2
Técnicas de Governança de Sistemas Multiagentes Abertos em G-Frameworks

Como no desenvolvimento de sistemas multiagentes abertos os agentes são implementados de forma independente, é necessário fornecer um instrumento para atingir um sistema coerente. Leis de interação são usadas para mitigar riscos em SMAS abertos que poderiam gerar diversos tipos de falhas [Leishman & Vanburen 2003]. Leis são regras de interação que especificam a integridade de um sistema aberto através de um conjunto de propriedades que devem ser obedecidas. Uma analogia com assertivas [Staa, 2000] pode ser feita, pois leis também procuram interceptar falhas dinamicamente.

De forma centralizada ou distribuída, o controle da integridade das leis ao executar o sistema é feito por mediadores que interceptam as mensagens ou informações de um ambiente e aplicam as leis ou penalidades previstas e especificadas. Alguns trabalhos foram propostos como forma de assegurar a governabilidade de sistemas através da mediação de leis de interação, dentre eles [Castelfranchi et al. 1999; Dignum, 2001; Dignum & Dignum 2001; Esteva 2003; Esteva et al. 2003; Jones & Sergot, 1993; Kollingbaum & Norman, 2003; Martin et al., 1999; Mineau, 2003; Minsky & Ungureanu, 2000; Rodriguez-Aguilar, 2001; Schumacher et Ossowski, 2005; Vazques-Salceda et al. 2005].

Como este trabalho tem o objetivo de apoiar a construção de mecanismos de governança de sistemas multiagentes abertos, será utilizada e aprimorada a abordagem proposta por [Paes et al. 2005] através da linguagem de especificação de leis de interação chamada XMLaw. Isto se deve ao fato de XMLaw ser uma linguagem declarativa, com abstrações e de alto nível que podem contribuir para a engenharia destes mecanismos. Esta linguagem é associada ao *middleware* (MLaw) [Paes et al. 2007] que implementa o mecanismo de regulação das interações dos agentes fazendo a verificação de conformidade de mensagens em tempo de execução. Por questões de foco e espaço, as vantagens que o XMLaw apresenta em relação às demais abordagens não serão discutidas, e podem ser encontradas em [Paes et al. 2005].

Não só a existência de leis é importante como também a descrição e documentação de requisitos elicitados para a geração de elementos de leis de interação. Para este propósito, ao final deste capítulo será apresentada a proposta de Casos de Leis.

Estas técnicas compõem as ferramentas utilizadas por um desenvolvedor de g-frameworks para a elaboração de uma família de mecanismos de governança de sistemas multiagentes abertos. Para haver um maior entendimento da solução proposta, uma visão geral da linguagem XMLaw será descrita adiante, seguida da apresentação do *middleware* MLaw e de Casos de Leis para a documentação de requisitos. O exemplo Trading Agent Competition – Supply Chain Management (TAC SCM) será apresentado como forma de facilitar o entendimento desta abordagem.

2.1. TAC SCM

Esta seção apresentará o exemplo escolhido para auxiliar na apresentação das técnicas propostas. O domínio desta aplicação é a negociação eletrônica em cadeias de suprimento, em inglês, *Supply Chain Management* (SCM). Aplicações de SCM incluem o planejamento e coordenação das atividades de uma cadeia de suprimentos, onde compradores e vendedores dependem uns dos outros [Chopra & Meindl 2004]. Essas atividades podem ter vários participantes e organizações, e a coordenação desses participantes é um ponto chave para uma cadeia eficiente. Uma cadeia de suprimento é extremamente dinâmica e envolve um grande fluxo de informação, produtos e recursos financeiros entre diferentes estágios. Um dos pontos de maior destaque atualmente é como fazer com que os participantes da cadeia de suprimentos obedeçam a regras ou contratos previamente estabelecidos.

O TAC SCM [Sadeh et al., 2003] é uma competição que foi desenvolvida por pesquisadores do laboratório de *e-Supply Chain Management* da *Carnegie Mellon University* e *Swedish Institute of Computer Science* (SICS), e suas últimas edições foram realizadas em encontros oficiais junto a conferências de renome como o AAMAS. Esta competição foi modelada para capturar a complexidade de uma cadeia de suprimentos dinâmica, porém com regras simples para que muitas equipes possam participar.

Basicamente, seis agentes "produtores de PC" participam em cada rodada do TAC SCM (Figura 1). Esses participantes competem por clientes com

incerteza na demanda e por peças de um número limitado de fornecedores. Todo dia, os clientes enviam pedidos de cotações e selecionam as melhores ofertas baseadas no preço e na data de entrega. Os agentes são limitados pela capacidade da fábrica, e têm que gerenciar a compra de peças de oito fornecedores e sua montagem para a entrega aos seus clientes. Quatro componentes são precisos para montar um PC: CPU, Placa Mãe, Memória, e Disco Rígido. Cada componente está disponível em diferentes modelos. A competição começa quando dois ou mais agentes se conectam ao servidor do TAC SCM. O servidor simula o comportamento dos fornecedores e clientes, e oferece um ambiente para cada participante. Este ambiente é composto por um banco que controla os pagamentos e as contas corrente, uma fábrica que monta os PCs para o agente, e informações sobre inventário como estoques de peças e PCs. No final da competição, o agente que possuir o maior valor em dinheiro na conta corrente é declarado como o vencedor.

