3
Técnicas de Reutilização de Leis de Interação

Sistemas multiagentes abertos têm na sua dinâmica uma importante característica que é a possibilidade de entrada e saída de agentes dinamicamente. Isto pode acarretar um nível de variabilidade maior do que em sistemas tradicionais. Um suporte para o projeto e a implementação da evolução deste tipo de aplicativo torna-se necessário para uma boa engenharia destas aplicações. No caso do desenvolvimento de mecanismos de governança é preciso aplicar uma abordagem que facilite a extensão de leis de interação e como conseqüência o desenvolvimento deste tipo de aplicativo.

A definição de como um agente de software interage é muito importante para entender o comportamento de um SMA. A especificação de leis de interação é utilizada como guia para garantir que o comportamento obtido de um sistema multiagente é próximo do esperado. Muitas vezes estas leis não são completamente compreendidas no início do ciclo de vida destes sistemas, ou então, não são aplicadas de forma a facilitar a reutilização dada à complexidade de suas regras. Outro detalhe importante é que não existe suporte explícito nos mecanismos de governança atuais [Esteva 2003; Jones & Sergot, 1993; Kollingbaum & Norman, 2003; Martin et al., 1999; Mineau, 2003; Minsky & Ungureanu, 2000; Schumacher et Ossowski, 2005] para a alteração ou extensão das leis. Pouco suporte a extensibilidade acabará tendo um impacto negativo na manutenibilidade de mecanismos de governança de sistemas multiagentes abertos.

Neste capítulo abordaremos o conjunto de técnicas desenvolvidas para o contexto de reutilização de leis de interação. Além desta apresentação, buscaremos comparar nossas propostas com as técnicas em que elas se baseiam. O primeiro tópico a amadurecer é o entendimento de variabilidade de interações em sistemas multiagentes abertos, onde discutiremos que natureza de alterações um elemento de lei pode sofrer e como esta evolução ocorre. A partir do conceito de variabilidade de elementos de leis foram propostos operadores de refinamento em XMLaw e também pontos de extensão em leis de interação. A abstração de pontos de extensão é fundamental para estruturarmos o conceito de reutilização de forma

produtiva em tempo de projeto de leis de interação. A partir de uma adaptação dos conceitos de framework orientados a objetos [Batory et al. 2000; Fayad et al. 1999; Pree 1997], este capítulo descreve a técnica de framework de governança [Carvalho et al. 2007], re-batizada de **g-framework**. Esta técnica será introduzida e explicada neste capítulo, porém somente será explorada nos próximos capítulos com exemplos mais detalhados.

O objetivo da próxima seção é descrever a noção de variabilidade em leis de interação. Para isto, o conceito de variabilidade e o seu propósito serão discutidos de forma genérica. Depois verificaremos que existem níveis de abstração onde a variabilidade pode ser explorada (design e implementação). Por fim, discutiremos a variabilidade dos elementos de leis de interação, detalhando suas características pertinentes.

3.1. Variabilidade em Leis de Interação

A variabilidade em software é a característica de um sistema ou artefato de ser alterado, customizado ou configurado para uso em um contexto específico. Quanto maior a variabilidade de um software, menos restrito é o seu domínio de aplicação. Um software flexível tende a ser aplicável em um número maior de contextos, isto é, o software é mais reutilizável [Bachmann & Bass, 2001]. No entanto a variabilidade do software e suas conseqüências são dependentes da natureza das abstrações utilizadas para o desenvolvimento das aplicações.

Em sistemas multiagentes abertos regulados por leis, a forma como os agentes interagem pode variar ao longo do tempo. Cada interação em um sistema aberto é um ponto de variação em potencial. Protocolos de interação especificam as características previstas e projetadas para as interações de agentes distribuídos [Jennings & Wooldrige, 1998]. A especificação de protocolos de interação pode ser flexível o suficiente para permitir a inclusão de novas restrições ou regras que definirão o comportamento desejado para o sistema multiagente aberto. Com flexibilidade obtém-se uma maior reutilização de uma lei de interação, pois é possível aplicar esta lei em vários contextos. Quando variações são definidas sobre interações, elas especificam alternativas aos comportamentos que agentes poderão ter ao longo de sua execução.

A variabilidade em leis implica em protocolos de interação customizáveis para domínios específicos. Independente desta variação, sempre deve existir um

esqueleto ou um núcleo estruturante da solução. Este núcleo deve definir a base da solução e as características que limitam a replicabilidade desta solução para outros domínios de solução.

Existem dois níveis de detalhe em que é possível analisar e realizar a variabilidade: em tempo de design ou em tempo de implementação. Com a proposta de g-frameworks pretende-se dar apoio a estes dois níveis. A variabilidade em tempo de design deve ser entendida como a identificação da necessidade de variação, sem muitos detalhes, seguida da definição do lugar onde esta flexibilidade é esperada no projeto da solução. Em uma lei de interação, este tipo de variabilidade será representado por pontos de extensão de interação.

A variabilidade em tempo de implementação em XMLaw requer uma análise mais detalhada do impacto da inclusão ou alteração de elementos de leis de interação, pois o nível de detalhes esperado aqui é maior. Recomenda-se que esta variação seja projetada, isto é, em tempo de projeto, o elemento de lei que está sendo variado já havia sido identificado como um ponto de extensão de interação, isto é, o impacto de sua materialização foi estudado e plenamente analisado. A inclusão de um novo elemento de lei sem a atenção devida pode ter impactos indesejados para a solução. Em tempo de implementação, anotamos o código XMLaw com operadores de refinamento que serão explicados adiante.

