4
O Desenvolvimento de G-Frameworks

Framework de Governança de Sistemas Multiagentes Abertos, ou g-framework, é uma abordagem que apóia três preocupações principais do ciclo de geração e manutenção de leis de interação: requisitos, projeto e implementação. Estas preocupações estão relacionadas e são fundamentais para a produção de uma família de mecanismos de governança. Para cada preocupação, é proposta uma técnica que tem como objetivo estruturar e documentar os resultados previstos.

O objetivo principal deste capítulo é descrever como é possível sistematizar a forma como são desenvolvidas e mantidas leis de interação para uma família de mecanismos de governança de sistemas multiagentes abertos. Esta seção descreve em detalhes a aplicação de técnicas de engenharia de software adaptadas para este contexto e apresentadas no capítulo anterior.

O desenvolvimento de g-frameworks é muito similar a qualquer desenvolvimento de software reutilizável. Começa com uma extensa análise de domínio [Arango, 1993; Prieto-Diaz et al., 1994; Kang et al., 1993], onde entre outras coisas, exemplos de instâncias são coletados e analisados para se obter partes comuns e partes variáveis. Durante nossa experimentação, este mesmo processo foi o aplicado em dois estudos de caso: o g-framework de liquidação e custódia de títulos públicos (SELIC) e o g-framework de cadeias de suprimento (TAC SCM). O exemplo do g-framework TAC SCM é apresentado nesta seção para facilitar a explicação da aplicação das técnicas propostas, e o g-framework SELIC é apresentado no próximo capítulo como nosso estudo de caso.

Abaixo está descrito de forma sucinta o caminho lógico de desenvolvimento de g-frameworks. O objetivo desta descrição é ilustrar uma forma de uso conjunto das técnicas propostas e não a proposição de uma metodologia propriamente dita.

Primeiramente, um domínio de aplicação é estudado e sua variabilidade é documentada em conjunto com seus requisitos. Instrumentos para identificar a proximidade dos requisitos podem ser utilizados para classificar os requisitos parecidos em grupos comuns, assim a variabilidade pode ser mais facilmente

identificada. Se os requisitos estiverem expressos em linguagem natural, então técnicas de processamento de linguagem natural podem ser utilizadas.

De acordo com a nossa proposta, em tempo de design, estes requisitos orientarão a separação entre o núcleo das leis de interação, que segundo a análise feita anteriormente são informações comuns a um conjunto de aplicações ou que são estáveis e que devem variar pouco ao longo do ciclo de vida da solução. Por outro lado, requisitos também podem indicar os pontos de extensão que explicitam o lócus de aplicação de uma variação (instanciação de nova aplicação). Esta distinção deve ser claramente documentada.

Com isto feito, em tempo de implementação, é possível gerar uma solução quase completa, postergando somente as customizações para a fase de instanciação posterior. Em tempo de instanciação, o desenvolvimento da solução deve ser finalizado, complementando os pontos de extensão deixados em aberto. A documentação da solução anterior deve ser detalhada o suficiente para facilitar a instanciação do g-framework.

Uma iteração do modelo de desenvolvimento de g-frameworks pode ser vista adiante (Figura 24). Este modelo se baseia fortemente em casos de leis, pontos de extensão e na implementação das leis de interação utilizando XMLaw. Casos de leis têm origem a partir de ameaças identificadas através de uma análise de risco sobre os casos de uso do sistema. Ameaças apoiarão o processo de elicitação de requisitos de leis em casos de leis, que será basicamente um processo de mitigação do risco da ocorrência de ameaças. Com os casos de leis (Figura 24), já é possível amadurecer o projeto do g-framework a partir da documentação clara do racional, incluindo questões de projeto e da necessidade de flexibilidade por parte da solução. A partir do projeto resultante já é possível implementar a solução utilizando XMLaw.

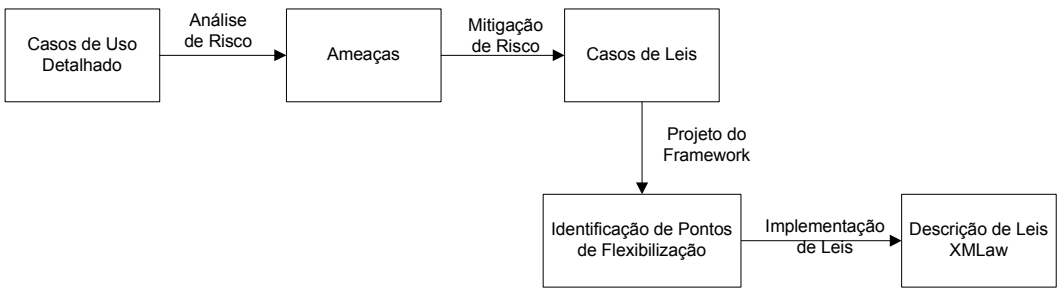

Figura 24 – Método Iterativo de Desenvolvimento: Foco nos Casos de Leis

As próximas subseções irão descrever passo a passo o processo de desenvolvimento de g-frameworks, dando ênfase na explicação de como as técnicas propostas podem ser efetivamente aplicadas.

4.1. Variabilidade do TAC SCM

O processo de reutilização de leis de interação é composto pela execução de diversas atividades básicas de reutilização tais como projeto, especialização e redefinição de elementos de leis. Embora estas atividades sejam extremamente simples, seu encadeamento pode levar a situações nada triviais uma vez que estas atividades podem ser feitas de forma não contínua e/ou por várias pessoas.