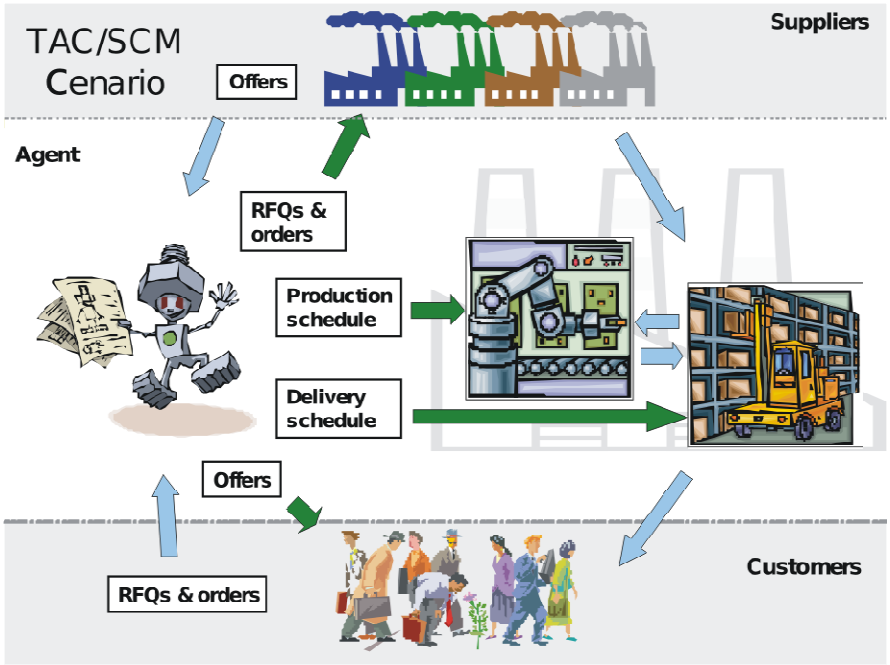

Figura 1 – Cenário da Competição TAC SCM

A competição apresenta um bom nível de complexidade, pois cada agente deve competir em vários mercados por diferentes componentes do lado dos fornecedores, e em mais outros mercados por diferentes contratos de compra de PCs no lado dos clientes. Além disso, todos esses mercados possuem interdependências e incertezas. A partir deste cenário, o modelo conceitual e a linguagem XMLaw serão apresentados na próxima seção para a especificação de leis de interação.

2.2. XMLaw

Em geral, os aspectos internos dos agentes em SMAs são inacessíveis, e a única informação disponível sobre os agentes é o comportamento observável por meio das mensagens que eles emitem ao se comunicarem com os outros agentes. O propósito de XMLaw é ser uma linguagem para expressar possibilidades e restrições sobre interações em um sistema multiagente aberto. A observação prevista no modelo de XMLaw é baseada em eventos, portanto o evento de chegada de uma mensagem é algo perceptível no contexto de leis. E este evento pode ser propagado, dando início a um ciclo de notificação determinado pela configuração e relacionamento de elementos de leis. Esta é a premissa básica e serviu de base para a proposição dos elementos presentes atualmente no modelo conceitual de XMLaw. A Figura 2 ilustra o modelo conceitual do XMLaw utilizado para regular suas interações. Estes elementos correlacionados compõem uma lei de interação. Elementos deste modelo são descritos utilizando uma linguagem declarativa baseada em XML (Código 1).

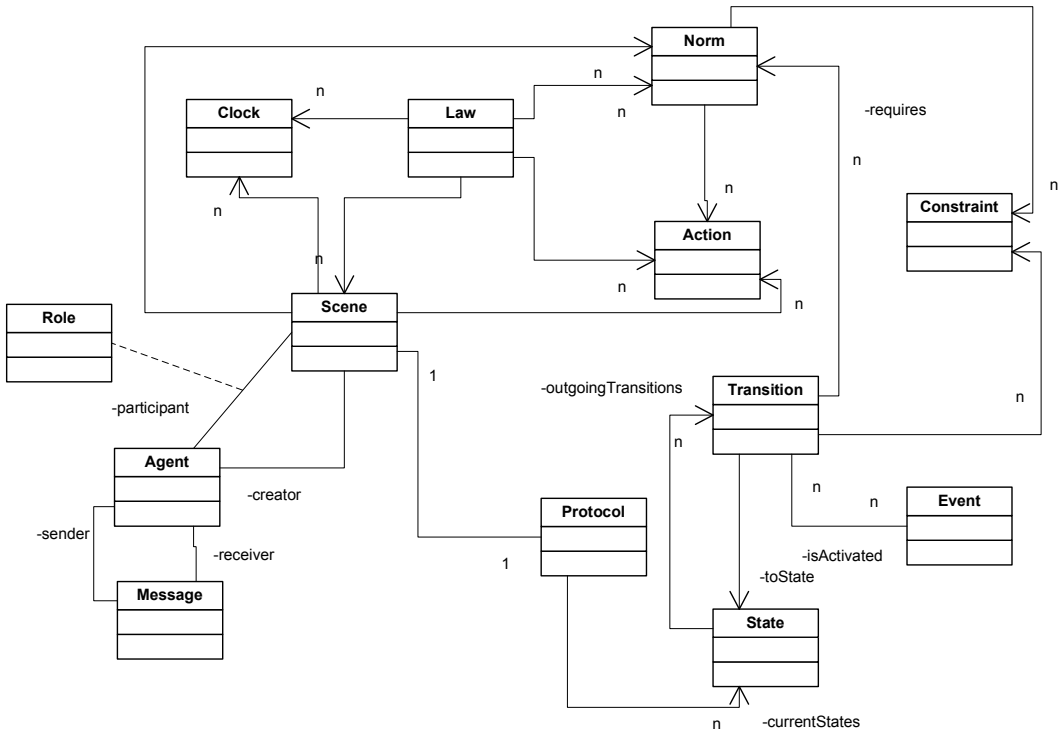

Figura 2 – Modelo Conceitual de XMLaw

O primeiro fato importante a observar é que os elementos de XMLaw não só percebem, como também se comunicam por meio de eventos. A cadeia de eventos pode ser responsável por um conjunto de ativações ou desativações relacionadas

ao ciclo de vida dos elementos, e previstas segundo as dependências pré-estabelecidas pelos elementos. Um exemplo simples é a chegada de uma mensagem que gera um evento (message_arrival) e que por sua vez pode ativar uma transição (transition_activation), e assim por diante. Adiante explicaremos a semântica de cada elemento deste modelo.

```
<Law id="…" name="…">
 <Scene id="…" time-to-live="…">
  <Creators>…</Creators>
   <Entrance>…</Entrance>
   <Messages>…</Messages>
   <Protocol>
     <States> … </States>
     <Transitions>…</Transitions>
   </Protocol>
   <Norms>... </Norms>
   <Clocks>...</Clocks>
   <Actions>...</Actions>
  </Scene>
 </Law>
```