Um elemento de lei pode ser definido como abstrato, para isto, basta identificá-lo desta forma. É preciso entender o alcance de eventuais alterações a estes elementos para que o projeto seja feito de forma adequada. Cada elemento do modelo conceitual de XMLaw possui uma natureza diferente, portanto a especialização de cada elemento se dará de forma diferente. Em tempo de implementação, é importante analisar com cuidado a estrutura e a conseqüência de eventuais alterações. Abaixo se discute a variabilidade dos elementos de leis de interação existentes em XMLaw para facilitar o entendimento sobre a possibilidade de customização e especialização. De forma geral, o ciclo de vida e o processo de criação e alteração de elementos serão explicados, para depois detalhar as características específicas de especializações dos elementos propriamente ditos.

CICLO DE VIDA - Alguns elementos em XMLaw (Clock, Norm e Action) podem ser ativados ou desativados por eventos. A ativação de um elemento indica quando o seu ciclo de vida terá início e a desativação indica quando o seu ciclo de

vida terá o seu fim. Estas informações são definidas em cada elemento em tempo de implementação e também podem ser previstas em projeto. Portanto, uma variabilidade sobre estes elementos é a possibilidade de incluir novos eventos que os ativam/desativam. Esta alteração terá impacto sobre o ciclo de vida do elemento e ainda pode causar alguma conseqüência sobre a cadeia de eventos descrita na lei de interação.

ALTERAÇÃO DE ELEMENTOS - A definição de um elemento semi-completo não requer muito trabalho. Para isto, obrigatoriamente, o identificador deste elemento já deve ter sido criado e espera-se que o escopo desejado para a sua alteração já seja conhecido em fase de projeto (ponto de extensão de interação). A partir daí, o detalhamento das demais informações pode ser postergado para futuro. No entanto, outros elementos podem fazer referência a eventos deste elemento abstrato.

CRIAÇÃO DE ELEMENTOS - A inclusão de um elemento novo pode ter impactos importantes na lei. Se nenhum outro elemento de lei referenciar este novo elemento, o impacto que ele terá sobre os demais é reduzido à entrada no final da cadeia de eventos já definida na lei. No entanto, é preciso analisar a cadeia de eventos para perceber se esta alteração não está modificando de forma não desejada os outros elementos. Adiante são descritas as características peculiares de cada elemento de lei quanto a sua alteração/criação.

CENA - Uma cena em XMLaw prevê dentre outras informações a definição de que participantes podem entrar, de que participantes podem criar a cena, o tempo de vida da cena e demais elementos já descritos no capítulo anterior. A variabilidade prevista para este elemento prevê a possibilidade de alteração destas definições. Quanto a alterações não recomendadas, uma cena pode ainda alterar um protocolo, um conjunto de normas, clocks, e ações. Devido à complexidade destes elementos, recomenda-se uma análise mais cuidadosa deles antes de sua alteração. Este elemento pode ser utilizado para encapsular um conjunto de alterações necessárias à customização de um protocolo de interação.

PROTOCOLO - A alteração de um elemento protocolo implica a possibilidade de alterar o estado inicial (se este não estiver definido) de uma conversação e do conjunto de mensagens, estados e suas respectivas transições. A criação de um novo protocolo deve ser feita a partir da definição de novos elementos ou da associação e substituição de elementos já existentes.

MENSAGEM - A alteração de atributos previstos em uma mensagem pode ter impacto no casamento das informações enviadas pelos agentes, modificando possivelmente o fluxo de observação esperado. Chama-se de redefinição a alteração de uma mensagem em que o evento message_arrival com identificador da mensagem já é referenciado por algum elemento de lei. Além disto, uma nova mensagem pode ser criada e adicionada ao protocolo de interação. A definição de uma mensagem sem associação do evento de sua chegada a algum observador não tem utilidade nenhuma sobre o ponto de vista de observação. É possível ainda pensar na inclusão de uma mensagem a partir de um conjunto de variações, por exemplo, inclusão de nova mensagem com id **nova**, alteração de referência à mensagem **nova** por uma transição já existente e definida como abstrata anteriormente.

TRANSIÇÃO - Uma transição relaciona um estado-origem a um estado-destino. A definição de uma nova transição implica a alteração do fluxo de evolução do protocolo de interação, desde que devidamente associada a dois estados já existentes ou em criação, e a associação a algum evento que venha a ser disparado em um fluxo de observação. É possível alterar os estados referenciados, as normas requeridas para a sua ativação, e as restrições que serão aplicadas aos eventos em uma transição. A alteração de uma transição também pode ter o impacto equivalente ao de uma nova definição de elemento.

ESTADO - A definição de um novo estado só faz sentido se associada a outras alterações, por exemplo, de transições. Como um estado define um contexto de observação esta alteração só faz sentido caso o protocolo de interação seja modificado com novas transições, ou transições que entram ou saem deste estado. Um estado define o seu tipo (inicial, execução, sucesso ou falha). Neste sentido, é importante observar que um protocolo só deve ter um estado inicial.

NORMA - A norma é um elemento que pode apresentar um importante papel na variabilidade de leis de interação. Este elemento pode muitas vezes ser utilizado para encapsular um conjunto de alterações necessárias à customização de um protocolo de interação. Como a norma é um contexto, a sua definição pode incluir a criação de novos elementos ações, clocks ou restrições. Sua definição sempre deve estar associada à existência de uma estrutura pré-definida, que deve incluir a definição do ciclo de vida da norma (ativação e desativação), a definição de quem receberá a norma no momento de ativação, e onde a norma está sendo

requerida (como ativada ou desativa) detalhando claramente o impacto de sua definição. Este elemento pode também influenciar a alteração do ciclo de acompanhamento e de possibilidades de evolução de um protocolo de interação de uma cena, pois uma norma habilita ou desabilita a evolução de um caminho dentro de um protocolo de interação.