Uma característica importante de uma boa solução própria para reutilização é que ela proveja regras maduras em um domínio específico ao qual se aplicará [Fayad et al. 1999]. Neste sentido, ao longo da apresentação das técnicas, as regras do TAC SCM [Arunachalan et al. 2004; Collins et al. 2004; Collins et al. 2005] serão utilizadas como prova de conceito para o projeto e a implementação do g-framework de cadeias de suprimento. O TAC SCM foi escolhido como exemplo por ter mais de três edições que apresentaram variações nas regras de interação.

Nosso objetivo com o g-framework TAC SCM é provermos uma estrutura capaz de gerar um conjunto bem definido de edições que já ocorreram desta competição, e ainda fomentar com isto evoluções desta mesma natureza. Na implementação gerada em nosso estudo, cada instância do g-framework corresponderá ao mecanismo de governança de uma edição da competição. A razão para estas regras variarem está na natureza e na complexidade do problema abordado. As regras do TAC SCM são leis que regulam a interação entre os agentes participantes da negociação. Elas evoluíram a partir da observação do comportamento de agentes em edições recentes da competição e de suas conseqüências positivas e negativas para os participantes (e.g. regras de interação foram definidas para proteger agentes de participantes maliciosos).

Analisando a evolução dos requisitos entre as edições do TAC SCM, é possível observar evidências de que o protocolo de interação possui um núcleo estável. Nesta mesma análise, é possível ainda perceber a existência de pontos de extensão que poderiam ser customizados para atender detalhes das diferentes edições da cadeia de suprimentos. Associadas ao núcleo do protocolo de

negociação, existiam variações no TAC SCM quanto à restrição de validação de atributos de mensagens RFQ existente, à permissão de envio de mensagens RFQ, e à obrigação de pagamento da negociação de peças para a produção de PCs. Para tornar este cenário mais concreto, no próxima seção detalharemos o projeto e as modificações necessárias em transições, normas, ações e restrições para atingir a solução do g-framework de cadeia de suprimentos TAC SCM.

4.2. Requisitos de Governança de Sistemas Multiagentes Abertos

Requisitos são identificados em um g-framework utilizando casos de uso e uma versão simplificada de casos de lei. Casos de uso descrevem requisitos de forma tradicional. A ênfase de nossa abordagem é amadurecer as leis de interação necessárias para atender estes requisitos, mitigando eventuais ameaças identificadas. A forma como descrevemos e derivamos leis de interação são casos de leis. O propósito desta técnica aplicada em nosso contexto é documentar a linha de raciocínio do analista da aplicação ao mapear as necessidades dos usuários ou do domínio em análise, considerando em nosso contexto a sua possibilidade de evolução ou variação dentro das aplicações que devem ser geradas por esta solução.

Abaixo descrevemos duas listas reduzidas de casos de uso (Tabela 9) e de ameaças (Tabela 10) identificadas na etapa de análise dos requisitos do domínio do g-framework TAC SCM. A partir dos documentos de caso de uso e da lista de ameaças é possível derivar os casos de leis que irão nortear a descrição de leis de interação.

Caso de Uso
Negociar Componentes Eletrônicos para PCs
Negociar a venda de PCs
Pagar a compra de componentes eletrônicos
Receber pagamento de PCs

Tabela 9 – Lista Parcial de Casos de Uso do TAC SCM

Ameaça
Surgir uma nova modalidade de negociação de componentes eletrônicos para PCs
Agente montador enviar mensagem fora da data permitida em pedido de cotações
Agente montador enviar número muito grande de mensagens rfq para fornecedor
Agente montador não pagar ao banco as compras feitas em um fornecedor

Tabela 10 – Lista Parcial de Ameaças ao Ambiente do TAC SCM

Abaixo (Figura 25, Figura 26, Figura 27, Figura 28) é exemplificada a aplicação da técnica de casos de leis na documentação de requisitos do TAC SCM. Neste exemplo, nenhuma técnica de comparação de similaridade foi utilizada. Esta situação será mais bem explorada no próximo capítulo.

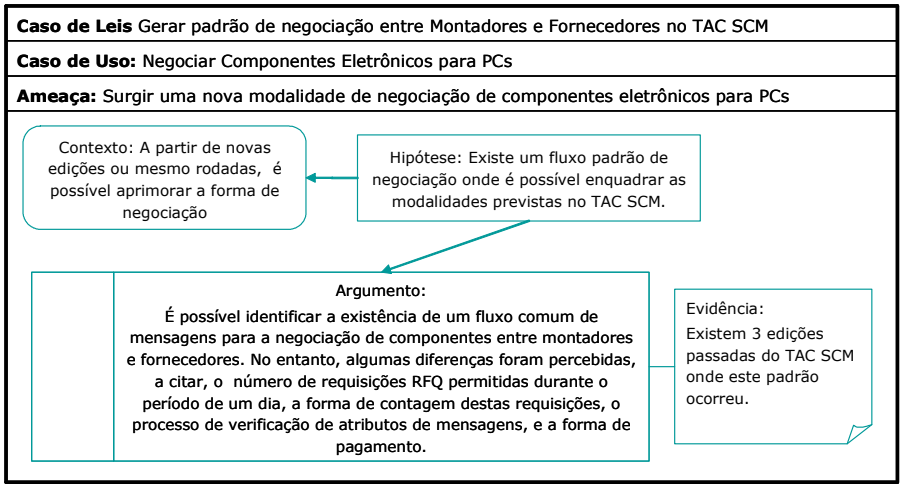

Figura 25 – Caso de Lei : Padrão de Negociação

A linha de raciocínio descrita em um Caso de Lei deve, quando oportuno, abordar a variabilidade dos requisitos e de suas leis. Uma análise quanto à possibilidade ou necessidade de variação de leis de interação pode ser sucintamente descrita, incluindo nesta racionalização um questionamento quanto à variabilidade daquele requisito. Um exemplo deste tipo de preocupação pode ser visto na Figura 26.