Código 1. Estrutura de uma Lei de Interação com Código XMLaw

O elemento *Message* corresponde à descrição da estrutura e do conteúdo de mensagens enviadas ou recebidas por agentes. Ele estabelece o padrão esperado para a observação de mensagens de agentes. Isoladamente, este elemento não permite especificar muitas restrições em uma interação. É preciso estabelecer a ordem em que as mensagens ocorrem e quais são as mensagens válidas em um determinado momento da interação. E esta ordem é definida por uma dependência entre eventos internos estruturada em um protocolo de interação. O Código 2 descreve a estrutura básica de mensagens de um protocolo de pagamento. A Tabela 1 descreve os eventos que o elemento Message pode emitir.

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

Código 2. Descrição das Mensagens de Pagamento

Evento	Descrição
message_arrival	**Representa que um agente enviou uma mensagem e que esta mensagem foi interceptada pelo mediador.**
compliant_message	**Representa que a mensagem casa com o padrão de mensagem esperado pela lei.**

Tabela 1 – Lista de Eventos do Elemento Message

Protocolos de interação representam o fluxo de estados previstos em uma conversação de agentes, definindo quais são as interações válidas e inválidas. O elemento *Protocol* representa protocolos de interação entre os agentes e é representado por um autômato finito não determinístico [Menezes, 1997], onde estados (elemento *State*) representam pontos na execução do protocolo e transições (elemento *Transition*) são as conexões entre os estados. Estados possuem um atributo do tipo que caracteriza quando este é considerado como estado inicial, estado de execução, estado final com sucesso e estado final com falha. Uma transição é acionada por eventos. Ao ser acionada, o protocolo evolui do estado origem da transição para o destino. Uma transição pode ter uma restrição a ela associada (elemento Constraint). Neste caso, a transição só será ativada quando a restrição permitir. Uma restrição é implementada como um componente Java. Uma transição pode ainda requerer que um elemento *Norm* (*Prohibition*, *Obligation* ou *Permission*) esteja ativo para que o protocolo evolua para outro estado. Se este elemento não estiver ativo, a transição não será disparada. O Código 3 transcreve o protocolo de pagamento previsto para o TAC SCM. A Tabela 2 descreve os eventos relacionados ao protocolo, incluindo transições e estados.

```
<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 3. Descrição do Protocolo de Interação de Pagamento do TAC SCM

Evento	Descrição
transition_activation	**Representa uma mudança de estado do protocolo, i.e, a transição que conecta o estado origem ao destino foi ativada.**
failure_state_reached	**Representa quando o protocolo chega a um estado terminal de falha a partir de uma transição.**
successful_state_reached	**Representa quando o protocolo chega a um estado terminal de sucesso a partir de uma transição**

Tabela 2 – Lista de Eventos Relacionados ao Elemento Protocol

O modelo conceitual fornece um elemento denominado *Clock* (ou relógio) que fornece os meios para especificar leis sensíveis ao tempo. Um relógio é capaz

de gerar eventos em intervalos de tempo específicos. Por exemplo, um relógio permite a ativação e desativação de normas após um determinado período de tempo ter se esgotado. Além disso, um relógio pode funcionar como um controle de *timeout*. Relógios são ativados por eventos (exemplos, por transições ou mesmo por normas). Um relógio pode ser definido como do tipo regular que indica que ele emitirá um único evento ao término do período de ativação (tick-period) e será desativado, e outro tipo definido como periódico, que ciclicamente emite um evento ao final do período de ativação. Um relógio pode ser definido em diferentes contextos, incluindo o de Lei (elemento Law) e o de cena (elemento Scene). Estes contextos têm como objetivo determinar a visibilidade e a propagação de eventos. O Código 4 transcreve o clock de marcação de novos dias no TAC SCM. A Tabela 3 descreve os eventos relacionados ao elemento Clock.

```
<Clocks>
   <Clock id="nextDay" type="regular" tick-period="12000">
      <Activations>
         <Element ref="negotiation"
            event-type="scene_creation"/></Activations>
   </Clock>
...
</Clocks>
```

Código 4. Implementação do Clock nextDay do TAC SCM

Evento	Descrição
clock_activation	**Sinaliza que um clock foi ativado e que começou a contar o período de tempo determinado.**
clock_tick	**Sinaliza que um ciclo com duração especificada no atributo tick-period terminou.**
clock_timeout	**Sinaliza quando um clock do tipo "once" gerou o evento clock-tick, e portanto finalizou a sua execução.**
clock_deactivation	**Sinaliza que houve desativação (encerramento da execução) do clock, causada por algum evento especificado na lista de desativações.**

Tabela 3 – Lista de Eventos Relacionados ao Elemento Clock

Normas (*Norms*) descrevem quais comportamentos dos agentes são permitidos, quais são obrigatórios e quais não são permitidos (proibidos). As normas são geralmente adquiridas pelos agentes no decorrer das interações, e conseguem representar noções de compromissos adquiridos e cumpridos. Por exemplo, em um processo de negociação, um agente pode assumir o compromisso (obrigação) de pagar por uma mercadoria negociada e, enquanto essa obrigação não for cumprida, o agente fica impedido de participar de novas negociações. Uma norma pode encapsular restrições, relógios e ações, isto é, estes elementos são definidos dentro da norma e somente serão ativados se a norma estiver ativa.

Uma restrição (elemento Constraint) encapsulada em uma norma tem como objetivo confirmar a validade daquela norma segundo um evento específico (e.g. a chegada de uma mensagem). Independente de estar ativa, uma norma só é válida quando as restrições assim disserem. Uma ação (elemento Action) encapsulada em uma norma só pode ser ativada quando e enquanto a norma estiver ativa, o mesmo vale para relógios e restrições. O Código 5 transcreve uma norma do TAC SCM. Esta norma impõe restrições aos valores de atributos permitidos para as ofertas feitas pelos agentes fornecedores. A Tabela 4 descreve os eventos relacionados a uma norma. A Tabela 5 descreve o evento relacionado a uma restrição (elemento Constraint).

