# Uma Abordagem para o Reuso de Leis de Interação em Sistemas Multi-Agentes

**Gustavo Robichez de Carvalho[1], Ricardo Choren[2], Rodrigo Paes[1], Carlos Lucena[1]**

[1]Departamento de Informática – PUC-Rio
Rua M. de São Vicente 225 – 22.453-900 – Rio de Janeiro – RJ – Brasil

[2]Departamento de Engenharia de Computação – IME/RJ
Pça Gen Tibúrcio 80 – 22.290-270 – Rio de Janeiro – RJ – Brasil

{guga, rbp, lucena}@inf.puc-rio.br, choren@de9.ime.eb.br

***Resumo.*** *Um dos desafios de desenvolvimento de software é produzir aplicativos que são projetados para evoluir reduzindo esforços de manutenção. Diversas técnicas desenvolvidas para a governança de leis de interação em sistemas multi-agentes (SMAs) foram desenvolvidas, no entanto a flexibilidade e a reutilização de leis não ocorriam de forma sistemática. Este artigo apresenta a técnica, proposta pelos autores, para a documentação de g-frameworks, um mecanismo para promover a flexibilidade e a reutilização de projetos e de implementações de leis de interação em SMAs. Esta flexibilidade é obtida através da introdução de incrementos específicos que as instâncias em desenvolvimento requerem, de modo a completar e adaptar as funcionalidades originais do g-framework. A reutilização vem justamente do re-aproveitamento de um mesmo projeto e código de lei de interação em instâncias geradas a partir do g-framework.*

***Abstract.*** *A major challenge for software development is to create software that is designed to evolve, reducing maintenance costs. Several approaches for multi-agent system (MAS) governance were developed, but these approaches do not provide support for systematic interaction law reuse. This paper discusses the g-framework approach for providing flexibility and reusability of MAS interaction law design and implementation. Flexibility and reuse are achieved by introducing specific add-ons to adapt the core functionalities of a g-framework thus allowing the instantiation of different applications from the same design.*

## 1. Introdução

O desenvolvimento de sistemas complexos e interativos é uma realidade. Em um cenário de ambiente aberto, agentes de software podem entrar e sair e não existe um controle único e centralizado sobre o seu desenvolvimento. Isto pode acarretar em riscos que tornam o ambiente imprevisível. Para limitar o grau de imprevisibilidade desses ambientes é possível estabelecer mecanismos que observam o comportamento dos agentes e fazem uma comparação desse comportamento com o comportamento esperado para o sistema.

No caso de sistemas multi-agentes (SMAs), a abordagem para limitação de imprevisibilidade na execução dos agentes é chamada de governança. Os agentes de software precisam estar integrados e interagindo, para realizar suas atividades. O comportamento

previsto é descrito em leis de interação que são interpretadas por um mecanismo responsável pela interceptação de mensagens e observação da conformidade com as leis.

Em SMAs regulados por leis, a forma com que os agentes interagem pode variar ao longo do tempo. Cada interação em um sistema aberto pode ser considerada um ponto de extensão em potencial, no qual é possível se definir restrições e regras que indicam as condições sobre como e quando uma interação é válida. Ao observar as interações é possível perceber que os protocolos correspondentes ao fluxo de conversação em diferentes aplicações de um mesmo domínio apresentam características comuns e pequenas variações que se bem identificadas podem ser isoladas de forma a promover o reuso de seus projetos e implementações.

Existem diferentes modelos para governança de leis de interação em SMAs [Dignum 2001, Esteva 2003, Minsky and Ungureanu 2000, Paes et al. 2005]. Associados a estes modelos, implementações de mecanismos apóiam a garantia da conformidade a estas regras. Estes modelos variam quanto às diferentes granularidades de elementos, diferentes abstrações e diferentes formalismos associados. No entanto, não existe abordagem que apóie, de forma efetiva e sistemática, a produção de leis de interação preparadas para o reuso de forma flexível. Falta ainda orientação de como é possível projetar ou reutilizar aplicações já desenvolvidas com este propósito. Neste sentido, a documentação é um dos pontos fundamentais.

Com o intuito de contribuir para um melhor projeto de leis de interação, para que seja possível reutilizar as partes e customizar as leis dando um suporte explícito a flexibilidade de seus elementos, pretende-se aplicar e adaptar o conceito de framework OO [Batory et al. 2000]. Este trabalho apresenta a abordagem de g-frameworks para a governança de SMAs. Nossa premissa é permitir a flexibilização e o reuso de leis de interação, a partir da modificação dos artefatos de governança de um SMA de forma fácil, para que seja possível aplicá-los em diferentes contextos para a produção de mecanismos de governança.

As seções seguintes deste artigo estão organizadas da seguinte maneira: na Seção 2 resume-se os principais conceitos que fundamentam este trabalho; na Seção 3 apresenta-se o método de desenvolvimento de frameworks de governança para SMAs; na Seção 4, os benefícios que esta abordagem traz para a modelagem de requisitos são apresentados através de um exemplo ilustrativo; alguns trabalhos relacionados a este são apresentados na Seção 5 ; por fim, as conclusões e futuros trabalhos são relatados na Seção 6.

## 2. Governança de Sistemas Multi-agentes Abertos

A abordagem apresentada neste artigo se aplica à modelagem e implementação de leis de interação em SMAs. Para isto, utilizamos o XMLaw [Paes et al. 2005] e alguns conceitos utilizados na documentação de frameworks OO [Batory et al. 2000]. O XMLaw e estes conceitos serão apresentados nas subseções a seguir.

### 2.1. XMLaw

O XMLaw [Paes et al. 2005] é uma linguagem para expressar possibilidades e restrições sobre interações em um SMA. A observação prevista no modelo de XMLaw é baseada em eventos, portanto o evento de uma mensagem é algo perceptível no contexto de leis. Isto serviu de base para a proposição dos elementos presentes atualmente no modelo

conceitual de XMLaw (Figura 1). Estes elementos correlacionados compõem uma lei de interação e são descritos utilizando uma linguagem baseada em XML (Figura 2).