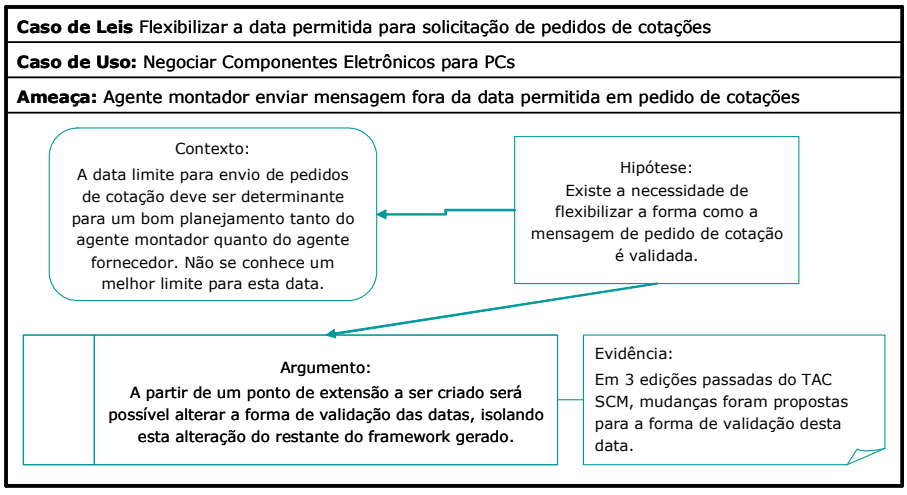

Figura 26 – Caso de Lei : Solicitação de Data Limite do Pedido de Cotações

O detalhamento de requisitos de leis continua sendo baseado em uma extensa análise de riscos e ameaças presentes no sistema aberto. No entanto, somado a isto, a análise de domínio da solução deve considerar a similaridade e

fatos relevantes que possam orientar o processo de geração de uma solução aderente à reutilização. Exemplo disto pode ser visto na Figura 27.

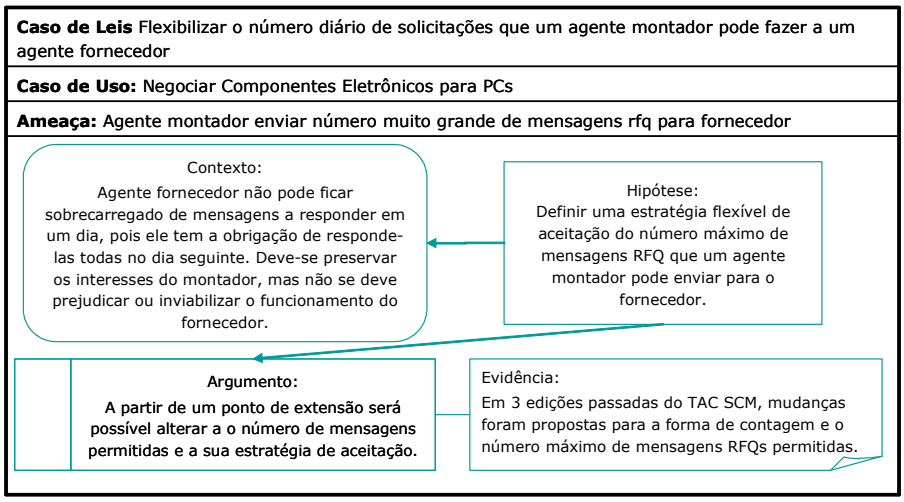

Figura 27 – Caso de Lei: Número de Solicitações

Mais do que só um instrumento de documentação das funcionalidades e controles esperados de um mecanismo de governança, um Caso de Lei deve ser utilizado como um instrumento para refletir ou induzir a reutilização dos elementos de leis que serão gerados *a posteriori*. Esta indução se dará pela documentação da linha de raciocínio associado à variação, que está mapeada na estrutura proposta nesta abordagem. Exemplo disto pode ser visto na Figura 28.

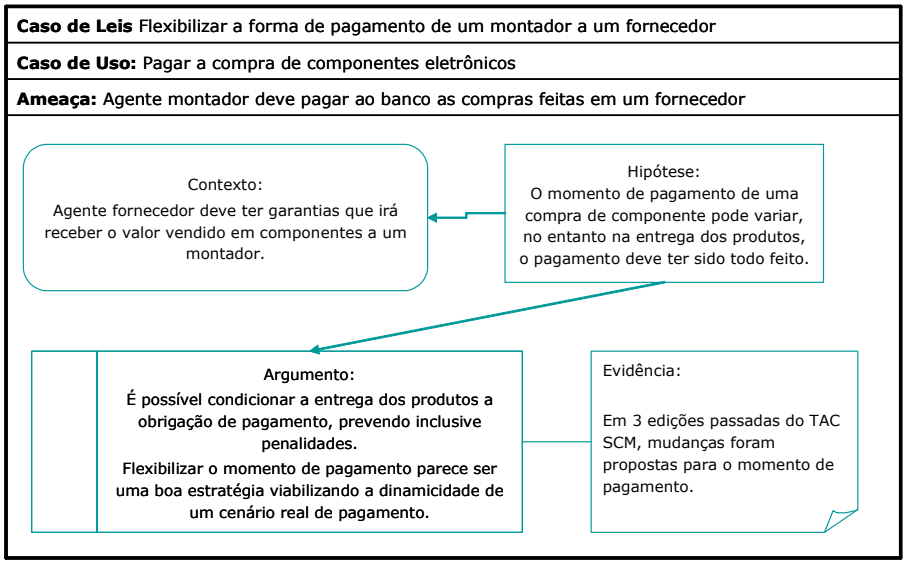

Figura 28 – Caso de Lei: Forma de Pagamento

Após elicitar os requisitos, identificar as ameaças e derivar os casos de leis com ênfase em uma análise de domínio, é possível caminhar para a etapa de projeto, onde o núcleo e os pontos de extensão de um g-framework serão identificados.