```
<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 5. Especificação Geral de uma Norma do TAC SCM

Evento	Descrição
norm_activation	**Sinaliza que a norma foi ativada**
norm_deactivation	**Sinaliza que a norma foi desativada dada a ocorrência de um evento de desativação.**

Tabela 4 – Lista de Eventos Relacionados ao Elemento Norm

Evento	Descrição
constraint_activation	**Sinaliza que uma constraint foi executada e o seu retorno foi verdadeiro.**

Tabela 5 – Lista de Eventos Relacionados ao Elemento Constraint

O modelo conceitual utiliza a abstração de cenas para auxiliar na organização e modularização das interações, representadas pelo elemento do modelo conceitual *Scene*. Este elemento especifica quais agentes e quais os papéis de agentes que podem interagir em uma cena, ou mesmo dar início a sua

execução. Além disso, uma cena é composta por um protocolo de interação e por um conjunto de normas, ações e relógios. Estes elementos compartilham um contexto comum de interação definido pela cena. Isto significa que uma norma definida no contexto de uma cena é somente visível naquela cena. Para que um agente inicie ou participe de um protocolo de interação de uma cena, é necessário que ele desempenhe algum papel previamente definido e especificado no contexto da cena. No Código 6 é possível observar um exemplo resumido de uma cena de negociação do TAC SCM. Nenhum detalhe sobre protocolo, normas, relógios, ou ações são apresentados neste exemplo. A Tabela 6 descreve os eventos relacionados ao elemento Scene.

```
<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 6. Estrutura da Cena de Negociação do TAC SCM

Evento	Descrição
failure_scene_completion	**Representa que uma cena foi encerrada com falha porque o protocolo gerou um evento failure_state_reached.**
sucessful_scene_completion	**Representa que uma cena foi encerrada com sucesso porque o protocolo gerou um evento successful_state_reached.**
scene_creation	**Sinaliza que a cena foi criada**

Tabela 6 – Lista de Eventos Relacionados ao Elemento Scene

Por fim, ações são serviços acessíveis ao mediador durante o processo de monitoramento das mensagens dos agentes. Um elemento Action é capaz de tornar o mediador em uma entidade ativa dentro do processo de garantia de aderência às leis. Por exemplo, uma mensagem de notificação de erro pode ser enviada pelo mediador a um agente a partir de uma ação criada para isto, ou então, é possível associar algum processamento a determinada etapa de monitoramento, para guardar alguma informação relevante ao processo de mediação. Ações são implementadas como componentes Java. O Código 5 transcreve uma ação definida no contexto de uma norma do TAC SCM. A Tabela 7 descreve os elementos relacionados a uma norma.

Evento	Descrição
action_activation	**Sinaliza que uma ação foi ativada e está em execução**

Tabela 7 – Lista de Eventos Relacionados ao Elemento Action

O elemento *Law* (Lei) corresponde ao contexto de definição de uma organização e engloba todos os demais elementos.

A partir da explicação do modelo conceitual e de sua representação em XMLaw, é possível descrever a estrutura de software gerada para interpretar a linguagem e implementar o comportamento de mediação de leis de interação. Adiante será apresentado MLaw, o middleware de mediação desenvolvido para esta finalidade.

2.3. MLaw

Como mencionado anteriormente, além do modelo conceitual apresentado, existe um *middleware*, chamado MLaw [Paes et al. 2007], que garante a regulação das interações dos agentes através dos elementos do modelo conceitual e do modelo de interação baseado em eventos. Este mecanismo intercepta as mensagens enviadas por agentes e verifica sua conformidade com as leis (Figura 3).

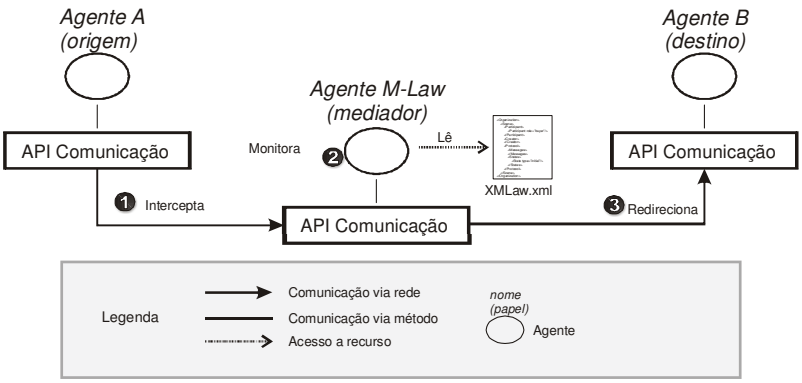

Figura 3 – Ciclo de Operação do Mediador MLaw

Para que o *middleware* MLaw pudesse prover os meios para efetivamente dar suporte à linguagem XMLaw e sua evolução, ele foi desenvolvido com cinco módulos (Figura 4).