RESTRIÇÃO - Uma restrição é um filtro de eventos que indica a possibilidade de ativação ou não de uma transição ou a validade de uma norma já ativada. Caso não tenha sido definida, a restrição pode ser criada em uma norma ou transição e tem impacto restrito a estes elementos. Caso este elemento já tenha sido definido estruturalmente (ele está localizado junto a uma norma ou transição) é necessário informar a classe Java que implementa o filtro condicional.

AÇÃO - Uma ação é um serviço executado pelo mediador ao longo do processo de observação. Uma ação pode ser definida no contexto de cena, norma ou lei. O impacto da definição deste elemento é restrito à definição da classe Java que implementa a ação, de que evento a ativará e ao uso não comum dos eventos de ativação de ações.

CLOCK - Um clock pode ser definido para especificar os atributos existentes como tipo (se periódico ou único), seu período de tempo e suas ativações e desativações. Um ciclo de notificações (clock_tick ou clock_timeout) é alterado ao se variar um clock. Estruturalmente, sua definição deve ser feita *a priori* ou em conjunto com outras definições, pois os eventos disparados pelo relógio serão utilizados por outros elementos que os precisam conhecer.

Abaixo iremos descrever a semântica de operadores de refinamento, um instrumento criado na linguagem XMLaw para viabilizar a definição de elementos abstratos e sua respectiva especialização. Comentaremos de forma sucinta como esta proposta evoluiu até o estágio atual.

3.1.1. Operadores de Refinamento

XMLaw já previa dois elementos que poderiam ser incluídos *a posteriori* na especificação de leis de interação: ações e restrições [Carvalho et al. 2005b]. Isto era possível, pois ambos os elementos dependiam de componentes implementados em Java que poderiam estar disponíveis somente em tempo de execução no mediador. Este fato permitia que sua implementação fosse alterada sem muito impacto para as leis de interação. Para que se entenda esta possibilidade explicaremos com um pouco mais de detalhes a semântica destes dois elementos.

Ações são elementos que incluem serviços implementados em componentes no sistema aberto. Serviços são funcionalidades específicas de domínio que podem ser disparadas enquanto o mediador monitora as interações. Um exemplo de ação é o envio pelo mediador de uma mensagem a um agente. Já restrições também são implementadas por componentes e funcionam como validadores ou filtros na utilização de determinado evento, por exemplo, uma mensagem com alguma informação inadequada pode ser filtrada impedindo a ativação de uma transição.

Abaixo estão dois exemplos de como os pontos de extensão eram abordados em um primeiro momento para depois serem refinados. O exemplo abaixo detalha leis aplicadas a vendas que são variáveis em determinadas épocas de um ano (e.g. política de descontos são oferecidas no verão e no inverno). A ação giveDiscount (Código 7) calcula e aplica o desconto e a restrição badClient (Código 8) restringe o desconto a um tipo específico de clientes. Este cenário permite que a partir de uma estratégia de venda, os componentes Java de restrições e ações sejam alterados, modificando e customizando as leis de interação propriamente ditas.

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

Código 7. Postergando a Definição de Ações

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

Código 8. Postergando a Definição de Restrições

Esta abordagem era bastante restrita e artesanal. Apesar da possibilidade de alteração da classe Java que implementava os elementos, não havia a possibilidade de alteração de outros elementos de leis também definidos. Além disto, não necessariamente ficara claro que aquele ponto poderia sofrer alteração ou se era um equívoco ao se preencher a lei. A ausência de uma identificação clara não apontava que aquele ponto de extensão havia sido projetado e pensado com este propósito. Melhoramos esta situação com a proposta de operadores de refinamento em XMLaw, à semelhança de operadores de herança em Java [Carvalho et al. 2006d]. Com os operadores de refinamento propõe-se que seja qualificado explicitamente um elemento abstrato (abstract). Além disto, propõe-se a existência de outros operadores que tornem estes elementos entidades completas (*completes* e *extends*). Abaixo descrevemos com mais detalhes estes operadores.

3.1.1.1. abstract

Elementos de lei podem ter atributos associados. O atributo **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.

Quando um elemento for definido como abstrato, algumas lacunas podem ser preenchidas *a posteriori* (Código 9). Esta é a realização do conceito de ponto de extensão mapeada para XMLaw, onde está definido claramente o contexto onde extensões são esperadas. Até o momento, é possível postergar a definição da implementação das classes Java de ações e restrições, ou então, a inclusão de qualquer elemento de lei. Abaixo, destacamos a estrutura usada para o mesmo exemplo anterior da restrição badClient e ação giveDiscount dentro da permissão Sale. Se esta permissão estiver ativa durante o processo de monitoramento, as ações referentes ao desconto e a restrição podem ser ativadas.

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

Código 9. Operador Abstract

Outro detalhe deste exemplo é que, como decisão de projeto, optamos por encapsular elementos referentes à política de descontos em uma permissão, isto visa somente estruturar e localizar esta responsabilidade e seus elementos correlacionados em um único contexto de definição. Nada impede que os

elementos Constraint e Action sejam definidos como abstratos, assim como no exemplo anterior.

3.1.1.2. completes

O atributo **completes** é um operador criado para preencher lacunas em elementos que foram definidos como abstratos (Código 10). 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 à 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

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

Código 10. Operador Completes

3.1.1.3. extends

O atributo **extends** é um operador mais poderoso, 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 (Código 11). Esta característica indica uma das diferenças entre o atributo **extends** e a operação de especialização de orientação a objetos, isto é, o atributo **extends** não permite a criação de uma hierarquia maior que dois níveis.

A partir deste momento, já existem instrumentos para implementar de forma simples os pontos de extensão em XMLaw.

