Refinement Operators to Facilitate the Reuse of Interaction Laws in Open Multi-Agent Systems

Gustavo Carvalho, Carlos Lucena, Rodrigo Paes
PUC-Rio
Marquês de São Vicente 225
4º Andar RDC – Rio de Janeiro Brasil
+55-21-25406915 – 137
{guga,lucena,rbp}@inf.puc-rio.br

Jean-Pierre Briot
LIP6
Univ. Paris 6 - CNRS
8 rue du Capitaine Scott
75015 PARIS , France
Jean-Pierre.Briot@lip6.fr

ABSTRACT
As new software demands and requirements appear, the system and its interaction laws must evolve to support those changes. Languages and models should provide the tools for dealing with this evolution. Poor support on evolution has a negative impact on system maintainability. In this paper, we propose some refinement operators to extend the interaction laws in open multi-agent systems. As an example of this idea, we implemented a customizable application in the supply chain management domain as an open system environment

Categories and Subject Descriptors
F.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems—*Multiagent Organizations*; F.2.11 [**Distributed Artificial Intelligence**]: Coherence and Coordination

Keywords
Reuse, law-enforcement, software agents, interaction protocol.

1. INTRODUCTION
Nowadays, one characteristic that is crucial for software in many situations is openness. Open systems are composed of autonomous distributed components that may enter and leave the environment at their will, and they may even have conflicting interests [8] because open software systems have no centralized control over the development of its parts [1]. Multi-agent auction systems and virtual enterprises are examples of such open and distributed applications [21].

Since open system components are often autonomous, sometimes they behave unpredictably and unforeseen situations arise. We believe that the specification of open multi-agent systems (open MAS) should include laws that define what and when something can happen in an open system. Laws are restrictions imposed by the environment to tame uncertainty and to promote open system dependability [14][15]. A governance mechanism is the mediator that enforces the law specification. Examples of governance mechanisms are LGI[14], Islander[7] and XMLaw[16]. In this paper, we will use the XMLaw description language [15] to map customizable specification of interaction rules into a governance mechanism.

The greater the dependence of our society on open distributed applications, the greater will be the demand for dependable applications and also for new solutions that are variations of previously existing ones. One of the challenges of software development is to produce software that is designed to evolve, and so be extended, therefore reducing the maintenance efforts.

Nowadays, we do not have much support on the reuse of law specifications. We will give a simple example that specifies the interaction laws between two communities. In order to specify this example, two group of laws elements are available, one for each community. The laws show how the communication between two communities is disabled unless the exchanges of specific messages is allowed. We borrowed this example from LGI homepage [13]. We can specify it in XMLaw (Figure 1) without considering any support to extend or configure a basic definition that can be reused for both communities.

In this case, we can observe that copying / modifying / pasting the law specification will derive a semi-identical specification with few peculiarities. If we observe the first example, the code is practically the same between the two definitions (Figure 1**Erreur ! Source du renvoi introuvable.**); the only difference is the id and the name of the organization and scene and also the constraint that is used twice in each scene. With a simple refinement, we could have a basic description of this law and with little customization effort both specifications could be proposed.

As the first example shown, open MAS should be specified and developed to facilitate extensions and law-governed approaches should also present a solution to this concern. As open MAS need to be customized according to different purposes and peculiarities, it is possible to express extensions over interactions of software agents. In this paper, we will address the problem of how to prepare a law for extensions and how to refine law specifications. For this purpose, we enhanced the XMLaw description language [15] with some refinement operators to support those requirements and to realize a mapping of a customizable specification to the monitoring of interaction rules of governance mechanisms.

The main contribution of this proposal is to provide extensibility support within the interaction specification and compliance verification in open systems applications. For example, the operators allow the extension of the interaction laws including new services to run during the interaction monitoring and with filters to validate or not a message or norm. Finally, the improvements on XMLaw were mapped to a law-governed mechanism that interprets those descriptions, plus its specializations, and analyzes the compliance of software agents that inhabit open software systems.