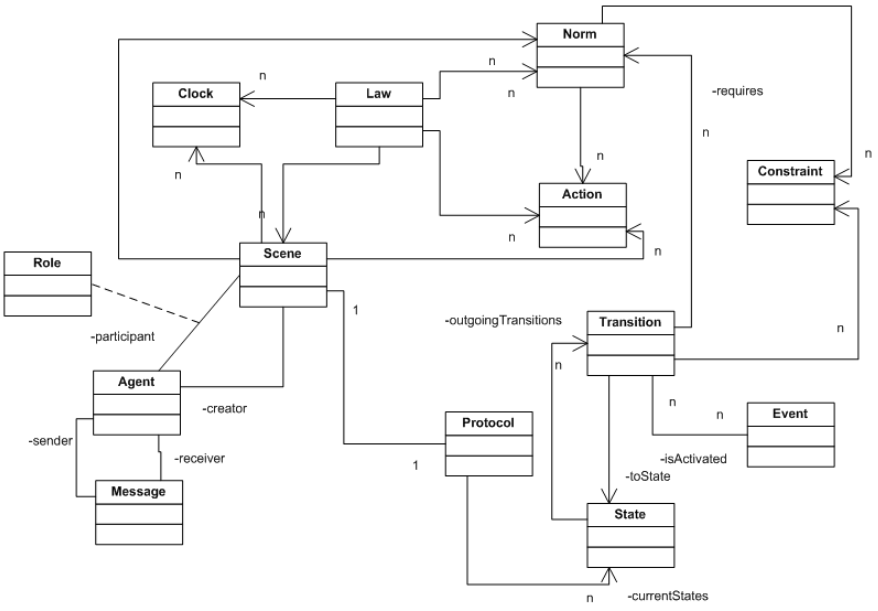



**Figura 1. Modelo conceitual de XMLaw**

```
<Law 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>
 </Law>
```

**Figura 2. Estrutura de um arquivo com código XMLaw**

O primeiro fato importante a observar é que os elementos de XMLaw não só percebem como se comunicam através desses eventos. A cadeia de eventos pode ser responsável por um conjunto de ativações ou desativações relacionadas ao ciclo de vida dos elementos, e previstas segundo as dependências pré-estabelecidas nos elementos. Um exemplo simples é a chegada de uma mensagem que gera um evento (message_arrival) e que por sua vez pode ativar uma transição (transition_activation). Abaixo descreveremos de forma ilustrativa os elementos *Message*, *Protocol*, *Transition*, *Clock* e *Norm*.

O elemento *Message* corresponde à descrição da estrutura e do conteúdo de mensagens enviadas ou recebidas por agentes. Ele estabelece o padrão esperado para a observação de mensagens de agentes, prescrevendo a ordem em que as mensagens

ocorrem e quais são as mensagens válidas em um determinado momento da interação, definindo um protocolo de interação.

Protocolos de interação representam o fluxo de estados previstos em uma conversão de agentes, definindo quais são as interações válidas e inválidas. O elemento *Protocol* representa protocolos de interação entre os agentes e é representado por um autômato finito não determinístico [Menezes 1997], no qual estados (elemento *State*) representam pontos na execução do protocolo e transições (elemento *Transition*) são as conexões entre os estados. O protocolo ao ser acionado evolui do estado origem da transição para o destino. Uma transição pode ter uma restrição associada a ela (elemento *Constraint*). Neste caso, a transição só será ativada quando a restrição permitir. Uma restrição é implementada como um componente Java. Uma transição pode ainda requerer que um elemento *Norm* (*Prohibition*, *Obligation* ou *Permition*) esteja ativo para que o protocolo evolua para outro estado. Normas (elemento *Norms*) descrevem quais comportamentos dos agentes são permitidos, obrigatórias ou proibidos. As normas são adquiridas pelos agentes no decorrer das interações, e conseguem representar noções de compromissos adquiridos e cumpridos.

O modelo conceitual fornece um elemento denominado relógio (elemento *Clock*) que provê os meios para especificar leis sensíveis ao tempo. Um relógio é capaz de gerar eventos em intervalos de tempo específicos. O modelo conceitual utiliza a abstração de cenas (elemento *Scene*) para auxiliar na organização e modularização das interações. Este elemento especifica quais agentes e quais os papéis de agentes que podem interagir em uma cena, ou mesmo dar início a sua execução. Além disso, uma cena é composta por um protocolo de interação e por um conjunto de normas, ações e relógios. Estes elementos compartilham um contexto comum de interação definido pela cena. Isto significa que uma norma definida no contexto de uma cena é somente visível naquela cena. Para que um agente inicie ou participe de um protocolo de interação de uma cena, é necessário que ele desempenhe algum papel previamente definido e especificado no contexto da cena.

### 2.2. Documentação de Frameworks G-Frameworks a partir de OO

Frameworks orientados a objetos representam uma tecnologia relevante para a implementação de arquiteturas de famílias de programas ou linhas de produto [Pree 1997]. Eles permitem tanto o reuso em larga escala quanto modular, através do encapsulamento de funcionalidades recorrentes de um dado domínio [Fayad et al. 1999]. Em geral, a adoção da abordagem de frameworks pode trazer expressiva produtividade e qualidade para o desenvolvimento de novas aplicações [Batory et al. 2000].

A documentação é de suma importância, uma vez que além do usuário da documentação intitulado mantenedor/desenvolvedor, existe também o reutilizador que normalmente não necessariamente é o desenvolvedor original e conseqüentemente pode não ser um expert no domínio da aplicação e no projeto proposto. Sendo assim, a documentação dos componentes reutilizáveis precisa apresentar de forma clara, não só suas estruturas internas, mas também como sua (re) utilização deverá proceder.