O módulo de Comunicação é utilizado para enviar e receber mensagens entre os agentes e o mediador MLaw. O módulo Agente contém as classes que os desenvolvedores de agentes devem utilizar para implementá-los. Este módulo por sua vez, utiliza o módulo de Comunicação. O módulo de Interpretação é utilizado para mapear elementos de lei descritos usando a linguagem XMLaw para o modelo de execução interno ao MLaw. O modulo de Evento é responsável pela notificação e propagação de eventos tão importantes para a realização do modelo

de eventos previsto no modelo conceitual de XMLaw. O módulo de Componente define um conjunto de classes abstratas e concretas, além de interfaces, que permitem que novas funcionalidades sejam inseridas. De uma forma geral, este módulo contém os componentes que implementam o comportamento dos elementos da linguagem XMLaw.

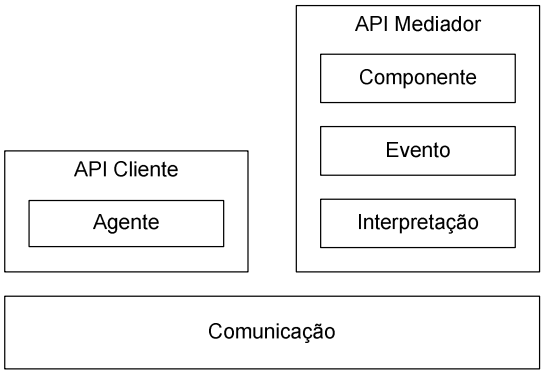

Figura 4 – Componentes do MLaw

Um agente cliente de nossa solução utiliza os módulos de Agente e de Comunicação. A máquina de interpretação de leis, chamada de Mediador, engloba os outros três módulos, Interpretador, Evento e Componente, e também utiliza o módulo de Comunicação. Os módulos de Interpretador, Evento e Componente não são visíveis para os desenvolvedores de agentes, mas podem ser estendidos para evoluir as funcionalidades do Mediador.

Elementos como cenas, relógios e normas são implementados e associados ao módulo componente. O módulo Componente contém um conjunto de classes que o mediador implementa para representar o comportamento dos elementos da linguagem XMLaw. Por exemplo, a Figura 5 detalha o elemento Scene (cena) em XMLaw e o conjunto de classes que implementam o seu comportamento. Na Figura 5, o elemento Scene em XMLaw é mapeado para hierarquias distintas, porém relacionadas, de elementos de descrição (*IDescriptor*) e elementos de execução (*IExecution*). Estas hierarquias são pontos de alteração previstos no módulo Componente, e elas são usadas para a proposição de novos elementos na definição da lei, permitindo, por exemplo, que o modelo conceitual de XMLaw evolua com mais facilidade.

As classes e interfaces principais do módulo de Componente e Interpretador fornecem a estrutura que deve ser implementada pelos elementos da linguagem. Esta estrutura está resumida adiante:

Handlers (SimpleHandler e ComposedHandler) – O módulo Interpretador tem um interpretador XML padrão que lê uma especificação de lei em XML e delega o tratamento de cada tag a um tratador específico. Este tratador é responsável por receber os tokens identificados pelo interpretador e deve construir os descritores de elementos.

IDescriptor – Ele representa o modelo de objeto de uma tag XML. Por exemplo, na Figura 5, a tag scene do XML é representada pela classe SceneDescriptor. A sua maior responsabilidade é criar instâncias de elementos de execução (*IExecution*) a partir do descritor, através do método createExecution(). Esta interface é definida no módulo Componente.

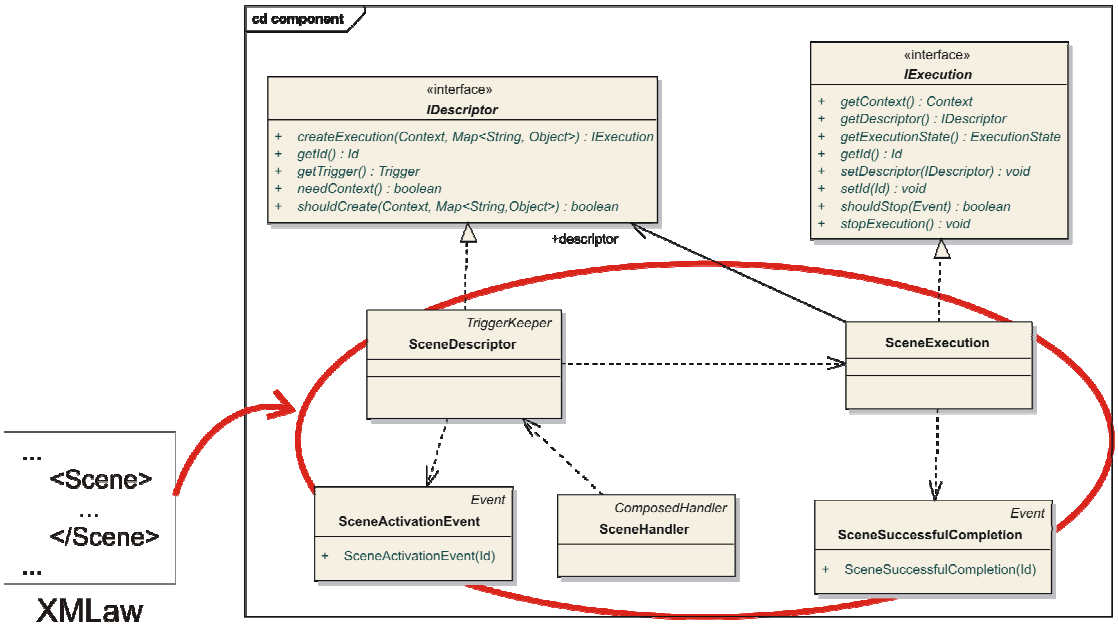

Figura 5 – Projeto do Elemento Scene

IExecution – Um objeto que implementa a interface IExecution é uma instância de um elemento representado pelo objeto IDescriptor. Por exemplo, uma cena pode ser instanciada muitas vezes e podem existir diferentes instâncias executando ao mesmo tempo (vários leilões executando em paralelo, por exemplo). Cada instância (IExecution) deve manter os seus atributos e controlar o seu ciclo de vida. A interface IExecution define todas as operações de *callback* necessárias pelo módulo Componente para controlar as suas instâncias.

Desenvolvedores de agentes podem querer estender a classe Agent oferecida pela API cliente MLaw. Esta classe fornece métodos para enviar e receber mensagens e métodos para a comunicação direta com o mediador. O mediador pode fornecer informações úteis sobre o estado corrente de uma interação, por exemplo: as cenas que estão executando, quantos agentes estão participando

destas cenas, dentre outras. Na verdade, a classe LawFacade fornece métodos para a comunicação direta com o mediador, e a classe Agent fornece métodos para o envio e recebimento de mensagens. A Figura 6 apresenta estas classes.