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

Código 11. Operador Extends

3.1.2. Pontos de Extensão de Interação de SMAs Abertos

Argumentamos que leis de interação devem ser especificadas para facilitar extensões para aprimorar a manutenção de SMAs abertos. A manutenção não tem impacto restrito à implementação da solução, ela impacta o projeto também. Ao prever em projeto uma solução adequada a um cenário de evolução, a contribuição para a redução no esforço de alteração de um sistema pode ser grande. A interação é uma abstração de primeira ordem em SMAs abertos [Agha 1997], assim como objetos são para orientação a objeto. Qualquer solução para viabilizar a extensão de mecanismos de governança deverá considerar como fazer com que o projeto e a implementação de leis de interação possam ser facilmente customizados segundo diferentes especificidades.

Pontos de extensão de interação é a proposta desta tese para representar o conhecimento sobre o lugar onde modificações e aprimoramentos em leis podem ser feitos. Mesmo com pontos de extensão, a lei semi-completa pode ser referenciada por outros elementos de lei. Esta lei ainda abstrata só não pode ser utilizada pelo mecanismo de governança para monitorar a conversação de agentes, visto que nem todos os elementos estarão completamente definidos.

Como conseqüência de uma alteração em leis de interação, o próprio mecanismo de governança estará sendo customizado. Um ponto de extensão de interação prevê que novos elementos de lei poderão ser incluídos, por exemplo, em um ponto de extensão podem ser incluídas novas ações ou serviços, restrições para filtrar mensagens ou invalidar normas, ou outros ainda incluir outros elementos de leis. Mais do que só incluir é possível alternar estratégias prevendo o lócus onde a alteração se dará.

É importante que um ponto de extensão tenha claramente identificado o que está sendo modificado e tenha uma documentação suficiente para instruir um usuário daquela aplicação quase-completa de como irá proceder para estender ou

completar aquela solução, incluindo restrições e recomendações de uso. A documentação de pontos de extensão é bastante importante e será feita na forma de guias de referência do g-framework. É preciso saber que alterações podem ser feitas na solução e como estas alterações podem ser feitas.

3.2. G-Frameworks para a Governança de SMAs Abertos

Um mecanismo de governança pode precisar ser customizado de acordo com diferentes propósitos e peculiaridades, para isto, deve ser possível expressar evolução como variações a leis de interações de sistemas multiagentes abertos. A partir desta hipótese, os sistemas abertos são projetados utilizando pontos de extensão para anotar a especificação de interação e viabilizar a customização de leis para determinar o comportamento esperado para os agentes. Argumentamos que alguns elementos de especificação devem ser predefinidos em uma solução semi-completa para que sejam refinados no desenvolvimento de mecanismos de governança para domínios específicos.

Esta abordagem, chamada de g-frameworks, fornece instrumentos de análise, projeto e implementação de mecanismos de governança reutilizáveis. G-Frameworks irão demonstrar na prática a habilidade de aplicar as regras pré-definidas, ou quando necessário, customizar as regras semi-completas para um cenário de aplicação específico. Como já discutido, isto significa que o mecanismo de governança terá que apoiar a interpretação e a materialização de leis de interação abstratas. Neste sentido, aprimoramos a linguagem de descrição XMLaw com operadores de refinamento e o middleware MLaw para interpretar esta especificação de regras de interação em um mecanismo ativo de governança.

Como a técnica de g-frameworks é uma adaptação da técnica de frameworks orientados a objetos, discutiremos sucintamente na próxima seção as características de frameworks OO a partir de suas definições.

3.2.1. Inspiração em Frameworks Orientados a Objeto

A tecnologia de g-frameworks é inspirada em características de frameworks orientados a objeto e adaptadas ao contexto de leis de interação para a geração de mecanismos de governança de sistemas multiagentes abertos. 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 a reutilização em larga escala e 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]. Adiante estão descritas algumas definições interessantes de frameworks OO que denotam características importantes discutidas a frente.

Parnas [Parnas 1976] enfatizou de forma pioneira as vantagens da **reutilização de design**, observando a existência de **famílias de programas** que permitem a **construção de novas aplicações a partir de um design pré-existente**. Johnson [Johnson & Brian, 1988; Johnson, 1997] afirmou que um framework é "um esqueleto de uma aplicação que **deve ser parametrizado pelo desenvolvedor**" e "um conjunto de classes que representa um **design abstrato** para soluções em uma **família de aplicações**".

Pree [Pree, 1995] ressaltou que frameworks são "constituídos por pedaços de **software semi-acabados e prontos para usar**, sendo que reutilizar um framework significa **adaptar estes pedaços para uma necessidade específica**, através da redefinição de métodos e algumas classes". A grande contribuição trazida por este trabalho é sem dúvida a definição do termo hot-spot, que permite identificar de forma clara os pontos de extensão de um framework, que servirão de base para o processo de reutilização também aplicado em g-frameworks. Já Fayad [Fayad et al., 1999] define o termo frameworks como sendo "… uma tecnologia promissora para **materializar projetos e implementações** de softwares comprovados, levando a redução de custo e ao aumento a qualidade do software".

Adiante estão detalhados o propósito da técnica de g-frameworks e as diferenças determinantes entre esta abordagem e frameworks OO.

3.2.2. Propósito da Técnica de G-Frameworks

O propósito de g-frameworks é fornecer uma abordagem para apoiar a sistematização do desenvolvimento de mecanismos de governança. A tecnologia de g-frameworks prevê a reutilização de design e de leis de interação semi-acabadas para a produção de uma família de mecanismos de governança. Este design é considerado o núcleo do g-framework e será comum a todos os mecanismos gerados a partir dele. Para que um novo mecanismo seja gerado o g-

framework deve ser parametrizado pelo desenvolvedor em um processo de instanciação, onde os pontos de extensão de interação serão adaptados para uma necessidade específica da regulação do SMA aberto.