4.3. Projeto de G-Frameworks

O projeto de g-frameworks (Figura 29) 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. Para isto, são utilizados tanto os requisitos (casos de uso), quanto os casos de lei existentes. 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 extensão de interação.

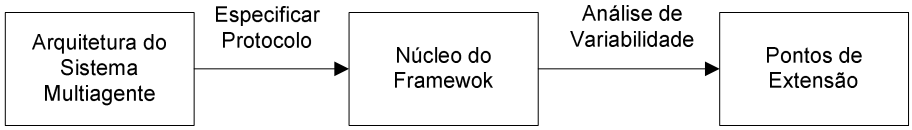

Figura 29 – Projeto de um G-Framework em SMAs Abertos

Neste processo será necessário classificar os requisitos/casos de leis como tendo características fixas (estáveis) ou flexíveis. Segundo a análise feita, caso não seja 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 deste 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, onde a partir de um maior detalhamento podem ser geradas aplicações com especificidades próprias.

Note-se que este é um trabalho complementar ao já executado na fase anterior, onde a variabilidade dos requisitos havia sido analisada e mapeada em parte para casos de lei. No entanto, agora a ênfase é em determinar claramente onde a solução não deve variar i.e., deve ser comum a todas as instâncias (núcleo da solução), e onde ela poderá ser flexibilizada (pontos de extensão da interação).

Abaixo iremos explicar a técnica de projeto com mais detalhes com o exemplo da geração da solução em projeto do g-framework de cadeia de suprimentos (TAC SCM). Utilizaremos o exemplo para tornar mais fácil o entendimento desta técnica.

4.3.1.
Descrição do Projeto do TAC SCM

No TAC SCM, agentes de montagem precisam negociar com fornecedores para a compra de componentes eletrônicos para a produção de PCs. Um agente banco é utilizado para monitorar o progresso financeiro dos agentes. Na arquitetura real do TAC SCM, existe um servidor que simula o comportamento de fornecedores, consumidores e fábricas. Convertemos parte destes componentes de simulação em agentes externos que foram desenvolvidos. Continuamos a ter o servidor TAC, mas sua principal função passa a ser o monitoramento e a análise da conformidade do comportamento dos agentes em relação às leis que são válidas para aquela edição.

Analisando a variabilidade das interações na negociação entre fornecedores e montadores e a estrutura de pagamento desta negociação nas edições do TAC SCM (Figura 25, Figura 26, Figura 27, Figura 28), propusemos um projeto para a geração do g-framework. Em resumo, a ser detalhado mais a frente, o núcleo do g-framework é composto por uma cena para a negociação, uma cena para pagamento, a definição dos estados 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.

O primeiro objetivo da fase de projeto é identificar os papéis de agentes que irão participar das interações governadas pelo framework. Espera-se que neste momento seja gerado um diagrama de classes de agentes em ANote [Choren & Lucena, 2005] ou um diagrama similar que cumpra este papel. Além disto, diagramas de máquina de estado podem ser úteis para a compreensão do encadeamento dos elementos de leis.

4.3.2.
Arquitetura 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 30) descreve os papéis, seus relacionamentos e suas cardinalidades.

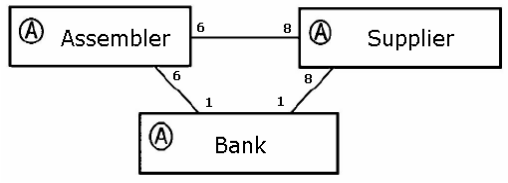

Figura 30 – Papéis, Relacionamentos e Cardinalidade no TAC SCM

O segundo objetivo da fase de projeto é analisar a estrutura da interação entre os agentes e determinar o núcleo da conversação, correspondente ao núcleo do g-framework. Este núcleo incluirá a organização da conversação em cenas, protocolos e elementos de leis comuns a todas as instâncias do g-framework. Espera-se que com a definição do núcleo do g-framework os protocolos de interação sejam representados por diagramas de interação em ANote [Choren & Lucena, 2005], ou outro diagrama similar.

4.3.3. Núcleo 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 31, Figura 32). A negociação entre montadores e fornecedores se dá através de quatro mensagens (Figura 33), cinco estados, e seis transições (Figura 34). 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).