Em [Johnson 1992] é afirmado que a documentação de um framework deve abranger o propósito do framework, o design, o propósito das aplicações exemplo, e instruções sobre como utilizar a proposta (Figura 3). O campo **propósito do framework** é uma descrição informal da utilidade do framework em questão, a partir de textos não

estruturados. Sua utilidade localiza-se normalmente na fase de análise de domínio para a verificação da reutilizabilidade no contexto em análise. O campo **design do framework** é uma descrição dos elementos presentes na solução, bem como suas interações. Normalmente, esta informação é consultada na fase de reutilização/integração. O **propósito das aplicações** exemplo é uma descrição do propósito e funcionamento das aplicações exemplo. Serve para o reutilizador conhecer o cenário geral do framework e às vezes é o ponto de partida para o desenvolvimento da nova aplicação. Pode ser utilizada na fase de domínio para uma verificação do potencial de reutilização, e na fase de implementação, como um guia para a reutilização. Por fim, **como usar o framework** descreve como o reutilizador deve proceder durante o processo de instanciação.

| **Propósito do Framework:** <descrição> |
|---|
| **Design do framework:** <design> |
| **Propósito das aplicações exemplo:** <propósito> |
| **Como usar**: <como_usar_framework> |

**Figura 3. Resumo da documentação proposta em [Johnson 1992]**

O reutilizador precisa conhecer os ponto de extensão e a forma usada para entendê-los. Cookbooks [Krasner and Pope 1988] são uma abordagem que auxilia este processo, permitindo que o projetista do framework especifique como o framework deve ser instanciado. Primeiramente esta forma de documentação, descreve o framework de forma geral, através de linguagem natural, e em seguida descreve as partes relevantes à instanciação. Por último apresenta uma série de exemplos que utilizam o framework (Figura 4).

| Descrição Geral do Framework: <descrição> |
|---|
| Partes Relevantes à Instanciação: <design> |
| Exemplos de Uso do Framework: <como_usar_framework> |

**Figura 4. Resumo da documentação proposta em [Krasner and Pope 1988]**

Hooks [Froehlich et al. 1997] são considerados um aprimoramento de Cookbooks uma vez que descrevem pontos de extensão de forma mais estruturada, detalhando aspectos relevantes como participantes no hook e alterações necessárias para usar o hook. Em sua estrutura, Hooks descrevem o nome único no contexto do framework, dado para cada hook (**Nome**); o problema que o hook soluciona (**Requisito**), um par ordenado que especifica o método de adaptação e apoio fornecido pelo framework (**Tipo**); as partes do frameworks que são afetadas pelo hook (**Áreas**); outros Hooks necessários para se usar este hook (**Usa**); os componentes que participam desse hook, existentes ou criados pelo reutilizador (**Participantes**); seção principal do hook e descreve as alterações que devem ser feitas nos componentes que participam de Hooks, esta seção descreve as alterações como herança e redefinições de métodos (**Mudanças**); indicam as restrições impostas ao uso do hook (**Restrições**) e descrições adicionais necessárias (**Comentários**).

Nossa abordagem utiliza os conceitos de documentação de frameworks, Cookbooks e Hooks na documentação de g-frameworks. Para os nossos propósitos, criamos os

| | |
|---|---|
| **Nome** <nome_identificador> | **Versão** <número da versão do framework> |
| **Propósito** <descrição> | |
| **Design** <descrição geral de características fixas e pontos de extensão > | |
| **Pontos de extensão** <lista com pares [elemento : nome] dos pontos de extensão> | |
| **Como instanciar** <como_instanciar_framework> | |

**Figura 5. Estrutura de documentação de um g-framework**

campos nome, propósito, versão, design e como instanciar (Figura 5). O campo textual **Nome** é um identificador do g-framework descrito, serve para facilitar a catalogação da solução gerada. O campo textual **Propósito** é uma descrição informal da utilidade do g-framework em questão, a partir de textos não estruturados. Um propósito é normalmente consultado na fase de análise de domínio para a verificação da reusabilidade no contexto em análise. O campo numérico **Versão** denota opcionalmente a versão do g-framework. O campo **Design** é uma descrição dos elementos presentes na solução, bem como suas interações. Normalmente, esta informação é consultada na fase de reutilização/integração. Por fim, o campo textual baseado em itens **Como instanciar** descreve como o reutilizador deve proceder durante o processo de instanciação, sendo uma enumeração seqüencial de passos em texto não estruturado.

O outro nível de detalhamento previsto em g-frameworks são os pontos de extensão, baseada na documentação de Hooks. Em sua estrutura (Figura 6), o campo textual **Nome** descreve o nome único no contexto do g-framework; o campo textual **Ameaça** identifica o problema que o ponto de extensão deve solucionar; um campo textual **Suporte** oferecido serve para especificar o apoio fornecido pelo g-framework; o campo **Dependência de Elementos** descreve pares elemento:identificador dos quais este ponto de extensão depende; o campo **Dependência de Eventos** descreve os pares evento:identificador aos quais este ponto de extensão depende ou referencia; o campo textual **Mudanças** é a seção principal desta documentação e descreve as alterações que devem ser feitas nos elementos de leis. É nesta seção que são detalhadas as instruções sobre as alterações necessárias para instanciá-lo como a inclusão e redefinição de elementos. Por fim, o campo textual **Restrições** indica as restrições impostas ao uso do ponto de extensão.

| | |
|---|---|
| **Nome** <nome_único_hot_spot> | |
| **Ameaça** <problema_solucionado> | |
| **Suporte Oferecido** <descricao_sucinta_hotspot> | |
| **Elementos criados** <referencia_elementos_criados> | |
| **Dependência de Elementos** <componentes_relacionados> | **Dependência de Eventos** <eventos_relacionados> |
| **Mudanças** <descrição_alterações_necessárias> | |
| **Restrições** <restrições_impostas_ao_uso> | |

**Figura 6. Estrutura de documentação de um ponto de extensão**

## 3. Um Método para o Desenvolvimento de G-Frameworks

É importante observar que o desenvolvimento de g-frameworks para SMAs governados por leis ocorre em duas fases. Na primeira fase, o objetivo é projetar e desenvolver as leis de interação com o propósito de propor e manter uma solução reutilizável. Esta etapa corresponde ao ciclo de vida de desenvolvimento do g-framework e compreende:

i. Análise do domínio de aplicação e documentação dos requisitos;
ii. Projeto, distinguindo o núcleo e os pontos de extensão de interação;
iii. Implementação detalhada utilizando operadores de refinamento;
iv. Documentação da solução para facilitar o reuso.