Os desenvolvedores de agentes são encorajados a utilizar a classe Agent por herança ou por delegação. Entretanto, é possível ainda que estes desenvolvedores construam os seus próprios agentes utilizando as tecnologias ou arquiteturas de sua preferência. O único requisito que um agente deve satisfazer é como se comunicar utilizando mensagens FIPA-Agent Communication Language [FIPA, 2002] com o mediador MLaw. A implementação padrão de MLaw utiliza Jade [Bellifemine et Poggi, 2001] como ambiente de comunicação e desenvolvimento básico de agentes de software.

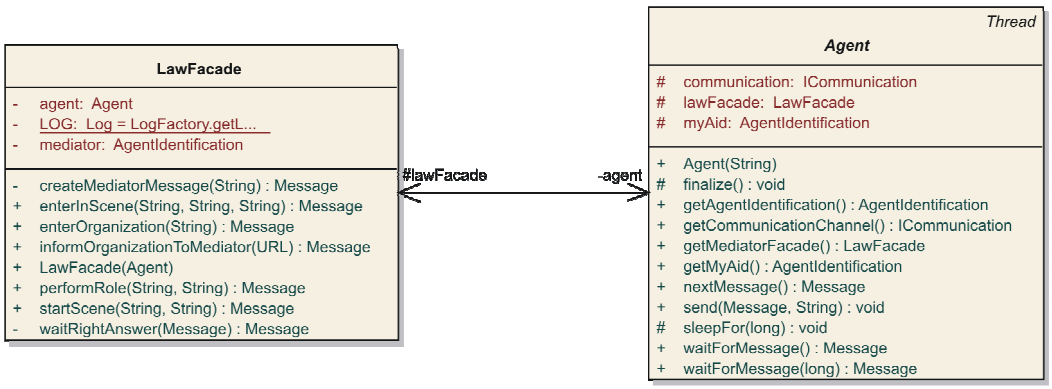

Figura 6 – API Cliente: Classes LawFacade e Agent

2.3.1. Ciclo de Vida do Mediador MLaw

Esta seção tem como objetivo explicar as alterações necessárias no mediador MLaw para interpretar leis de interação com pontos de extensão de interação. Conforme explicado, o mediador MLaw é a estrutura de software utilizada para interpretar as leis de interação e monitorar a conformidade dos agentes com o comportamento desejado para o sistema.

O ciclo de vida previsto para o mediador pode ser consultado na figura (Figura 7). Durante a fase de configuração do mediador no estado OCIOSO, ele recebe as leis de interação que vigorarão a partir do início do monitoramento. Esta lei de interação é interpretada pelo componente Interpretador. Este interpretador verifica se a implementação da lei XMLaw está bem formada e consistente. É responsabilidade do interpretador mapear os elementos descritos na lei, para uma

estrutura própria do modelo de execução que será utilizado em tempo de execução, e este processo não admite má-formação dos elementos.

Com a proposta de pontos de extensão de interação, este processo ocorrerá em dois passos (INTERPRETANDO e ESTENDENDO). Primeiramente, a lei base será verificada e sua estrutura de execução será criada. Caso ela esteja com alguma má formação, o mediador irá para o estado LEI BASE INCONSISTENTE. Depois de fornecer um conjunto de leis bem formadas, as extensões propostas serão interpretadas e todos os pontos de extensão de interação devem ser refinados e materializados em elementos concretos.

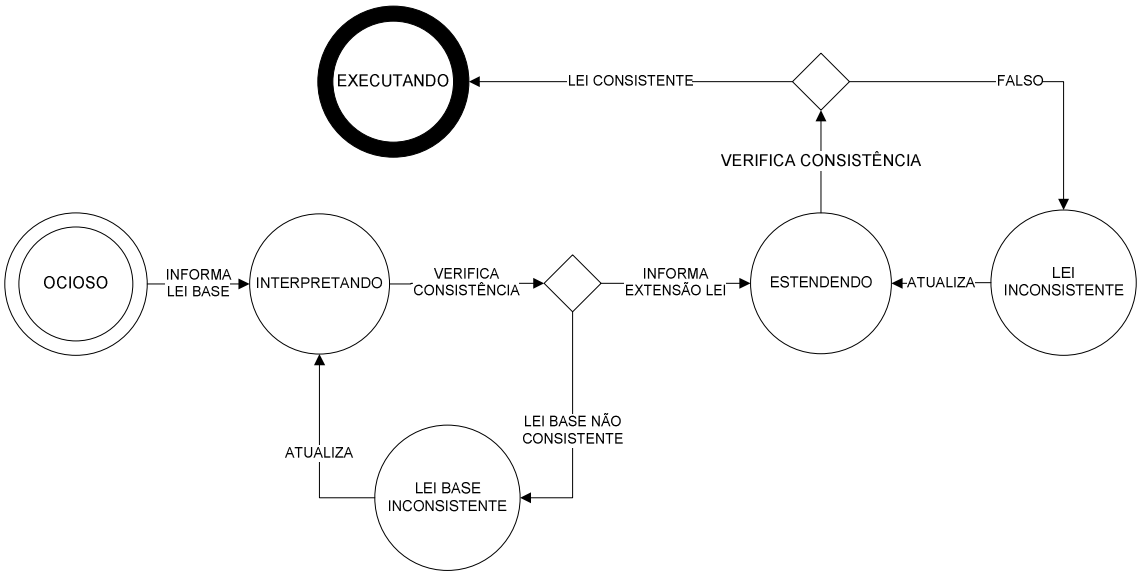

Figura 7 – Ciclo de Vida do Mediador MLaw

Estes dois passos permitem a verificação de algumas regras de construção. Por exemplo, se a lei é definida como concreta, ela não pode deixar nenhum ponto de extensão em aberto, isto é, a lei deve estar completamente definida. Todos os elementos precisam estar implementados, caso contrário o interpretador indicará um erro e irá para o estado LEI INCONSISTENTE. Caso a lei esteja inconsistente, será solicitada uma nova extensão. No final do processo, com a lei consistente o mediador é capaz de executar e mediar a conversação dos agentes (estado EXECUTANDO).

Para a realização deste ciclo de vida, o mediador MLaw foi projetado de forma a separar classes de definição e descrição de leis (Descriptor) e classes de execução e controle do estado corrente da lei (Execution). Esta separação já foi explicada na Figura 5.

A partir da apresentação da linguagem XMLaw e do mediador MLaw, nos falta explicar como requisitos são documentados e como esta documentação pode auxiliar na proposição de leis de interação. Adiante é apresentada a proposta de Casos de Leis para apoiar esta atividade.

2.4. Casos de Leis

O principal requisito a ser abordado com a técnica de Casos de Leis é a documentação de decisões de análise (racional) derivadas a partir da especificação de requisitos para sistemas multiagentes abertos. Esta documentação deve apresentar claramente o mapeamento e o registro de requisitos [Gatti et al., 2006b], que futuramente serão mapeados para infra-estruturas capazes de verificar em tempo de execução se os requisitos, ou melhor, se as regras são obedecidas.

Procura-se prover com Casos de Leis uma maneira explícita de registrar as decisões (o chamado rastro de *rationale*) para derivar requisitos, que por sua vez, vão derivar as leis. Mais especificamente, a documentação iterativa de requisitos deve ser feita de forma a preservar o conjunto de decisões que foram tomadas, as variabilidades identificadas e as razões que determinaram a escolha de requisitos ou de sua implementação. O registro das variabilidades pode justificar e documentar as decisões e necessidades de reutilização da solução desenvolvida. Este item em específico será discutido em mais detalhes no próximo capítulo. Neste capítulo, será simplificada a proposta de [Gatti, 2006b] a partir da experiência prática do uso desta abordagem.