Figure 1 Copy and Paste problem in XMLaw

The organization of this paper is as follows. Section 2 describes an example of an evolvable open MAS that is used to describe our proposal. Section 3 details the law-governed approach, its architecture and some elements of the XMLaw description language [15]. In, Section 4, we discuss variations in open MAS interactions and we describe how we included refinement operators in XMLaw. Section 4 also explains some examples of extension points identified in TAC-SCM's editions and we show how XMLaw can be used to support extensibility in a compliance mechanism. Related work is described in Section 5. Finally, we evaluate this approach and describe some future work and our conclusions in Section 6.

2. An Example of Evolvable Open MAS

A proof of concept prototype has been developed based on the specification of the Trading Agent Competition - Supply Chain Management (TAC SCM). TAC SCM [3][6][17] editions provide some evidences that the interaction specification evolves over time and so an extension support can reduce maintenance efforts.

The TAC SCM [3] has been designed with a simple set of rules to capture the complexity of a dynamic supply chain. SCM applications are extremely dynamic and involve an important number of products, information and resources among their different stages. In our case study, we mapped the requirements of TAC SCM into interaction laws and agents are implemented with JADE [4].

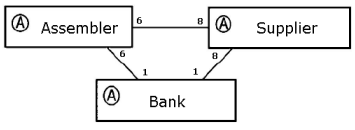

Figure 2 - Roles, relationships and cardinalities of TAC SCM

In TAC SCM, we chose the scenario of negotiation between the suppliers and assemblers to explain how extensions on interaction laws are used (**Figure 2**). According to [6], the negotiation process involves an assembler agent that buys components from suppliers. A bank agent also participates in this negotiation because an assembler must pay the components for the supplier. In this scenario, an assembler may send RFQs to each supplier everyday to order components offered by the supplier. Each RFQ represents a request for a specified quantity of a particular component type to be delivered on a specific date in the future. The supplier collects all RFQs received during the "day" and processes them. On the following "day", the supplier sends back to each agent an offer for each RFQ, containing the price, adjusted quantity, and due date. If the agent wishes to accept an offer, it must confirm it by issuing an order to the supplier.

3. Governing Interactions in Open Systems with XMLaw

Law governed architectures can be designed to guarantee that the specifications of open systems will be obeyed. We developed an infrastructure which includes a communication component that is provided to agent developers [16]. This architecture is based on a pool of mediators that intercept messages and interpret the laws previously described. As more clients are added to the system, additional mediators' instances can be added to improve throughput.

The core of a law governed approach is the mechanism used by the mediators to monitor the conversations between components. We have developed a software support [16] that permits whenever necessary to extend this basic infrastructure to fulfill open system requirements or interoperability concerns. Distributed software agents are independently implemented, i.e., the development is done without a centralized control. We assume that every agent developer may have a priori access to the open system specification, including the protocol descriptions and the interaction laws.

In this section, we explain the XMLaw description language [15]. Here, XMLaw is used to represent the interaction rules of an open system specification. Those rules are interpreted by a mechanism that at runtime analyzes the compliance of software agents to interaction laws [16].

XMLaw represents the structure and the relationships of important law elements (Figure 3). A law specification is a description of law elements. Law elements are interrelated in a way that it is possible to specify interaction protocols using time restrictions, norms, or even time sensitive norms.

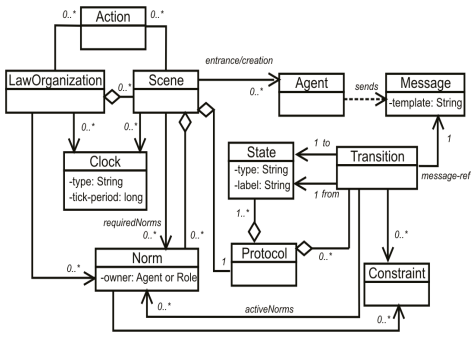

Figure 3 Conceptual Model

In this section, we describe how norms are used to enhance scene and transition definitions; how constraints in norms and transitions act as filters of events; and how actions are used as an adaptation mechanism to support an active behavior of the environment in an open system. For further details on each of the concept appearing in the conceptual model please refer to [15]. Below, we will discuss XMLaw structure using the specification of laws for TAC SCM example to facilitate its understanding.