A segunda fase corresponde ao processo de escolha e instanciação do g-framework. A partir das documentações é possível identificar se o g-framework existente é adequado para a solução do problema desejado. Em caso afirmativo, os pontos de extensão devem ser detalhados e customizados para o propósito da aplicação em desenvolvimento. A partir do momento em que os pontos de extensão de interação estiverem detalhados é possível habilitar o mediador e dar início à execução do sistema aberto. Por fim, agentes de software podem entrar em cena e interagir segundo as regras definidas. É importante observar que na proposta do g-framework não existe orientação para a forma com que os agentes de software devem ser desenvolvidos.

### 3.1. Projeto de G-Frameworks

O projeto de g-frameworks (Figura 7) se resume em estruturar a análise de variabilidade elaborada em conjunto com a fase de requisitos em uma solução baseada em leis de interação que facilitem a manutenção de seus elementos, visando atingir uma solução coesa e que esteja preparada para a evolução identificada como necessária no estudo de variabilidade. Espera-se com a fase de projeto, que a arquitetura do g-framework esteja definida, incluindo o núcleo da solução e os seus respectivos pontos de flexibilização.

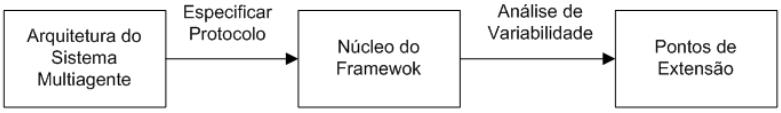



**Figura 7. Projeto de um g-framework para SMAs**

Neste processo é necessário classificar os requisitos como tendo características fixas (estáveis) ou flexíveis. Segundo a análise feita, caso não tenha sido possível identificar variações acerca do requisito, ele é considerado como estável dentro da solução e fará parte do núcleo da solução, isto é, ele terá uma mesma implementação para todos os mecanismos de governança gerados a partir desse g-framework. Se for possível antecipar a possibilidade e a natureza de variação, este requisito motivará a geração de um ponto de extensão na fase de projeto. Um ponto de extensão é a identificação do lócus de alteração, no qual a partir de um maior detalhamento podem ser geradas aplicações com especificidades próprias.

### 3.2. Implementação de G-Frameworks

A implementação de g-frameworks está relacionada à utilização de operadores de refinamento: *abstract*, *completes* e *extends*. Operadores de refinamento são instrumentos inseridos na linguagem XMLaw para apoiar o processo de desenvolvimento de g-frameworks, sendo responsáveis pela definição e materialização dos pontos de extensão. Estes operadores têm duas funções principais: (1) identificar elementos abstratos em uma implementação XMLaw; e (2) adaptar e estender elementos abstratos ou concretos. Estas duas funções viabilizam de forma prática o desenvolvimento de soluções semi-completas; isto é, é possível desenvolver parcialmente um mecanismo de governança, identificando claramente as lacunas que precisam ser preenchidas a posteriori.

O operador *abstract* foi criado para definir quando um elemento de lei não está completamente implementado. Se este atributo não for declarado, o elemento é considerado como concreto (default abstract = "false"). Se o analista quiser especificar que o elemento de lei precisa de mais refinamentos para que possa ser utilizado, é necessário que este atributo seja explicitamente associado ao valor true (abstract = "true"). Se um elemento de lei é definido como concreto, ele não deve deixar de completar nenhuma informação, isto é, a lei é considerada auto-contida e suficiente para a função de monitoramento. Isto quer dizer que a lei deve estar completamente implementada e o interpretador da lei deve indicar um erro caso encontre algum elemento carente de definição.

O operador *completes* é um operador criado para preencher lacunas em elementos que foram definidos como abstratos. É um operador simples que promove a extensão do elemento, através da definição das classes que implementam as ações e restrições. O operador *completes* transforma um elemento abstrato em concreto e neste sentido não pode deixar nenhum elemento para ser especificado a posteriori, a menos que este mesmo elemento seja redefinido como abstrato. O operador *completes* é limitado a materialização de classes Java de Constraints e Actions que estavam ausentes em um elemento abstrato, ele não pode incluir outros elementos de leis em um elemento abstrato.

O operador *extends* é um operador similar à operação de especialização em linguagens de orientação a objetos. Basicamente, um operador *extends* reutiliza a descrição de elementos de leis e inclui qualquer modificação que se faça necessária para a customização daquele elemento para a necessidade do usuário, incluindo a redefinição ou inclusão de novos elementos de leis. Por exemplo, este operador pode incluir novas referências de ativação, novas ações, novos elementos de normas, ou até sobrepor elementos que foram definidos a priori. O operador *extends* também torna um elemento abstrato em concreto e neste sentido não pode deixar nenhum elemento para ser especificado a posteriori, a menos que este mesmo elemento seja redefinido como abstrato.

## 4. Estudo de Caso: o TAC SCM

O TAC SCM [Collins et al. 2006] é uma competição que foi desenvolvida por pesquisadores do *Laboratório de e-Supply Chain Management da Carnegie Mellon University e Swedish Institute of Computer Science (SICS)*, e suas últimas edições foram realizadas em encontros oficiais da conferência AAMAS (Autonomous Agents and Multiagent Systems). Esta competição foi modelada para capturar a complexidade de uma cadeia de suprimentos dinâmica, porém com regras simples para que muitas equipes possam participar.