2.4.1. Estrutura de Documentação de Casos de Leis

Um Caso de Leis é um documento que fornece evidências estruturadas na forma de um argumento válido e convincente que demonstra que um SMA aberto exibe todos os atributos requeridos. Isto será feito para uma dada aplicação num dado ambiente através do *rationale* da derivação de elementos de leis.

Casos de Leis associam a aplicação de casos de uso [Booch & Rumbaugh, 1999] e casos de fidedignidade [Weinstock et al. 2004]. O primeiro é útil para mantermos uma visão geral sobre os requisitos que são identificados e o segundo é útil na documentação das decisões que são tomadas e o *rationale* em torno destas decisões. Parte da decisão está diretamente relacionada à criação de leis de

interação, ou então elementos de leis, que serão verificadas e aplicadas em tempo de execução.

A estrutura proposta para a documentação visa correlacionar os diagramas de casos de leis, com diagramas de requisitos funcionais do sistema, expressos utilizando casos de uso [Booch & Rumbaugh, 1999] ou cenários descritos textualmente [Sommerville, 2004].

Figura 8 – Relacionamento de Casos de Leis com Casos de Uso

Casos de Leis são uma técnica complementar ao processo de derivação de leis de interação para a governança de sistemas multiagentes abertos. Resume-se a uma forma de documentação e modelagem baseada no modelo conceitual adaptado da proposta original de Casos de Fidedignidade [Weinstock et al. 2004]. A partir de alguns estudos de caso, foi possível perceber que a aplicação desta estrutura no contexto de sistemas multiagentes consegue representar, em diferentes níveis de detalhe, as decisões e os requisitos derivados a partir do entendimento do problema.

Um Caso de Fidedignidade [Weinstock et al. 2004] é uma documentação de evidências que fornece um argumento válido e convincente quanto à aderência da solução à realidade de aplicação da solução em desenvolvimento. Esta abordagem já foi aplicada em diferentes contextos [Alexander, 2003; Firesmith, 2003]. Em nosso contexto, a técnica proposta se baseará em uma estrutura ou cadeia de raciocínio que visa levantar evidências e argumentos de que um dado sistema possui todos os atributos requeridos e necessários para uma determinada aplicação. Quanto melhor for a argumentação e as evidências durante o raciocínio, maior a chance de a hipótese ser verdadeira e, conseqüentemente, mais convincente é sua especificação gerada. No processo de argumentação, é necessário pensar numa estratégia que auxilie o entendimento da hipótese identificada. E, por sua vez, ao especificar uma estratégia talvez seja necessário especificar sub-hipóteses, até que a argumentação seja suficiente para o analista que a desenvolve.

Considerando-se o processo acima e adaptando-o [Gatti,2006b], quatro elementos conceituais são propostos e adaptados para o contexto de caso de leis (Figura 9): contexto, hipótese, argumento e evidência. A partir desta proposta, um

modelo conceitual de rastro do *rationale* adaptado foi também proposto. Como foi discutido, o ponto principal da proposta de casos de fidedignidade está na estrutura dos argumentos e nas evidências que apóiam e corroboram para o entendimento/atendimento desta hipótese. Estes documentos devem ter como principal objetivo convencer um leitor da sua validade perante as necessidades identificadas. Desta forma, este é considerado um elemento chave de nossa abordagem de governança de leis, sendo responsável por registrar todas as hipóteses, contextos e argumentos adotados no processo de derivação de leis de interação.

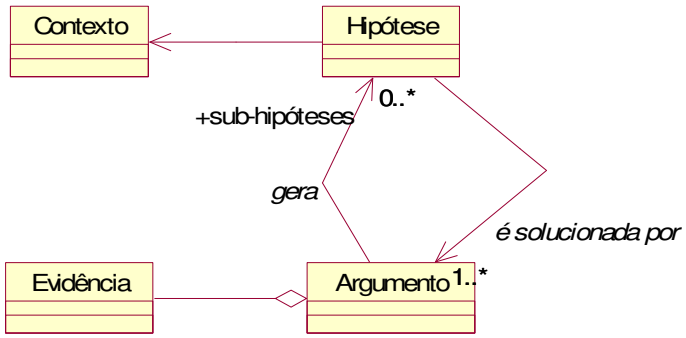

Figura 9 – Modelo Conceitual de Casos de Leis

O modelo conceitual apresentado na Figura 9 é derivado da aplicação da proposta original de casos de fidedignidade a problemas de governança de leis. O ponto principal que merece destaque neste modelo é a possibilidade de descrever o conhecimento parcial sobre o sistema em desenvolvimento. Isto é, uma hipótese baseia-se em contextos (detalhes ou informações sobre o ambiente analisado) e é solucionada por um conjunto de argumentos que são justificados por evidências. O contexto nada mais é do que informações que especializam o problema e definem melhor o escopo de análise. Por sua vez, um argumento pode identificar novos contextos que derivam novas hipóteses que necessitam de mais argumentos para o completo entendimento da solução. Por outro lado, a partir de um argumento é possível comprovar a hipótese através de evidências por meio de informações e dados coletados experimentalmente ou ainda por aplicações que possuam o registro de um processo de verificação de corretude, ou mesmo log de execução.