Statically, an interaction protocol defines the set of states and transitions (activated by messages or any other kind of event) allowed for agents in an open system. Norms are jointly used with the protocol specification, constraints, actions and also temporal elements, to provide a dynamical configuration for the allowed behavior of components in an open system. Norms prescribe how the active distributed software components ought to behave, and specify how they are permitted to behave and what their rights are. To verify the compliance of software agents, the mediator keeps information about the set of activated norms, the set of deactivated norms, and any other data regarding the system execution.

There are three types of norms in XMLaw: obligations, permissions and prohibitions. The obligation norm defines a commitment that software agents acquire while interacting with other entities. For instance, the winner of an auction is obligated to pay the committed value and this commitment might have some consequences to avoid breaking this rule. The permission norm defines the rights of a software agent in a given moment, e.g. the winner of an auction has permission to interact with a bank provider through a payment protocol. Finally, the prohibition norm defines forbidden actions of a software agent in a given moment, for instance, if an agent does not pay its debts, it will not be allowed of future participations in a scene.

In TAC SCM, one permission norm about the maximum number of requests for quotation that an assembler can submit to a supplier was created. According to TAC SCM specification [6], each day, each agent may send up to a maximum number of RFQs. Besides this permission, the constraint over the acceptable due date of a RFQ regulates the same interaction, the request for quote message.

The structure of the Permission (Listing 1), Obligation and Prohibition elements are equal. Each type of norm contains activation and deactivation conditions. In that example, an assembler will receive the permission when it logged in the scene (scene activation event) and will lose the permission after emitting an order (event orderTransition). Besides, norms define the agent role that owns it through the attribute Owner. In that case, the assembler agent will receive the permissions. Norms have also constraints and actions associated with it, but those elements will be explained later. Norms also generate activation and deactivation events. For instance, as a consequence of the relationship between norms and transitions, it is possible to specify which norms must be made active or deactivated for firing a transition. In this sense, a transition could only fire if the sender agent has a specific norm.

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

Listing 1: Permission Structure

Constraints are restrictions over norms or transitions and generally specify filters for events, constraining the allowed values for a specific attribute of an event. For instance, messages carry information that is enforced in various ways. A message pattern enforces the message structure fields [15]. However, message pattern does not describe what the allowed values for specific attributes are, but constraints can be used for this purpose. Constraints are expressed using Java code. In this way, developers are free to build as complex constraints as needed for their applications.

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

Listing 2: Constraint in Transition Tags

Constraints are defined inside the Transition (Listing 2**Erreur ! Source du renvoi introuvable.**) or Norm (Listing 1) elements The Constraint element defines the class attribute that indicates the java class that implements the filter. This class is called when a transition or a norm is supposed to fire, and basically the constraint analyzes if the message values or any other events' attributes are valid. In **Erreur ! Source du renvoi introuvable.**, a

constraint will verify if the date expressed in the message is valid according to TAC rules; if it is not, the message will be blocked. In Listing 1, a constraint is used the verify the number of messages that the agent has sent until now, if it has been exceeded the permission is not valid anymore..

Environment actions, or just actions, are domain-specific Java code that runs integrated with XMLaw specifications. Actions can be used to plug services in an environment. For instance, an environment can call a debit service from a bank agent to automatically charge the purchase of a good in a negotiation.

Since actions are also an XMLaw element, they can be activated by any event such as transition activation, norm activation, and even action activation. An action can be activated by as many events as we wish. The action structure is showed in the example of Listing 1. The class attribute of an Action specifies the java class in charge of the functionality implementation. The Element tag references the events that activate this action, and as many Element tags as needed can be defined to trigger an action. In this example, the action is used to update the context of the norm, counting the number of submitted messages.

An action can be defined in three different scopes: Organization, Scene and Norms. An action defined in a Norm is only visible at this level, this means that any element in this scope can reference events issued by this action and that this action can get and update information at this level and upper levels. Actions defined in the scene scope can be referenced by any element at this level. And actions defined in the organization scope are visible by all elements in this level.