Seis agentes "produtores de PC" participam em cada jogo do TAC SCM. Esses participantes competem por clientes com incerteza na demanda e por peças de um número limitado de fornecedores. Todo dia, clientes enviam pedidos de cotações e selecionam as melhores ofertas baseadas no preço e na data de entrega. Os agentes são limitados pela capacidade da fábrica, e têm que gerenciar a compra de peças de oito fornecedores e a montagem de PCs para a entrega aos seus clientes. Quatro componentes são precisos para montar um PC: CPU, placa mãe, memória, e disco rígido. Cada componente está disponível em diferentes modelos.

O servidor do TAC SCM simula o comportamento dos fornecedores e clientes, e oferece um ambiente para cada participante com um banco que controla os pagamentos e as contas corrente, uma fábrica que monta os PCs para o agente, e informações sobre inventário como estoques de peças e PCs.

Analisando a variabilidade das interações na negociação entre fornecedores e montadores e a estrutura de pagamento dessa negociação nas últimas edições do TAC SCM, propusemos um projeto para a geração de um g-framework. Em resumo, o núcleo do g-framework é composto por uma cena para a negociação, uma cena para pagamento, a definição dos passos de interação (transições), estados para controle da conversação e de mensagens trocadas pelos agentes, e a permissão sobre o encadeamento e correlação entre duas mensagens da conversação (RFQ e OFFER). Os pontos de extensão do g-framework incluem a permissão associada a montadores para submeter requisições durante o período de um dia (número de requisições permitidas e a forma de sua contagem), as restrições para verificar a validade das datas de pedidos de cotação, e o método de pagamento implementado por ações dentro de uma obrigação criada.

### 4.1. Análise de Domínio e Documentação dos Requisitos

Inicialmente, o TAC SCM conta com dois agentes: montador e banco. No entanto, para demonstrarmos a aplicabilidade do g-framework, convertemos parte desses componentes de simulação em agentes externos que foram desenvolvidos. Assim, a arquitetura de nossa solução é composta dos agentes montador, banco, fornecedor, consumidor e fábrica, além do servidor TAC, cuja principal função passa a ser o monitoramento e a análise da conformidade do comportamento dos agentes em relação às leis que são válidas.

Depois de identificar os papéis de agentes que irão participar das interações governadas pelo framewok, é preciso documentar os requisitos do sistema. Neste estudo de caso, utilizamos a linguagem de modelagem ANote [Choren and Lucena 2005] para fazer a documentação do sistema. Na negociação, montadores compram componentes de fornecedores para a produção de PCs. O agente banco intermedia estes dois papéis de agentes. Seis agentes de montagem produzem PCs em cada rodada do jogo do TAC SCM. Estes agentes interagem tanto com agentes fornecedores quanto com o banco. Existem oito agentes fornecedores diferentes em cada cadeia de suprimento. Somente um banco é a entidade responsável por gerenciar pagamentos. O diagrama de classe de agentes (Figura 8) descreve os papéis, seus relacionamentos e suas cardinalidades.

Depois de modelar as classes de agentes que formam a arquitetura do sistema, é preciso modelar as interações entre estes agentes. É a partir da estrutura da interação entre os agentes que se determina o núcleo da conversação, correspondente ao núcleo do g-framework. Este núcleo incluirá a organização da conversação em cenas, protoco-

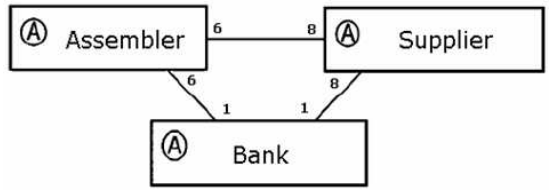



**Figura 8. Classes, relacionamentos e cardinalidades dos agentes no TAC SCM**

los e elementos de leis comuns a todas as instâncias do g-framework. Os protocolos de interação foram modelados por diagramas de interação do ANote.

### 4.2. Projeto do Núcleo e dos Pontos de Extensão do G-Framework

No estudo da descrição das edições do TAC SCM, observamos a existência de um núcleo fixo da interação que corresponde ao fluxo de conversação entre os agentes. A idéia é tornar este núcleo reutilizável em cada instância gerada a partir do g-framework. Decidimos organizar esta solução utilizando duas cenas: uma para a negociação entre os montadores e fornecedores, e outra cena para o pagamento envolvendo o agente montador e o banco.

A negociação entre montadores e fornecedores está diretamente relacionada ao processo de pagamento entre o montador e o banco. Resumidamente, o pagamento é feito por uma mensagem enviada pelo montador para o banco, que responde com uma mensagem de confirmação, representando o recibo da operação (Figura 9a). A negociação entre montadores e fornecedores se dá através de quatro mensagens (Figura 9b), cinco passos, e seis transições. De forma sucinta, a negociação é feita pelo envio de uma mensagem de pedido de cotações (RFQ), seguida de uma oferta de fornecimento (OFFER). A partir daí o montador pode solicitar a compra de suprimentos (order); e por fim, o fornecedor entrega os produtos (DELIVERY).