3.2.3. Diferenças Determinantes de G-Frameworks

É importante identificar claramente as diferenças entre as abordagens de frameworks OO e g-frameworks. A Tabela 8 lista estas diferenças e adiante explicaremos cada uma delas em mais detalhes. A primeira diferença está na abstração de primeira ordem utilizada em cada abordagem. Enquanto frameworks OO lidam com objetos, g-frameworks promovem a flexibilidade das leis de interação. Outra diferença está relacionada à definição de pontos de extensão; enquanto frameworks OO definem classes abstratas ou interfaces como pontos de alteração, g-frameworks definem elementos de leis como entidades abstratas que devem ser especializadas.

A terceira diferença está relacionada à forma de implementação destes elementos. Enquanto frameworks OO se baseiam em linguagens orientadas a objetos, como por exemplo, Java; g-frameworks são realizados com linguagens de especificação de leis de interação, como por exemplo, XMLaw. Enquanto o ambiente de execução de um framework OO é o ambiente de execução da linguagem de programação OO (por exemplo, a máquina virtual Java); em contrapartida, o ambiente de execução de um g-framework é um middleware de governança, como é o caso de MLaw.

O resultado de um framework OO é uma solução para desenvolvimento de famílias de aplicações orientadas a objeto; enquanto g-frameworks produzem uma família de mecanismos de governança. Sendo assim, uma instância de framework OO produz aplicações orientadas a objeto, enquanto uma instância de um g-framework produz um mecanismo de governança. O propósito de uma aplicação gerada por um framework OO é a computação de uma aplicação, enquanto que um mecanismo de governança gerado por um g-framework tem como propósito a verificação da conformidade da execução de agentes de software em um sistema aberto.

Quanto à especialização de um framework OO, a materialização de classes ou interfaces é feita modificando-se sua estrutura de dados e seu comportamento. A especialização OO permite ainda a sobrecarga (um mesmo elemento com

mesmo nome com parâmetros diferentes); permite sobrescrita (substituição do comportamento da classe mãe); e permite polimorfismo (é possível obter vários comportamentos, e até trocá-los eles em tempo de execução dependendo de qual implementação é usada). Já em g-frameworks, a materialização de elementos de leis ocorre modificando os seus relacionamentos, ciclo de vida e propriedades. Este procedimento não permite sobrecarga (um identificador único representa somente um elemento de lei); permite a sobrescrita (a especialização recebe, aprimora e substitui os elementos básicos de leis); e não prevê suporte ao polimorfismo.

O suporte em linguagem de programação para a realização da solução em termos de implementação conta com operadores de apoio em ambas as técnicas. Por exemplo, em frameworks OO gerados utilizando Java existem os operadores *abstract*, *extends*, *implements*; em g-frameworks utilizando XMLaw existem os operadores de refinamento *abstract*, *completes*, *extends*. O fluxo de execução de uma aplicação gerada por um framework OO é definida pela ordem de chamada de métodos. Em um g-framework, este fluxo é definido pela seqüência de ativação de eventos e a interdependência entre elementos de leis.

Quanto ao núcleo (elementos estáveis) de um framework OO, ele é composto por classes e relacionamentos. Enquanto isto, em um g-framework o núcleo é composto por elementos de leis e pelas dependências de ativação de elementos. A modularização de um framework OO é feita em termos de pacotes e classes; já em g-framework os contextos como leis, normas e cenas são os blocos de construção de elementos.

Existe ainda uma distinção quanto à obrigatoriedade de implementação dos elementos referenciados nas duas abordagens. Em frameworks OO, a implementação de todas as classes e interfaces referenciadas é necessária para gerar uma instância. Enquanto isto em um g-framework, a implementação de todos os elementos de leis é necessária, porém os agentes são caixas pretas e não estão relacionados à instanciação do g-framework, isto é, o mecanismo de governança está pronto para o uso, independentemente dos agentes referenciados por ele.

Diferença	Frameworks OO	G-Frameworks
Abstração de primeira ordem	**Objetos**	**Interação**
Definição de Pontos de Extensão	**Classes ou Interfaces**	**Elementos de leis**
Linguagem de Programação	**OO - exemplo: Java**	**Leis de interação - exemplo XMLaw**
Instância	**Aplicação**	**Mecanismo de governança**
Resultado	**Família de Aplicações**	**Família de Mecanismos de Governança**
Propósito	**Computação**	**Verificação de conformidade**
Especialização	**Materialização de classes ou interfaces, modifica sua estrutura de dados e seu comportamento.**	**Materialização de elementos de leis, modifica seus relacionamentos, ciclo de vida e propriedades.**
Sobrecarga	**Permite a sobrecarga: um mesmo nome com parâmetros diferentes.**	**Não permite sobrecarga: um identificador único representa somente um elemento em XMLaw.**
Sobrescrita	**Permite sobrescrita onde pode ser alterado o comportamento da classe mãe.**	**Permite a sobrescrita onde a especialização recebe, aprimora e substitui os elementos básicos de leis.**
Polimorfismo	**Permite polimorfismo: é possível obter vários comportamentos, e até trocar eles em tempo de execução dependendo de qual a implementação é usada.**	**Não prevê suporte a polimorfismo.**
Suporte em linguagem de programação	**Java – abstract, extends, implements**	**XMLaw - abstract, completes, extends**
Fluxo de execução	**Definido pela ordem de chamada de métodos.**	**Definido pela seqüência de ativação de eventos.**
Núcleo (elementos estáveis)	**Classes e relacionamentos**	**Elementos de leis e dependências de ativação de elementos**
Distinção sobre a obrigatoriedade de implementação de elementos referenciados	**A implementação de classes e interfaces é necessária para gerar uma instância de um framework OO**	**A implementação de elementos de leis é necessária, porém a implementação de agentes não está relacionada à instanciação do g-framework.**
Modularização	**Pacotes e classes**	**Contextos (leis, normas e cenas)**
Ambiente de execução	**Máquina virtual Java**	**MLaw**