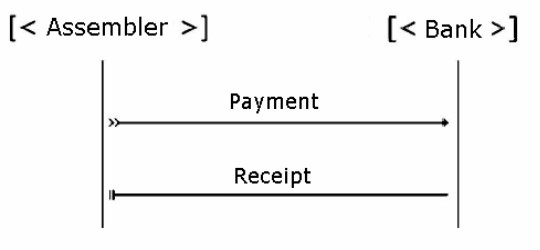

Figura 31 – Diagrama de Interação de Pagamento

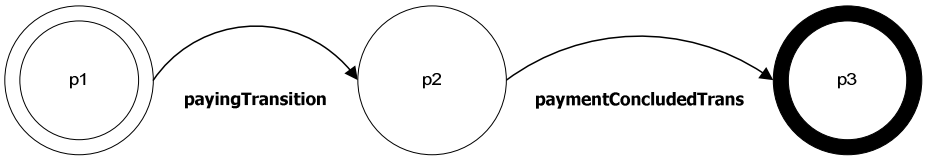

Figura 32 – Máquina de Estados da Cena Pagamento

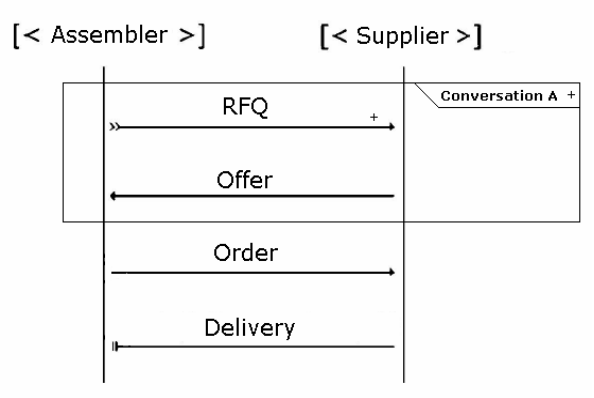

Figura 33 – Diagrama de Interação de Negociação

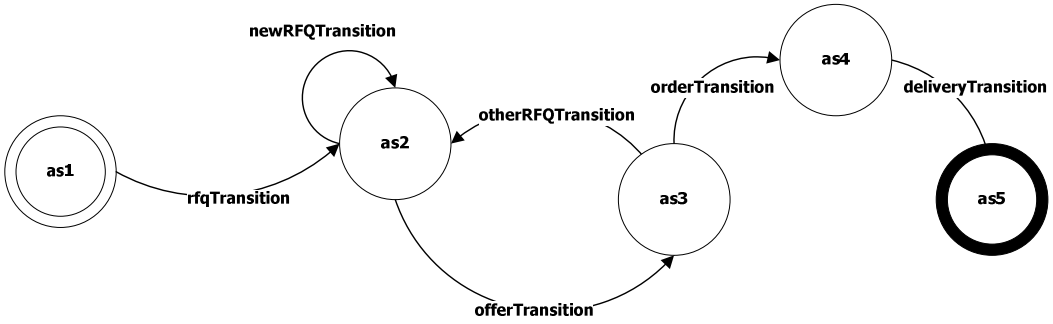

Figura 34 – Máquina de Estados de Cena de Negociação

A identificação de uma permissão é descrita como forma de ilustrar outra natureza de informação incluída no núcleo do g-framework Esta 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). Abaixo, descrevemos sucintamente o requisito de acordo

com [Arunachalan et al. 2004; Collins et al. 2004; Collins et al. 2005] documentado na forma de cenário.

"No dia seguinte à chegada de uma mensagem de "request for quotation" (RFQ), o fornecedor envia uma resposta para cada RFQ, contendo o preço, quantidade oferecida e data limite para entrega. O fornecedor pode ainda responder com até duas respostas com propostas alternativas que relaxam uma das restrições solicitadas: quantidade ou data limite para entrega. Uma oferta parcial é gerada com a quantidade de itens menor que a solicitada; ou uma oferta com o menor prazo de entrega com a quantidade solicitada é oferecida. Ofertas são recebidas no dia seguinte a submissão de RFQs e o montador precisa escolher se aceitará a oferta. No caso do montador aceitar as duas ofertas (com itens reduzidos e data postergada), somente a primeira mensagem recebida será aceita, a outra mensagem não deve ser considerada."

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.

4.3.4. Pontos de Extensão do TAC SCM

Pontos de extensão são decisões de projeto, baseados na identificação clara das possibilidades de variação de uma solução. Em tempo de instanciação, cada ponto de extensão formulado de forma abstrata será materializado, gerando a aplicação customizada. Abaixo, descrevemos três pontos de extensão do TAC SCM.

O ponto de extensão mais simples analisado no TAC SCM é a validação necessária, porém variável, dos atributos esperados em uma mensagem do tipo RFQ. A solução para este requisito é associar uma restrição à 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.

Ainda de acordo com as especificações das ultimas edições do TAC SCM [Arunachalan et al. 2004; Collins et al. 2004; Collins et al. 2005], a cada dia, cada agente pode enviar um número máximo de mensagens do tipo RFQs para um fornecedor. No entanto, o número preciso de RFQs variou segundo as regras das edições da competição, e portanto, este detalhe deve ser postergado para o

momento da instanciação. Uma permissão chamada de AssemblerPermissionRFQ foi criada abstratamente para encapsular este ponto de extensão.

Outro ponto de extensão foi especificado para definir o relacionamento entre ordens e ofertas dentro de um protocolo de negociação. De acordo com [Sadeh et al., 2003], 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.

Após a arquitetura, o núcleo e os pontos de extensão estarem identificados é possível prosseguir para a fase de implementação dos g-frameworks. Esta implementação se resume a uma descrição da solução a partir da utilização dos elementos de leis presentes em XMLaw.

4.4. Implementação de G-Frameworks

A implementação de g-frameworks está relacionada à utilização de operadores de refinamento: *abstract, completes* e *extends*. Como visto, 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) customizar 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*.