4. Refinement Operators to Specify Laws in Open Multi-Agent Systems

The definition of how the agents interact is very important to understand the open MAS behavior. The interaction specification is used as guidelines to enforce the expected behavior of agents in open MAS. Sometimes, the interaction laws that enforce the relationships between agents are not always fully understood early in the open MAS life cycle. Still, many more interaction laws are not applied, because of a lack of systems support for changing interaction laws (i.e. extensibility) or because the interaction laws are exceptionally complex.

We argue that the interaction laws of open MAS should also be specified and developed to facilitate extensions to deal with this challenge. 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. In our context, it is useful to permit the inclusion of norms, constraints and actions into a pre-defined law specification. Even with extension points, the semi-complete law element specification can be referenced by other law elements.

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 in open systems. The first attempt to define extension points was deferring the definition of the class implementation [5], in contrast with the action specification showed in Listing 1**Erreur ! Source du renvoi introuvable.**. The same applies to constraints, instead of defining a reference to a class, we defer it definition (Listing 3).

```
<Action id="orderID"/>
   <Element ref="rfqTransition"
            event-type="transition_activation"/>
</Action>
   ...
<Constraints>
   <Constraint id="checkDueDate"/>
</Constraints>
```

Listing 3: Action and Constraint hook

The subsections below explain how the interaction specification with extension points can be prepared to further refinements. Before in [5], the extensions were restricted to action and constraint definitions and it was not clearly defined when an element was an abstract element. In this sense, we propose three new operators to facilitate refinements in XMLaw specification: abstract, completes and extends.

4.1 Identifying Extension Points: abstract

We made some improvements on the attempt [5] towards extension points. Before, it was not clearly documented which element could be extended, the designer had to find out where were the semi-complete specifications by browsing actions and constraints that did not reference a concrete class implementation.

The abstract attribute defines when a law element is not completely implemented; 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. If no value is determined, the element is a concrete one (default abstract=”false”) or the designer can optionally specify that abstract=”false”. If he 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 also a means to achieve extension point idea, defining clearly the context where the extensions are expected. We still can defer the definition of the implementation of actions and constraints classes, as well as, we can define other law elements as abstract, and as we will see we can also extend their definition by including new or superposing elements.

In TAC SCM, the constraint checkDueDate (Listing 4**Erreur ! Source du renvoi introuvable.Erreur ! Source du renvoi introuvable.**) is associated with the transition rfqTransition. It means that if the verification is not true the transition will not be fired. The decision regarding the implementation of the checkDueDate constraint is deferred and so no class is specified in Listing 4**Erreur ! Source du renvoi introuvable.**.

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

Listing 4: Permission and Constraint over RFQ message

In Listing 5**Erreur ! Source du renvoi introuvable.**, there are two extension points: the constraint checkCounter and the action orderID. To customize this constraint and this action, we need to

plug-in the class implementation. The constraint checkCounter is an extension point that 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 **Erreur ! Source du renvoi introuvable.**is defined under the permission AssemblerPermissionRFQ and it is triggered by a clock-tick everyday, turning to zero the value of the counter of the number of requests issued by the assembler in this day. We do not give further details regarding the clock definition. The other action orderID **Erreur ! Source du renvoi introuvable.**is also an extension point and it is activated by every transition transitionRFQ. It is used to count the number of RFQs issued by the assembler, updating a local variable.

```
<Permission id="AssemblerPermissionRFQ"
            type="abstract">
   <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 5: AssemblerPermissionRFQ Norm description

Another example of extension is given in the specification of the relationship between orders and offers of the negotiation protocol. According to Collins et al. [6], agents confirm supplier offers by issuing orders. After that, an assembler gains a commitment with a supplier, and this commitment is expressed as an obligation. It is expected that suppliers receive a payment for its components. This requirement specifies the structure of the ObligationToPay obligation (Listing 6), 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 (Listing 7**Erreur ! Source du renvoi introuvable.**).

```
<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 6: Obligation to pay specification

```
<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 7: ObligationToPay usage

4.2 Filling the gaps: completes

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.

Below, we present the refinements proposed to the law described above. In TAC SCM 2005 [6], 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. For this refinement, another action component named RFQCounter2005 is plugged-in (Listing 8**Erreur ! Source du renvoi introuvable.**). It counts the number of RFQs according to the type of component. The constraint CounterLimit2005 was also chosen a specific counter for each type of component that a supplier provides.

