systems, providing a customizable conformance verification mechanism. We are also targeting the improvement on the quality in 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 purpose of this paper was to derive an approach that could be useful to facilitate extensions on governance mechanisms for open systems. Interaction is the first order abstraction 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. 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.

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

[5] 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

[6] 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, 2005.

[7] 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

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

[9] 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.

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

[11] 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.

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

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

[14] 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 Modelling and Knowledge Bases, Frontiers in Artificial Intelligence and Applications.IOS Press, 2000

[15] 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.

[16] 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.

[17] 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 Multiagent Systems (ANIREM), AAMAS2005, 2005.

[18] 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

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

[20] 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.

[21] 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.

[22] Zambonelli, F, Jennings, N; Wooldridge, M. Developing multiagent systems: The gaia methodology, ACM Trans. Softw. Eng. Methodol. 12 (3) 317–370, 2003.1.1 LNCS Online