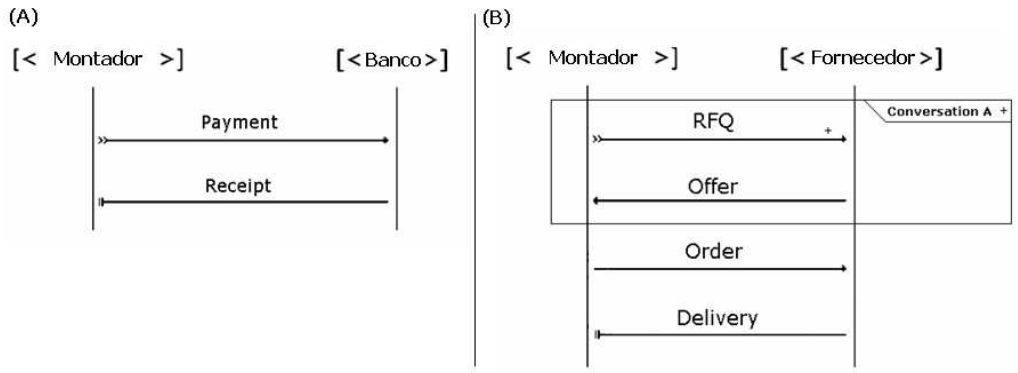



**Figura 9. Diagrama de interação de (A) pagamento e (B) negociação**

A identificação de uma permissão é descrita como forma de ilustrar outra natureza de informação incluída no núcleo do g-framework. Um exemplo de uma permissão presente no núcleo da lei de interação define a relação entre a mensagem request for quote (RFQ) enviada por um montador e a oferta subseqüente enviada por um fornecedor (OFFER). Com estes exemplos, ilustramos a natureza de um núcleo de um g-framework de

governança. Após definido o núcleo da solução, é possível descrever os pontos de extensão existentes no framework de governança.

Um ponto de extensão claro é a validação necessária, porém variável, dos atributos esperados em uma mensagem do tipo RFQ. A solução para este requisito é associar uma restrição a transição equivalente a este estágio da conversação. Segundo o histórico de alterações das edições do TAC SCM, as validações são feitas sempre sobre o atributo data de entrega (*dueDate*). Chamaremos este ponto de extensão de *checkDueDate*.

Outro ponto de extensão foi especificado para definir o relacionamento entre ordens e ofertas dentro de um protocolo de negociação. De acordo com [Collins et al. 2006], agentes confirmam ofertas de fornecedores através de ordens. Depois disto, um montador tem o compromisso para com o fornecedor. Este requisito pode ser representado como uma obrigação. Espera-se que os fornecedores recebam pagamento pelos componentes oferecidos. No entanto, não está completamente definido no núcleo da solução quando este pagamento será feito. Neste sentido, outro ponto de extensão é especificado na estrutura da obrigação: *ObligationToPay*.

### 4.3. Implementação do G-Framework

Conforme dito acima, as leis de interação dessa aplicação estão organizadas em duas cenas (negociação e pagamento). O Código 1a detalha a especificação inicial da cena que representa a negociação entre os agentes fornecedores e montadores. Cada cena de negociação é válida por um período de 3300000ms (220 dias x 15000ms) equivalente a um ciclo de negociação. Isto é uma restrição de domínio da aplicação e está representada pelo valor do atributo time-to-live. Já o Código 1b descreve o processo de pagamento. Como não existe uma restrição de domínio sobre o limite de tempo para esta cena, o atributo time-to-live desta cena possui o valor "infinity". O Código 2 apresenta detalhes da cena de negociação entre montadores e fornecedores, utilizando XMLaw.

(A)

```
<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>
```

(B)

```
<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>
```

**Código 1. Estrutura da cena de (A) pagamento e (B) negociação**

Já a implementação da lei que define a relação entre a mensagem *request for quote* (RFQ) enviada por um montador e a oferta subseqüente enviada por um fornecedor está ilustrada no Código 3. A permissão controla quando a mensagem de aceitação de oferta é válida na conversação, e é ativada depois de um evento de ativação da transição rfqTransition. Duas restrições são definidas no contexto da permissão: (i) uma para determinar as configurações válidas para os atributos de oferta que um fornecedor deve enviar para um montador, e (ii) a outra restrição verifica se uma mensagem de oferta foi enviada um

(A)

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

(B)

```
<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>
```

**Código 2. Descrição (A) das mensagens e (B) do protocolo de negociação**

dia após o recebimento do RFQ. Esta permissão só é válida se ambas as restrições são verdadeiras.

A partir da implementação do núcleo da solução é possível descrever e incluir os pontos de flexibilização previstos na solução. O ponto de extensão *checkDueDate* verifica se o atributo data está de acordo com as regras impostas pela edição da competição. Isto significa que se a mensagem não passar pela verificação (restrição = false), a transição não será ativada. Esta restrição está associada com a transição rfqTransition e esta transição é definida como abstrata para deixar clara a existência de um ponto de extensão (Código 4). Neste exemplo de ponto de extensão, foi necessário manter o atributo class da restrição *checkDueDate* não especificado, no qual a classe que implementa esta restrição só será conhecida na instanciação do g-framework.

Já a obrigação *ObligationToPay* será ativada por uma mensagem do tipo order, e será desativada com a entrega dos componentes, posterior ao seu pagamento. Um fornecedor irá entregar o produto somente se o montador tiver a obrigação de pagar por eles (Código 5). O montador pode somente entrar em uma cena de pagamento se ele tiver a obrigação de pagar pelo produto. Um montador não pode entrar em outra negociação se suas obrigações não tiverem sido cumpridas.

### 4.4. Documentação da Solução

Ao término de uma iteração para o desenvolvimento de um g-framework, é necessário documentar o resultado de forma que ele possa ser realmente reutilizado. A documentação da solução do TAC SCM seguiu a proposta apresentada na seção 2.2. O guia de referência do g-framework do TAC SCM está ilustrado na Figura 10. A Figura 11 descreve a documentação do ponto de extensão *ObligationToPay*.

(A)

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

(B)

```
<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>
```

**Código 3. Especificação (A) da transição e (B) da norma da mensagem RFQ**

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

**Código 4. Ponto de extensão *checkDueDate***

## 4.5. Instanciação do G-Framework: a Edição 2005 do TAC SCM

Na edição de 2005 do TAC SCM [Collins et al. 2004], fornecedores deviam receber um pagamento imediato de parte do valor (10%) da compra. O restante deste valor seria pago assim que o pedido fosse despachado. Na edição de 2004 [Arunachalam et al. 2004], o fornecedor deveria receber o pagamento do montador pela venda dos componentes somente após a entrega dos componentes. Esta é uma variação que demonstra a necessidade de se modelar e desenvolver a solução visando o reuso.