Como foi visto na subseção 4.3.1, o projeto de leis de interação está sujeito a alterações específicas que são feitas a partir da identificação de pontos de extensão. Um ponto de extensão é uma abstração utilizada para representar o conhecimento sobre o local onde as modificações ou melhorias em leis devem ser feitas. Estes pontos já foram identificados na fase anterior. Na prática, é útil para permitir a inclusão de novos elementos de leis como normas, restrições e ações

em uma estrutura já pré-definida. Abaixo iremos descrever a implementação do exemplo do g-framework de cadeia de suprimentos TAC SCM.

4.4.1. Implementação do TAC SCM

A implementação da solução é feita a partir do projeto já estipulado. São duas atividades de desenvolvimento: o desenvolvimento dos agentes de software e a descrição das leis de interação utilizando XMLaw. A primeira atividade está fora do escopo desta tese. Apresentaremos abaixo, a descrição das leis da interação do núcleo e dos pontos de extensão do g-framework para cadeia de suprimento.

4.4.1.1. Implementação do Núcleo do G-Framework TAC SCM

Conforme explicitado anteriormente as leis de interação desta aplicação estão organizadas em duas cenas. O Código 12 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. O Código 13 descreve o processo de pagamento. Decidimos não especificar nenhum tempo limite para esta cena (representado pelo tempo "infinity" no atributo time-to-live).

```
<Scene id="negotiation" time-to-live="3300000">
   <Creators>
      <Creator role="assembler"/></Creators>
   <Entrance>
      <Participant role="assembler" limit="6"/>
      <Participant role="supplier" limit="8"/></Entrance>
</Scene>
```

Código 12. Estrutura da Cena de Negociação

```
<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 13. Estrutura da Cena de Pagamento

Abaixo é possível observar detalhes da implementação da cena de pagamento entre o montador para o banco (Código 15 e Código 14), e no Código 16 e Código 17 são descritos detalhes da cena de negociação entre montadores e fornecedores utilizando XMLaw.

```
<Protocol>
   <States>
    <State id="p1" type="initial"/>
    <State id="p2" type="execution"/>
    <State id="p3" type="success"/></States>
   <Transitions>
     <Transition id="payingTransition"
       from="p1" to="p2" message-ref="payment"/>
     <Transition id="paymentConcludedTrans"
       from="p2" to="p3" message-ref="receipt"/></Transitions>
</Protocol>
```

Código 14. Descrição do Protocolo de Interação de Pagamento

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

Código 15. Descrição das Mensagens de Pagamento

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

Código 16. Descrição das Mensagens do Protocolo de Interação de Negociação

```
<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 17. Descrição do Protocolo de Interação da Negociação

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 18 e Código 19. 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 dia após o recebimento do RFQ. Esta permissão só é válida se ambas as restrições são verdadeiras. Abaixo se ilustra a offerTransition (Código 18) e descreve-se a permissão RestrictOfferValues e o seu código XMLaw (Código 19).

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

Código 18. Especificação Geral de uma Transição

```
<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 19. Especificação Geral de uma Norma

XMLaw inclui a noção de contexto. Elementos em um mesmo contexto utilizam uma memória local para compartilhar informação, i.e., colocar, obter e atualizar qualquer valor que seja importante para ele ou para outros elementos de lei. Código 19 ilustra o uso do contexto, onde a ação keepRFQInfo é um serviço responsável por preservar a informação de uma mensagem RFQ para ser utilizada *a posteriori* pelas restrições checkAttributes e checkDates.

A partir da implementação do núcleo da solução é possível descrever e incluir os pontos de extensão previstos na solução.

4.4.1.2. Implementação dos Pontos de Extensão do G-Framework

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 20). Neste exemplo de ponto de extensão, foi necessário manter o atributo class da restrição checkDueDate não especificado, i.e., a classe que implementa esta restrição só será conhecida na instanciação do g-framework.

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

Código 20. Ponto de Extensão checkDueDate

A permissão AssemblerPermissionRFQ foi criada na forma abstrata para encapsular outro ponto de extensão (Código 21); alguns elementos de lei foram deixados em aberto nesta permissão e guiarão a especialização de uma instância (edição de jogo) a partir do g-framework gerado. Esta permissão regula o número máximo de RFQs que um montador pode submeter a um fornecedor. Para implementar este tipo de verificação, a restrição checkCounter é associada com a permissão AssemblerPermissionRFQ. Isto significa que a verificação não é verdadeira se a norma não estiver válida, mesmo quando ativa. A ação ZeroCounter é definida na permissão AssemblerPermissionRFQ e é acionada pelo clock-tick every day, zerando o valor do contador do número de requisições enviadas pelo montador durante este dia. A outra ação orderID é ativada por toda transição transitionRFQ e é utilizada para contar o número de RFQs enviados por um montador, atualizando um contador local.

Finalmente, o clock nextDay é utilizado para registrar o período de um dia, e esta marca é utilizada para zerar o contador de RFQs pela ação ZeroCounter. Nesta exposição não iremos descrever o clock nextDay. Tanto a restrição checkCounter quanto a ação orderID não foram completamente definidas, sendo *a posteriori* necessária a sua materialização.