Tabela 8 – Diferenças entre Frameworks OO e G-Frameworks

Mecanismos de governança regulados por leis baseados em XMLaw referenciam somente a definição de papéis e as regras de interação que compõem o que chamamos de leis. Esta técnica visa aprimorar a manutenibilidade das leis de interação e deixar a implementação dos agentes para seus provedores, respeitando assim uma das características principais de sistemas abertos. Em um sistema multiagente, a estrutura de colaboração define os papéis de agentes e seus relacionamentos. Papéis são úteis para especificar descrições gerais de responsabilidades de agentes em uma organização [Yu & Schmid 1999] e são

utilizados para adequar/restringir os agentes reais ao cenário esperado deles em um sistema aberto. Quando um agente executa um papel, ele adquire a obrigação de obedecer às leis que são especificadas como suas responsabilidades. A partir desta premissa, é possível monitorar a conformidade com as leis prescritas no protocolo de interação. Portanto, a maior preocupação da abordagem de g-frameworks não é discutir como estruturar a reutilização de agentes de software como papéis [Zambonelli et al. 2001; Kendall 2000]. Estes papéis serão realizados como agentes de software externos a aplicação.

3.2.4. Por que adaptar Frameworks OO?

Quando falamos de reutilização devemos pensar nas duas dimensões de variabilidade discutidas anteriormente: projeto e implementação. De forma geral, a reutilização de código puro pode ser restritiva, pois no processo de implementação de uma determinada abstração em uma linguagem de programação, as idéias originais (a abstração) são normalmente intercaladas e escondidas pelo idioma da linguagem utilizada [Jacobson et al., 1997]. Isto não permite que todo ou parte do conhecimento adquirido durante o processo de desenvolvimento do artefato reutilizável seja reaproveitado em situações diversas [Jacobson et al., 1997]. Procuramos na abordagem de g-frameworks conciliar o projeto (com pontos de extensão de interação) e a implementação (com operadores de refinamento).

Independente de um código reutilizável torna-se necessário que a técnica de g-framework seja capaz de facilitar o entendimento, a estruturação e a produção de mecanismos de governança manuteníveis. A técnica de frameworks OO se presta a este mesmo propósito, pois promove a reutilização de design de aplicativos, o que já se mostrou bastante promissora por diversos fatores apresentados em [Garlan & Shawn, 1996]. Visamos manter algumas características na adaptação da técnica de frameworks OO para g-frameworks. Para a nossa adaptação merecem destaque:

- Melhoria no entendimento do sistema em geral, através da distinção entre pontos fixos e flexíveis;
- Melhoria na captura de erros de projeto, uma vez que a reutilização é decidida em fases iniciais do desenvolvimento; e

- Indução à reutilização de código, uma vez que este design necessita de uma representação "executável" que comprove sua utilidade.

3.2.5. Arquitetura de G-Frameworks

Ao analisar um domínio de sistemas de software aberto, é possível distinguir dois grupos de especificações em torno da interação de agentes: fixa (estável/núcleo) e variável (extensível). Por esta análise, é possível projetar parte da evolução do sistema aberto na própria solução. Se uma característica desejável para os mecanismos de governança apresenta uma baixa variabilidade, isto é, não deverá se alterar, então se deve reproduzir esta estabilidade em um projeto modular que apresente esta característica e facilite a produção de uma família de mecanismos de governança. Caso a característica demonstre um maior grau de variabilidade, deve-se defini-la como abstrata.

Um g-framework é flexível por design. Flexibilidade é o oposto do conceito de leis e protocolos de interação estáticos e totalmente pré-definidos. Em tempo de projeto, a preocupação com a customização irá garantir que o g-framework possa receber novos elementos, ou adaptar elementos já existentes. Para este propósito um g-framework provê "hooks" para suas instâncias; definindo especificações de leis abstratas como referência/modelos. As funcionalidades que são específicas aos seus mecanismos de governança somente serão implementadas por completo depois, no entanto, todas as definições e implementações comuns são desenvolvidas *a priori* e disponibilizadas para todos os mecanismos de governança. Com isto temos que a realização de interações abstratas é postergada para o tempo de instanciação.

Uma solução para o desenvolvimento deste tipo de solução é atingida ao relaxar a fronteira entre o g-framework (a parte comum de uma família de mecanismos de governança) e suas instâncias (a parte específica de um mecanismo de governança). Como já discutido em um g-framework, determinados elementos de leis do sistema aberto são definidos como abstratos, porque não foram detalhados completamente, pois iriam expor detalhes que variariam entre implementações particulares de mecanismos de governança.

Os elementos que compõem as diretrizes para a modularização de g-frameworks de sistemas abertos regulados por leis são (Figura 15): (i) núcleo de leis de interação; e (ii) pontos de extensão de interação.