```
<Permission id="APRFQ2005"
            completes="AssemblerPermissionRFQ">
   <Constraint id="checkCounter"
class="tacscm.norm.constraints.CounterLimit2005"/>
   <Action id="orderID"
class="tacscm.norm.actions.RFQCounter2005">...</Ac
tion>
</Permission>
```

Listing 8: Permission AssemblerPermissionRFQ extension

An RFQ with DueDate beyond the end of the game will not be considered by the supplier. RFQs with due dates beyond the end of the game, or with due dates earlier than 2 days in the future, will not be considered. This requirement is implemented by the constraint ValidDate2005 (Listing 9**Erreur ! Source du renvoi introuvable.**). Notice that if we want to extend this law to other editions of TAC SCM, we just need to define and associate new implementations of these actions and constraints.

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

Listing 9: Constraint checkDueDate extension

4.3 OO specialization: extends

The **extends** attribute is a more powerful operator and it is similar to the specialization operation in object-oriented languages (e.g extends in Java).

Basically, the 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, new norm elements and can also superpose any element that was previously specified. Similarly to **completes**, the **extends** operator turns an abstract element into a complete one and can not leave any element unspecified unless it redefines this element as an abstract one.

According to [6], suppliers wishing perhaps to protect themselves from defaults, will bill agents immediately for a portion down of the cost of each order placed. The remainder of the value of the order will be billed when the order is shipped. In TAC SCM 2005, the down payment ratio is 10%. This down payment is implemented by the action SupplierPayment (Listing 10**Erreur ! Source du renvoi introuvable.**). Notice that we have added a definition regarding the existence of an action in the context of the obligation definition.