```
<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="norm.actions.ZeroCounter">
         <Element ref="nextDay" event-type="clock_tick"/>
      </Action>
      <Action id="orderID">
         <Element ref="rfqTransition"
                  event-type="transition_activation"/>
       </Action></Actions>
</Permission>
```

Código 21. Ponto de Extensão AssemblerPermissionRFQ

Outro ponto de extensão especificado foi o ObligationtoPay (Código 24, Código 25, Código 22 e Código 23). Esta obrigação define o relacionamento entre ordens e ofertas dentro de um protocolo de negociação. Esta obrigação 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 25). O montador pode somente entrar em uma cena de pagamento se ele tiver a obrigação de pagar pelo produto (Código 22). Um montador não pode entrar em outra negociação se suas obrigações não tiverem sido cumpridas (Código 23). Optou-se que o processo de pagamento da compra fosse efetuado automaticamente pelo mediador. Os agentes não precisam fazer nada, pois as ações enviam as mensagens por eles.

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

Código 22. Cena e Norma de Pagamento

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

Código 23. Cena de Negociação e Norma de Pagamento

```
<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>
```

Código 24. Ponto de Extensão ObligationToPay

```
<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 25. Transições e Norma de Pagamento

Ao término de uma iteração para o desenvolvimento de um g-framework, é necessário documentar o resultado de forma que ele possa ser reutilizado de fato. Abaixo exemplificaremos o uso da técnica de documentação proposta para g-frameworks.

4.5. Documentação do G-Framework

Uma das tarefas mais importantes e difíceis de ser realizada durante o desenvolvimento de software é a sua documentação. Produzir uma documentação que possibilite, dentre outras coisas, a substituição do desenvolvedor original para uma futura manutenção, requer a capacidade de catalogar e estruturar os dados necessários para a compreensão do design/código em questão. Como visto no capítulo anterior, um g-framework deve ter uma descrição clara referente a decisões de projeto. Estas decisões incluem o g-framework (solução como um todo), seus pontos de extensão, e os requisitos que derivaram estas decisões de projeto.

Adiante apresentamos o resultado da documentação do exemplo TAC SCM. O guia de referência do g-framework de cadeias de suprimento (TAC SCM) está ilustrado abaixo (Figura 35). As documentações dos três pontos de extensão previstos para o TAC SCM estão descritas nas Figura 36, Figura 37 e Figura 38.

Após uma documentação adequada da solução é possível falarmos em reutilização de g-frameworks. Na próxima subseção descreveremos a criação de duas instâncias a partir da solução de cadeia de suprimentos TAC SCM.

Nome G-Framework 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 São definidos fluxos padrão para a negociação entre fornecedores e montadores e o processo de pagamento destas compras entre o banco e os montadores. 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). 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 extensão 1. Restrição: checkDueDate 2. Permissão : AssemblerPermissionRFQ 3. Obrigação : ObligationToPay	
Como instanciar 1. Crie novos elementos Action estendendo a obrigação ObligationToPay para incluir a forma de pagamento prevista para esta edição 2. Crie elementos Action e Constraint estendendo a permissão AssemblerPermissionRFQ para incluir a forma de cálculo e a restrição sobre o número de mensagens que um montador pode enviar para seus fornecedores 3. Defina a classe que implementa a restrição que valida os atributos de uma mensagem RFQ, completando a transição rfqTransition	

Figura 35 – Guia de Referência do G-Framework TAC SCM

Nome checkDueDate	
Ameaça Agente montador enviar uma mensagem fora da data permitida para pedido de cotações	
Suporte Oferecido Prever na transição que é ativada a partir de uma mensagem RFQ, a existência de uma restrição, não especificada, que verifique a validade do atributo data da mensagem enviada.	
Dependência de Elementos Transição : rfqTransition	**Dependência de Eventos** message_arrival : rfq
Mudanças 1. Completar a transição transitionRFQ com a classe que implementa a restrição de validação de atributos de mensagem	
Restrições ❖ Uma classe para a restrição checkDueDate deve ser definida	

Figura 36 – Documentação do Ponto de Extensão checkDueDate

Nome AssemblerPermissionRFQ	
Ameaça Agente montador enviar um número muito grande de mensagens rfq para um fornecedor	
Suporte Oferecido Prever uma permissão que regule o número máximo de mensagens rfq enviadas pelo montador ao fornecedor.	
Dependência de Elementos Clock : nextDay	**Dependência de Eventos** scene_creation : negotiation transition_activation: rfqTransition

Mudanças
1. Completar a permissão AssemblerPermissionRFQ
2. Implementar checkCounter
  1. Desenvolver uma restrição que controle o limite de mensagens que podem ser submetidas
  2. Preencher obrigatoriamente a classe que implementa a restrição de checkCounter que controla o limite de mensagens que um montador pode enviar
3. Implementar orderId
  1. Desenvolver uma ação que mantenha atualizado o status do número de mensagens enviados
  2. Preencher obrigatoriamente a classe que implementa a ação de orderId que mantém o estado sobre o número de mensagens que um montador já enviou enviar	

Restrições ❖ O contador será zerado a cada dia pela ação ZeroCounter (id = permissionRenew) acionada pelo clock nextDay.	

Figura 37 – Documentação do Ponto de Extensão AssemblerPermissionRFQ

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 Cena : payment Cena: negotiation Transição : deliveryTransition	**Dependência de Eventos** transition_activation : orderTransition transition_activation: payingTransition
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 38 – Documentação do Ponto de Extensão ObligationToPay

4.6. Instanciação do G-Framework TAC SCM

O processo de reutilização e customização de g-frameworks é comumente chamado de instanciação. É durante este processo que os pontos de extensão existentes no g-framework serão preenchidos para se obter a aplicação final. O que pode ser observado durante a instanciação é que o processo tradicional de desenvolvimento de um mecanismo de governança deve ser alterado de modo a incorporar a verificação da aderência do g-framework aos requisitos elicitados para a aplicação.