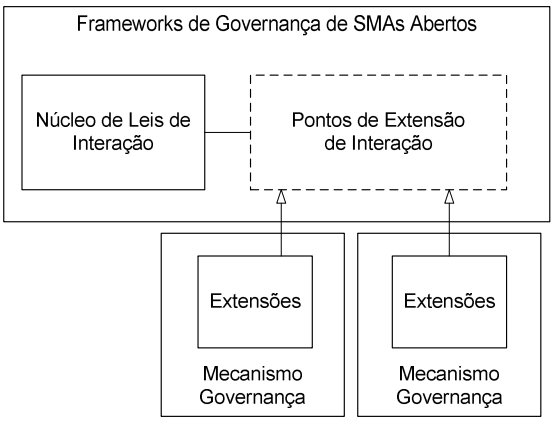

Figura 15 – Visão Geral de G-Frameworks de SMAs Regulados por Leis

A parte fixa da especificação de interações é chamada de núcleo de leis de interação. Este núcleo pode ser derivado pela análise de domínio. Se qualquer elemento de interação é comum a todas as instâncias desejadas, este elemento é associado ao núcleo. O núcleo do g-framework implementa funcionalidades obrigatórias e elementos de leis fixos em uma família de mecanismos de governança. A estabilidade também é caracterizada pela estrutura geral do protocolo de interação e por algumas regras que são comuns a todas as instâncias de mecanismos de governança naquele domínio. Este núcleo também deve estar preparado para customizações para propósitos específicos e pré-determinados. Nele serão identificados os pontos exatos onde é possível modificar a solução.

A variabilidade de interações implica a especificação de protocolos mais flexíveis para incluir alternativas e opções de projeto. Estas extensões de regras de interação impactarão o sistema aberto e deverão ser claramente identificadas com pontos de extensão. Estes devem ter uma ligação e dependência explícita ao núcleo. Pontos de extensão de interações podem ser utilizados para expor um conjunto de eventos do g-framework que será implementado *a posteriori*. Desta forma eles têm como propósito a documentação da solução a ser customizada na instanciação do g-framework. Existem guias de referência associados aos pontos de extensão que facilitam o entendimento do projeto, propósito e como reutilizar a solução gerada. A próxima seção detalhará esta forma de documentação.

3.2.6. Documentando G-Frameworks

Quando falamos de artefatos reutilizáveis, o problema da documentação é de suma importância, uma vez que, além do usuário da documentação intitulado mantenedor/desenvolvedor, existe também o reutilizador que 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 artefatos reutilizáveis precisa apresentar de forma clara não só suas estruturas internas, mas também como sua (re) utilização deverá proceder.

O objetivo desta seção é apresentar uma abordagem para a documentação da solução de g-frameworks. Adiante, descreveremos sucintamente um estudo sobre diferentes abordagens para a documentação de soluções reutilizáveis, mantendo sempre em mente adaptar trabalhos que tenham relação com frameworks OO.

3.2.6.1. Abordagens de Documentação em prol de Reutilização

Johnson (1992) afirma 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. 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 16 – Resumo da Estrutura da Documentação de Johnson (1992)

É sabido que o reutilizador precisa conhecer os pontos de extensão e a forma usada para estendê-los. Abordagens como UML-F [Fontoura, 1999; Fontoura et al., 2001], CookBook [Krasner & Pope, 1988] e Hooks [Froehlich et al., 1997] auxiliam este processo permitindo que o projetista do framework

especifique como a sua solução deve ser instanciada. Cookbook foi originalmente proposto em [Krasner & Pope, 1988] como um tutorial para a utilização do framework Model-View-Controller (MVC) presente na biblioteca de classes de Smalltalk. 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.

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

Figura 17 – Resumo da estrutura da documentação Cookbook

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 deste *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*, é onde se colocam as alterações como herança e redefinições de métodos (**Mudanças**); indicam as restrições impostas ao uso do *hook* (**Restrições**) e descrições adicionais necessárias (**Comentários**).

Nome <nome_único_contexto_framework>	
Requisito <problema_solucionado>	
Método de Adaptação	**Suporte Oferecido**
Áreas <partes_do_framework_afetadas>	
Usa <hooks_relacionados>	
Participantes <componentes_relacionados>	
Mudanças <descrição_alterações_necessárias>	
Restrições <restrições_impostas_ao_uso_hook>	
Comentários <informações_adicionais>	

Figura 18 – Estrutura da Documentação de Froehlich et al. (1997)

Fontoura (1999) propôs uma linguagem como extensão de UML para capturar os pontos de extensão no nível de design. Esta linguagem permite expressar pontos de extensão como métodos de variação, classes de extensão e interfaces. Um fator importante desta abordagem é o mapeamento dos pontos de extensão existentes em possíveis formas de instanciação, que permitem uma rápida execução pelo desenvolvedor da aplicação. De forma adicional este trabalho também apresenta uma série de ferramentas baseadas em Prolog [Bowen, 1979], que auxiliam o usuário do framework no processo de instanciação. Este trabalho pode ser promissor a partir do momento em que exista uma notação gráfica de representação associada à XMLaw.

3.2.6.2. Documentação de G-Frameworks

A partir destas informações é importante observar que um g-framework deve ter uma descrição clara referente a decisões de projeto. Estas decisões incluem o framework (solução como um todo), seus pontos de extensão, e os requisitos que derivaram estas decisões de projeto. Propomos que estas três preocupações estejam claramente documentadas de forma relacionada (Figura 19). Isto visa facilitar o entendimento geral do projeto (granularidade alta), o entendimento dos detalhes de pontos de extensão (granularidade baixa), e o entendimento da correlação entre requisitos identificados e a decisão por pontos de extensão (casos de leis). A organização lógica de documentação de um g-framework precisa prever o relacionamento entre estes diferentes níveis.