```
<Obligation id="ObligationToPay2005"
            extends="ObligationToPay">
   <Actions>
      <Action id="supplierPayment"
      class="tacscm.norm.actions.SupplierPayment">
         <Element ref="orderTransition"
               event-type="transition_activation"/>
      </Action>
   </Actions>
</Obligation>
```

Listing 10: ObligationToPay extension for TAC SCM 2005

4.4 Execution order of law elements after extensions

In XMLaw, the composition and interrelationship among law elements is done by events. Every law element is related to events, one law element can generate events to signal something to other elements (Figure 4). Other elements can sense events for many purposes, for instance, activating or deactivating themselves, and so on. In this sense, the order of execution must be considered and clearly understood even with extensions, i.e, we need to clearly define the order of elements' activation in the case of using the refinement operations.

Figure 4 - XMLaw Event Model

The event monitoring model is implemented using the observer pattern [9]. During law interpretation, new elements are attached to the observer structure. Laws are interpreted at least in two steps, first the base law is read and then extensions are attached to the execution model. In this sense, any new element defined after an extension is associated with the execution model after the basic elements and then their activation will occur after the activation of basic elements.

Let us consider the example scenario (Figure 5) where you have hierarchy of extensions for a payment policy that consider the client importance for the bank The discount action will calculate the percentage of discount that a client can have on his payment (it is possible to accumulate discounts). During system execution, suppose that the monitoring mechanism receives an event PAYMENT from a PRIME Client, and then the expected order of activation of actions will be discount 10%, discount 5% and finally discount 20%. If the product cost is 100 units, at the end of this execution it would cost 68, 4 units.

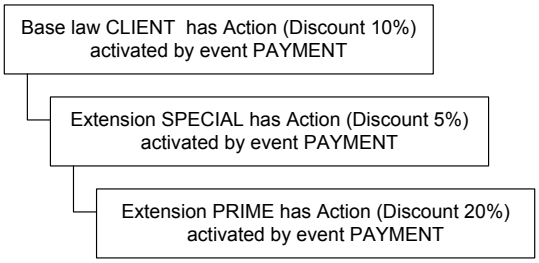

Figure 5 - Example of the execution order of law elements

5. Related Work

In Esteva [7] approach, scenes and protocol elements specify the interaction protocol using a global view of the interaction. XMLaw includes the concept of actions, which allows execution of Java code in response to some interaction situation, and we use them to implement the extension points.

Minsky [2][14] proposes a coordination and control mechanism called law governed interaction (LGI). This mechanism is based in two basic principles: the local nature of the LGI laws and a decentralization of law enforcement. It provides a language to specify laws and it is concerned with architectural decisions to achieve a high degree of robustness. In contrast, our approach provides an explicit conceptual model and focuses on different concepts such as Norms and also interaction extensibility support. Ao and Minksy [2] propose an approach to enhance LGI with the concept of policy-hierarchy to support that different internal policies are formulated independently of each other, achieving by this means a flexibility support. Differently from our approach, Ao and Minsky consider confidentiality as a requirement for their solution. The extensions that we have presented until now has the goal of supporting open system law maintenance, instead of flexibility for confidentiality purposes. One interesting characteristic that we would like to bring to XMLaw is the possibility of defining limits or how extensions can be redefined.

COSY [10] 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 [12], protocols inherit from one another. They are described as scripts containing the various steps of a possible sequence of interactions. Beliefs also are embedded into scripts. Koning and Huget [11] deal with the modeling of interaction protocols for multi-agent systems, outlining a component-based approach that improves flexibility, abstraction and protocol reuse. All of the approaches listed in this paragraph are useful instruments to promote reuse, they can be seen as instruments for specifying extendable laws.

Singh [19] proposes a customizable governance service, based on skeletons. His approach formally introduces traditional scheduling ideas into an environment of autonomous agents without requiring

unnecessary control over their actions, or detailed knowledge of their designs. Skeletons are equivalent to state based machines and we could try to reuse their formal model focusing on the implementation of extensions. But [19] has few implementation details and examples which could allow us to understand how his proposal was implemented.

6. Conclusions and Future Work

We are addressing the problem of constructing governance mechanisms that ensure that agents will conform to a well defined customizable specification. Our main goal is to contribute on the engineering on how we can productively define and reuse laws. We are also contributing with the study on how to engineer governance mechanisms development. With the refinement operators, we support the design of law elements for extension.

While analyzing the open software system domain, it is possible to distinguish two groups of specifications concerning agent's interactions: fixed (stable) and flexible (extensible). By this analysis, it is possible to design part of the open system evolution in the solution. If a desired characteristic of a system is long-term stability, then the challenge to developers is to deliver a product that identifies the aspects of the open MAS that will not change and cater the software to those areas. Besides some basic services, in open systems, system 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. It is our interest to continue to research these topics, so we will continue to enhance XMLaw to support interaction extensibility specification.

We are aware of possible problems about consistency when redefining or extending laws. We are dealing with this problem through the definition of a formal framework that enables us to check possible inconsistencies. However, a deeper discussion is out of scope of this paper.

7. 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).

8. References

[1] Agha, G. A. (1997) 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.

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

[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] 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

[5] CARVALHO, Gustavo; PAES, Rodrigo; LUCENA, Carlos; Extensions on Interaction Laws in Open Multi-Agent Systems. First Workshop on Software Engineering for Agent Oriented Systems, Brazilian Symposium on Software Engineering (SBES2005). Uberlândia, Brazil, 2005.

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

[7] Esteva, M. (2003) Electronic institutions: from specification to development, Ph.D. thesis, Institut d'Investigació en Intelligència Artificial, Catalonia - Spain.

[8] Fredriksson M. et al. (2003) First international workshop on theory and practice of open computational systems. In Proceedings of twelfth international workshop on Enabling technologies: Infrastructure for collaborative enterprises (WETICE), Workshop on Theory and practice of open computational systems (TAPOCS), pp. 355 - 358, IEEE Press.

[9] Erich Gamma, Ralph Johnson, Richard Helm, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1995.

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

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

[12] 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.

[13] LGI homepage. http://www.moses.rutgers.edu/ . Visited in 12/01/2005.

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

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

[16] Paes, R.B; Lucena, C.J.P; Alencar, P.S.C. (2005b). A Mechanism for Governing Agent Interaction in Open Multi-Agent Systems. http://www.les.inf.puc-rio.br/governance/pubs.html

[17] Pree,W.Essential Framework Design Patterns.Object Magazine 1997

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

[19] 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.

[20] Wooldridge, M; Weiss, G; Ciancarini, P. (Eds.) (2002) 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.

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