Para a instância do mecanismo de governança da edição de 2005, implementamos este pagamento como duas ações, uma referente ao pagamento da antecipação e outra referente ao pagamento do saldo ao final da negociação. A obrigação *ObligationToPay* foi estendida para incluir estas ações (Código 6a). Para o ponto de extensão *checkDueDate*, criamos um componente restrição *ValidDate2005* que verifica se o atributo *DueDate* (data limite) de uma RFQ não ultrapassa a data final de negociação (dois dias no futuro). Completamos a transição rfqTransition concluindo a extensão desse item (Código 6b).

```
<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>
```

**Código 5. Transições e norma de pagamento**

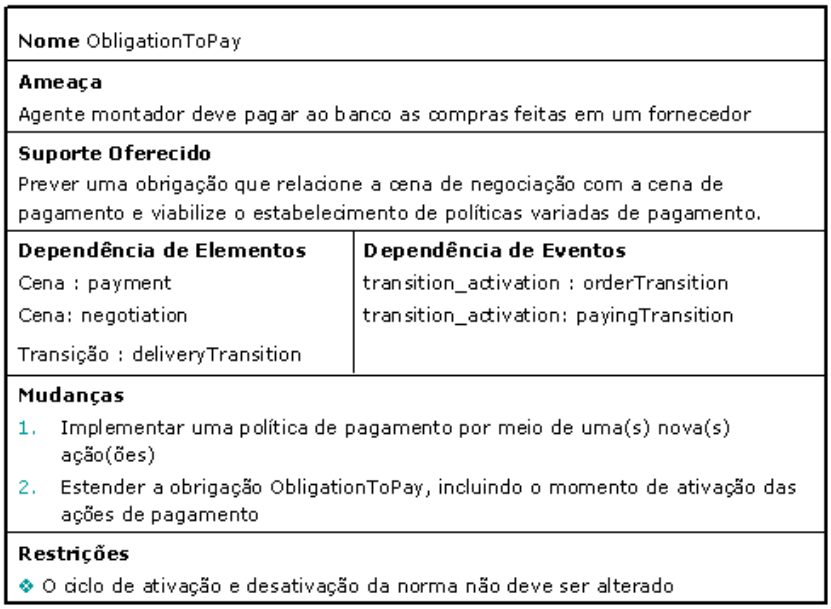
| **Nome** Framework de Governança de Cadeias de Suprimento (TAC SCM) | **Versão** 1.1 |
|---|---|
| **Propósito** Regular as interações entre fornecedores, montadores e banco em uma cadeia de suprimentos idealizada na competição TAC SCM. O propósito desta solução é realizar as edições da competição a partir das leis de interação previstas nesta solução. | |
| **Design**<br>São definidos fluxos padrão para a negociação entre fornecedores e montadores e o processo de pagamento destas compras entre o banco e os montadores.<br><br>O núcleo do framework é composto de uma cena para a negociação, uma cena para pagamento, a definição dos passos de interação (transições), estados para controle da conversação e de mensagens trocadas pelos agentes, e a permissão sobre o encadeamento e correlação entre duas mensagens da conversação (RFQ e OFFER).<br><br>Os pontos de extensão do framework incluem a permissão associada a montadores para submeter requisições durante o período de um dia (número de requisições permitidas e a forma de sua contagem), as restrições para verificar a validade das datas, e o método de pagamento implementado por ações dentro da obrigação. | |
| **Pontos de flexibilização**<br>1. Obrigação : ObligationToPay<br>2. Permissão : AssemblerPermissionRFQ<br>3. Transição : rfqTransition | |
| **Como instanciar**<br>1. Crie um novo elemento Action extendendo a obrigação ObligationToPay para incluir a forma de pagamento prevista para esta edição<br><br>2. Crie elementos Action e Constraint extendendo a permissão AssemblerPermissionRFQ para incluir a forma de cálculo do número de mensagens que um montador pode enviar para seus fornecedores<br><br>3. Defina a classe que implementa a restrição que valida os atributos de uma mensagem RFQ, completando a transição rfqTransition | |

**Figura 10. Guia de referência do G-Framework TAC SCM**

## 5. Trabalhos Relacionados

[Ao e Minsky 2003] propõem uma abordagem que aprimora o LGI (Law Governed Interaction) com o conceito de hierarquia de políticas para apoiar que as diferentes políticas internas ou leis sejam formuladas uma independente da outra em busca de atingir um apoio a flexibilidade desta forma. A natureza dos elementos definidos em LGI é de diretivas primitivas do tipo *disconnected*, *reconnected*, *forward*, e *deliver*. O propósito da extensão é viabilizar a confidencialidade das leis desenvolvidas em organizações que não podem ter acesso a detalhes específicos de outras organizações. Não existe uma metodologia clara para a aplicação do mecanismo de customização proposto. As diferenças entre o trabalho proposto neste artigo e o LGI são a linguagem de mais alto nível de especificação de leis de interação, o propósito do suporte a extensibilidade e as abstrações em projeto e implementação para reuso.

[Singh 1998] propõe um serviço de govenança customizável, baseado em skele-



**Nome** ObligationToPay

**Ameaça**
Agente montador deve pagar ao banco as compras feitas em um fornecedor

**Suporte Oferecido**
Prever uma obrigação que relacione a cena de negociação com a cena de pagamento e viabilize o estabelecimento de políticas variadas de pagamento.

| **Dependência de Elementos** | **Dependência de Eventos** |
|---|---|
| Cena : payment | transition_activation : orderTransition |
| Cena: negotiation | transition_activation: payingTransition |
| Transição : deliveryTransition | |

**Mudanças**
1. Implementar uma política de pagamento por meio de uma(s) nova(s) ação(ões)
2. Estender a obrigação ObligationToPay, incluindo o momento de ativação das ações de pagamento

**Restrições**
- O ciclo de ativação e desativação da norma não deve ser alterado

**Figura 11. Documentação do ponto de extensão *ObligationToPay* do TAC SCM**

(A)

```
<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>
```

(B)

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

**Código 6. Instância do TAC SCM 2005 (A) *ObligationToPay* e (B) *checkDueDate***

tons. A técnica de skeletons é equivalente a máquina de estados finitas. Sua abordagem introduz formalmente idéias de escalonamento tradicionais em um ambiente no qual agentes autônomos sem que o projetista do mecanismo possua controle sobre as suas ações, ou conhecimento detalhado sobre o seu projeto. No entanto, o propósito da abordagem de [Singh 1998] que é a construção do SMA. Nossa preocupação está restrita ao monitoramento e ao cumprimento de leis de interação por parte dos agentes.