Figura 19 – Níveis de documentação de um g-framework

O objetivo da documentação de um g-framework é facilitar o entendimento do escopo de aplicação daquela solução, e facilitar o processo de identificação se aquela proposta realmente resolve o problema desejado. Outra preocupação desta documentação é auxiliar o usuário do g-framework no seu processo de instanciação. A partir destas premissas propusemos uma estrutura para a documentação do g-framework baseada em [Johnson, 1992] e [Krasner & Pope, 1988]. A estrutura proposta (Figura 20) para o guia de referência de um g-

framework propõe que a sua documentação deve abranger o nome, propósito, versão, design do g-framework, e instruções de como utilizar a solução.

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 20 – Estrutura de Guia de Referência de G-Frameworks

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. O campo numérico **Versão** denota opcionalmente a versão do g-framework. Um propósito é normalmente consultado na fase de análise de domínio para a verificação da reusabilidade no contexto em análise. 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. O campo **Pontos de extensão** lista os pares elemento : nome dos pontos de extensão previstos nesta abordagem. Por fim, o campo **Como instanciar** descreve como o reutilizador deve proceder durante o processo de instanciação, sendo uma enumeração seqüencial de passos em texto não estruturado. Com relação ao campo **Design**, não impomos nenhuma restrição quanto à natureza de informação trazida, isto é, pode ser texto não estruturado ou mesmo um diagrama baseado em alguma notação gráfica. No entanto, a informação presente neste campo deve ser suficiente para promover o entendimento sobre o projeto da solução. O exemplo de guia de referência do g-framework de cadeias de suprimento (TAC SCM) está ilustrado adiante (Figura 21).

O outro nível de detalhamento previsto em g-frameworks são os pontos de extensão. Cada ponto de extensão também terá uma estrutura própria para a sua documentação. A estrutura proposta neste trabalho é baseada em Hooks [Froehlich et al., 1997]. Esta estrutura (Figura 22) detalha aspectos relevantes e alterações necessárias para usar o ponto de extensão. Em sua estrutura, 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 seção principal desta documentação e descreve as alterações que devem ser feitas nos elementos de leis, é onde se colocam 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.

O preenchimento dos campos Elementos criados, Dependência de Elementos, e Dependência de Eventos é opcional. A seção **Ameaça** deve ser utilizada para relacionar este ponto de extensão a um Caso de Lei descrito na fase de requisitos. Isto foi proposto de forma a apoiar a rastreabilidade dos artefatos gerados ao longo do ciclo de vida de um g-framework. Adiante estão descritas as documentações dos pontos de extensão previstos para o TAC SCM (Figura 23).

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 21 – Guia de Referência do G-Framework TAC SCM

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 22 – Estrutura de Documentação de um Ponto de Extensão

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 23 – Documentação do Ponto de Extensão checkDueDate

3.2.7. Visão Geral do Desenvolvimento de G-Frameworks

É importante observar que o desenvolvimento de g-frameworks para sistemas multiagentes abertos 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. Espera-se que nesta etapa sejam feitos os seguintes passos:

1. Análise cuidadosa do domínio de aplicação,
2. Documentação cuidadosa dos requisitos utilizando casos de leis,
3. Projeto da solução com a distinção entre o núcleo e os pontos de extensão de interação seja bem feito;
4. Implementação detalhada utilizando operadores de refinamento; e
5. Documentação da solução para facilitar a reutilização.

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 de interação 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 de g-frameworks não existe orientação para a forma com que os agentes de software devem ser desenvolvidos. A única restrição existente é que eles utilizem a camada de comunicação desenvolvida junto ao mediador MLaw.

3.2.8. Desafios do uso de G-Frameworks

Uma vez que o desenvolvimento de sistemas complexos é difícil, o desenvolvimento destes sistemas de forma abstrata tendo em mente reutilização é mais difícil ainda. É necessário, além de tempo, o auxílio de especialistas no domínio para o qual g-frameworks está sendo desenvolvido, bem como uma boa dose de criatividade.

Um problema comum no processo de reutilização é o tempo necessário para se tornar capacitado para obter vantagens desta reutilização. Quando tratamos de g-frameworks, isto não é diferente. Sendo assim, a menos que o custo de aprendizagem seja amortizado por vários projetos ou que o ganho de produtividade e qualidade seja expressivo, este investimento não se tornará atraente. Este esforço adicional tende a ser recompensado pelo aumento na qualidade da solução gerada e na facilidade em reaplicação da solução em diferentes contextos.

Como todo artefato de software, seus requisitos iniciais evoluem com o tempo, obrigando a criação de novas versões do g-framework. As aplicações instanciadas a partir de um dado g-framework também devem evoluir com a finalidade de se manterem de acordo com a especificação do g-framework. Este problema está relacionado com a modificação de aplicações que já estão em produção, o que pode ser problemático quando, por exemplo, o serviço prestado por estas aplicações não puder parar. Trabalhos com evolução dinâmica não

antecipada foram explorados ao longo do estudo, mas ficaram fora do escopo desta tese.

3.3. Conclusão

A abordagem de g-frameworks oferece técnicas para resolver o problema de construir mecanismos de governança que garantam que agentes interagirão conforme uma solução bem definida, porém customizável. Nosso objeto principal é contribuir para a engenharia de como é possível se produzir e reutilizar leis de interação.

Foram apresentadas as técnicas de apoio à reutilização de leis de interação em sistemas multiagentes abertos desenvolvidos com g-frameworks. Apesar de baseada na tecnologia de frameworks orientados a objeto, g-frameworks trabalham com uma natureza de elementos diferentes de classes e objetos. Cada elemento de lei tem sua particularidade própria que foi descrita na seção de variabilidades de leis de interação. Esta diferença terá impacto sobre o núcleo, pontos de extensão, fluxo de execução, modularização e sobre alternativas de especialização possíveis. É importante destacar a estrutura de documentação apresentada para g-frameworks visa registrar informações em conjunto com a técnica de Casos de Leis para promover o entendimento desta abordagem.