Este processo começa de forma similar ao desenvolvimento de aplicações clássicas com uma fase de requisitos, no qual os requisitos funcionais e não

funcionais são coletados e expressos utilizando casos de leis. Em seguida o domínio da aplicação é investigado para se verificar se o g-framework em questão pode ser apresentado como solução parcial para o problema. Uma vez escolhido o g-framework, inicia-se o processo de instanciação que tem como objetivo integrar/adaptar o modelo da aplicação ao modelo presente no g-framework. Em seguida temos os passos tradicionais de codificação em XMLaw. Todos os pontos de extensão devem ser preenchidos para que se gere um mecanismo de governança completo.

Abaixo, descreveremos dois exemplos de instâncias do g-framework de cadeia de suprimento TAC SCM. Esta instanciação será explicada em função dos pontos de extensão e da documentação que foram identificados e detalhados anteriormente.

4.6.1. Edição 2004 do TAC SCM

Na edição de 2004 do TAC SCM [Arunachalan et al. 2004], o fornecedor deve receber o pagamento do montador pela venda dos componentes após a entrega dos componentes. Isto quer dizer que, ao final do processo, o custo total da venda será pago pelo montador ao banco. Conforme explicado anteriormente, a política de pagamento foi implementada de forma automática como uma ação de XMLaw, implementada por um componente Java **actions.SupplierPayment100**, onde o mediador obrigava o agente a pagar o débito ao final da negociação. A obrigação ObligationToPay foi estendida para incluir esta ação (Código 26).

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

Código 26. Instância do TAC SCM 2004 : Obligation

Ainda nesta edição da competição [Arunachalan et al. 2004], a cada dia cada agente podia enviar até 10 mensagens do tipo RFQ (request for quote) para cada fornecedor, solicitando a cotação de um componente. Desenvolvemos um componente restrição chamado CounterLimit para garantir que o número de mensagens RFQs enviadas por um agente fosse inferior ao limite de 10. Ainda desenvolvemos um componente ação RFQCounter que incrementa a memória deste contexto, atualizando um atributo quando novas mensagens do tipo RFQ

fossem recebidas. Completamos a permissão AssemblerPermissionRFQ com os elementos criados, associando o componente CounterLimit a restrição checkCounter, e associando o componente RFQCounter a ação orderId (Código 27).

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

Código 27. Instância do TAC SCM 2004: Permission

Para o ponto de extensão checkDueDate, foi criado um componente restrição chamado ValidDate2004 que impede que uma RFQ tenha atributo DueDate (data limite) maior que a data final para o fim da negociação. Completamos a transição rfqTransition concluindo a materialização deste item (Código 28).

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

Código 28. Instância do TAC SCM 2004: checkDueDate

4.6.2. Edição 2005 do TAC SCM

Na edição de 2005 do TAC SCM, fornecedores devem receber um pagamento imediato de parte do valor da compra [Collins et al. 2004]. O restante deste valor é pago assim que o pedido é despachado. Em 2005, o pagamento antecipado foi de 10% do valor total. 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 29).

```
<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>
```

Código 29. Instância do TAC SCM 2005: Obligation

Quanto ao número de mensagens enviadas, os limites foram alterados para até cinco mensagens por produto por fornecedor. Se cada fornecedor tinha obrigatoriamente dois produtos, o número total de mensagens do tipo RFQs que poderiam ser enviadas era de 10. Desenvolvemos um componente restrição chamado CounterLimit2005 para considerar um atributo específico por tipo de componente provido por um fornecedor. Ainda desenvolvemos um componente ação RFQCounter2005 para contar o número de mensagens segundo o tipo de componente. Completamos a permissão AssemblerPermissionRFQ com os elementos criados, associando o CounterLimit2005 a restrição checkCounter, e associando o componente RFQCounter2005 a ação orderId (Código 30).

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

Código 30. Instância do TAC SCM 2005: Permission

Para o ponto de extensão checkDueDate, foi criado um componente restrição chamado ValidDate2005 que valida que uma RFQ tenha atributo DueDate (data limite) que não pode ultrapassar a data final de negociação e não pode ser superior a dois dias no futuro. Completamos a transição rfqTransition concluindo a extensão deste item (Código 31).

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

Código 31. Instância do TAC SCM 2005: checkDueDate

4.7. Conclusão

Neste capítulo descrevemos uma aplicação da tecnologia de g-frameworks em SMAs abertos governados por leis. Apresentamos um exemplo didático visando descrever como é possível sistematizar a forma como são desenvolvidas e mantidas leis de interação para uma família de mecanismos de governança de cadeias de suprimento baseada no TAC SCM. Esta seção descreveu em detalhes a aplicação de técnicas de engenharia de software propostas para g-frameworks.

Convertemos a aplicação de simulação e competição TAC SCM em um sistema aberto regulados por leis. Cadeias de suprimentos atualmente são uma realidade e movimentam trilhões de dólares. Apoiar o desenvolvimento deste tipo

de aplicação de forma mais confiável é um dos resultados da aplicação de técnicas de governança de sistemas multiagentes abertos. Finalmente, propusemos um g-framework baseado nas leis de interações de edições passadas da competição TAC SCM em prol de gerar com maior facilidade suas instâncias experimentais.

Este g-framework foi implementado em um ambiente controlado, composto por no mínimo três agentes (fornecedor, montador e banco), além do mediador. Apesar do desenvolvimento dos agentes não estar no escopo deste trabalho, ele foi feito de forma a validar o funcionamento correto das leis de interação e da solução proposta. Todas as leis descritas e os seus respectivos componentes foram implementados e executados segundo cenários de validação. Estes cenários foram utilizados para avaliar as condições de contorno e a efetividade das implementações definidas. Todos os contextos de variação descritos foram gerados, correspondendo às edições de 2004, 2005 e 2006 da competição.