COSY [Haddadi 1995] é um trabalho pioneiro que propôs que protocolos de interação sejam uma agregação de protocolos primitivos. Cada protocolo primitivo pode ser representado como uma árvore, no qual cada nó corresponde a uma situação particular e transições corresponderiam a mensagens que um agente pode tanto receber quanto enviar, correspondendo às várias alternativas de interação. Tal abordagem pode ser vista como instrumento para especificar possíveis adaptações em leis de interação extensíveis em g-frameworks.

## 6. Conclusões e Trabalhos Futuros

Levando em consideração os princípios de reuso de leis de interações em SMAs, definimos a abordagem de g-frameworks que tem o objetivo de promover a flexibilidade e da reutilização de projetos e implementações de leis de interação em sistemas multiagentes abertos. A flexibilidade é obtida com a introdução de incrementos específicos que as diferentes instâncias em desenvolvimento, de modo a completar e adaptar as funcionalidades originais do g-framework. Para auxiliar a aplicabilidade desta técnica, propôs-se uma estrutura de documentação da solução para apoiar o entendimento da solução.

A abordagem de g-frameworks trouxe uma maior preocupação quanto a modularidade e separação de elementos de leis em XMLaw, e de uma forma geral para o desenvolvimento de mecanismos de governança de leis de interação. A identificação clara dos pontos de extensão em leis de interação (código e documentação) tornou a abordagem mais própria para a sistematização da reutilização de leis como forma de gerar e adaptar uma família de mecanismos de governança. Contribuímos assim para a engenharia de como é possível produzir e reutilizar leis de interação.

Dentre diversos trabalhos futuros e outros em andamento é possível destacar a proposição de scripts de instanciação de g-frameworks a partir dos guias de referências gerados. Este trabalho futuro segue a linha proposta em [Oliveira et al. 2007] onde uma linguagem de script chamada RDL foi proposta para aprimorar a sistematização do desenvolvimento de frameworks OO. Uma linguagem adaptada de RDL pode ser criada para orientar a geração de novos mecanismos de governança a partir de g-frameworks.

## Referências

Ao, X. e Minsky, N. (2003). Flexible regulation of distributed coalitions. In *Proc. of the 8th European Symposium on Research in Computer Security (LNCS 2808)*, pages 39–60. Springer.

Arunachalam, R., Sadeh, N., Eriksson, J., Finne, N., and Janson, S. (2004). *The supply chain management game for the trading agent competition 2004*. CMU-CS-04-107.

Batory, D., Cardone, R., and Smaragdakis, Y. (2000). Object-oriented frameworks and product-lines. In *Proc. of the 1st Software Product-Line Conference*.

Choren, R. and Lucena, C. (2005). The anote modeling language for agent-oriented specification. In *Software Engineering for Multi-Agent Systems III (LNCS 3390)*, pages 198–212. Springer.

Collins, J., Arunachalam, R., Sadeh, N., Eriksson, J., Finne, N., and Jason, S. (2004). *The supply chain management game for the 2005 trading agent competition*. CMU-ISRI-04-139.

Collins, J., Arunachalam, R., Sadeh, N., Eriksson, J., Finne, N., and Jason, S. (2006). *The supply chain management game for the 2007 trading agent competition*. CMU-ISRI-07-100.

Dignum, F. (2001). Agents, markets, institutions, and protocols. In *Agent Mediated Eletronic Commerce, The European AgentLink Perspective*, pages 98–114. Springer.

Esteva, M. (2003). *Eletronic Institutions: from specification to development*. PhD Thesis, LSI - UPC.

Fayad, M., Schmidt, D., and Johnson, R. (1999). *Building application frameworks : object-oriented foundations of framework design*. Wiley.

Froehlich, G., Hoover, H., Liu, L., and Sorenson, P. (1997). Hooking into object-oriented application frameworks. In *Proc. International Conference on Software Engineering*, pages 491–501.

Haddadi, A. (1995). Modelling interactions in cosy agent architecture. In *Communication and Cooperation in Agent Systems (LNCS 1056)*, pages 101–131. Springer.

Johnson, R. (1992). Documenting frameworks using patterns. In *Proc. of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 63–76.

Krasner, G. and Pope, S. (1988). A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *J. of Object Oriented Programming*, 1:26–49.

Menezes, P. (1997). *Linguagens Formais e Autômatos*. Editora Sagra.

Minsky, N. and Ungureanu, V. (2000). Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering Methodology*, 9(3):273–305.

Oliveira, T., Alencar, P., Lucena, C., and D.D.Cowan (2007). Rdl: A language for framework instantiation representation. In *Journal of Systems and Software*.

Paes, R., Carvalho, G., Lucena, C., Alencar, P., Almeida, H., and Silva, V. (2005). Specifying laws in open multi-agent systems. In *Agents, Norms and Institutions for Regulated Multi-agent Systems (ANIREM at AAMAS)*.

Pree, W. (1997). *Essential Framework Design Patterns*. Object Magazine.

Singh, M. (1998). Developing formal specifications to coordinate heterogeneous autonomous agents. In *Proc. of the 3rd International Conference on Multiagent Systems*, pages 261–268.