Em um sistema real, com várias necessidades e propriedades que precisam ser satisfeitas, é natural que a quantidade de informações derivadas de um processo de análise cresça com o desenvolvimento da aplicação, isto implicará um

número considerável de hipóteses e argumentos. Para facilitar o entendimento da documentação por parte de um leitor, é proposta uma notação gráfica para a representação de elementos (Figura 10). Pequenas adaptações para a notação proposta em [Weinstock et al, 2004] foram definidas, em prol de adequá-las ao português.

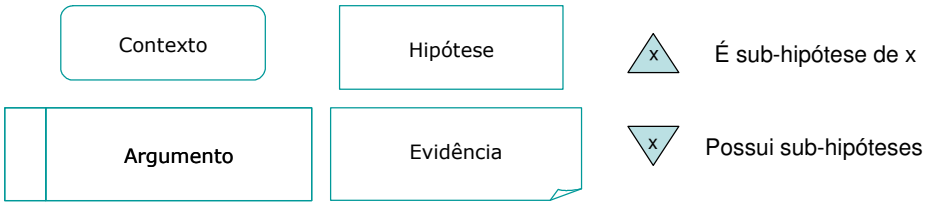

Figura 10 – Notação Gráfica para Diagrama de Casos de Leis

Este diagrama deve estar descrito de forma a relacionar o diagrama de casos de leis com documentos de caso de uso ou cenários do sistema. Mais do que isto, definir claramente a ameaça que está sendo mitigada com as leis de interação em um sistema multiagente aberto é importante. O conceito de ameaça foi adaptado da literatura de análise de risco [Fairley, 2002] e teve sua primeira aplicação em nosso contexto para a documentação de requisitos de leis em [Carvalho et al., 2005]. Desta experiência, iremos explorar a noção de ameaça.

A estrutura proposta para a documentação de um caso de lei prevê o campo textual **Casos de Leis** que contém o nome que identifica o caso de lei em questão, o campo textual **Autor** que identifica o nome do autor, o campo textual **Data** que registra a última data da elaboração deste documento, o campo textual **Caso de Uso** que referencia o caso de uso relacionado com o caso de lei, o campo textual **Ameaça** que prevê a descrição da ameaça que estamos mitigando com a abordagem de leis, o campo textual **Pré-condições** que enumera as pré-condições de ativação, o campo **Detalhamento** onde estará detalhado o diagrama de casos de leis, e por fim o campo textual **Pós-condições** contendo uma enumeração com as pós-condições previstas para após a realização deste caso de lei. A estrutura é apresentada de forma esquemática na Figura 11.

Caso de Leis: <nome_do_caso_de_leis>	
Autor: <nome_do_autor>	**Data**: <data_da_elaboração>
Caso de Uso: <nome_do_caso_de_uso_referenciado_por_este_caso_de_leis>	
Ameaça: <ameaça_a_ser_mitigada>	
Pré-condições: <pré-condições_de_ativação>	
Detalhamento <diagrama de casos de leis>	
Pós-condições: <pós-condição_se_ativado>	

Figura 11 – Estrutura de Descrição de um Caso de Lei

2.4.2. Desenvolvendo Casos de Leis

Para desenvolver Casos de Leis, é necessário desenvolver os Casos de Uso e gerar o documento de requisitos funcionais e não funcionais do sistema (na forma de cenários, por exemplo). Feito isso, faz-se necessário levantar as ameaças do sistema através dos requisitos e refinar os Casos de Leis iterativamente, baseando-se sempre nos documentos levantados. Após ter gerado os Casos de Leis, deve ser feita uma análise dos mesmos para gerar os requisitos de leis que, por sua vez, podem ser mapeados para a linguagem de especificação XMLaw. Na Figura 12, é apresentado de forma abstrata o processo de análise e projeto de leis utilizando Casos de Leis e as responsabilidades envolvidas.

Figura 12 – Processo de Análise de Leis (ênfase em Requisitos)

Utilizando um processo de análise e projeto de leis de forma iterativa, é possível refinar não somente as ameaças do sistema, como os requisitos de leis que atenuam tais ameaças. Casos de Leis são derivados a partir de hipóteses sobre um sistema e da descrição de que existem evidências que comprovem que as hipóteses são verdadeiras. A Figura 13 descreve o processo de desenvolvimento do *rationale* de um Caso de Leis utilizando o modelo conceitual adaptado. Esta estrutura permite uma maior compreensão e um melhor detalhamento da solução.

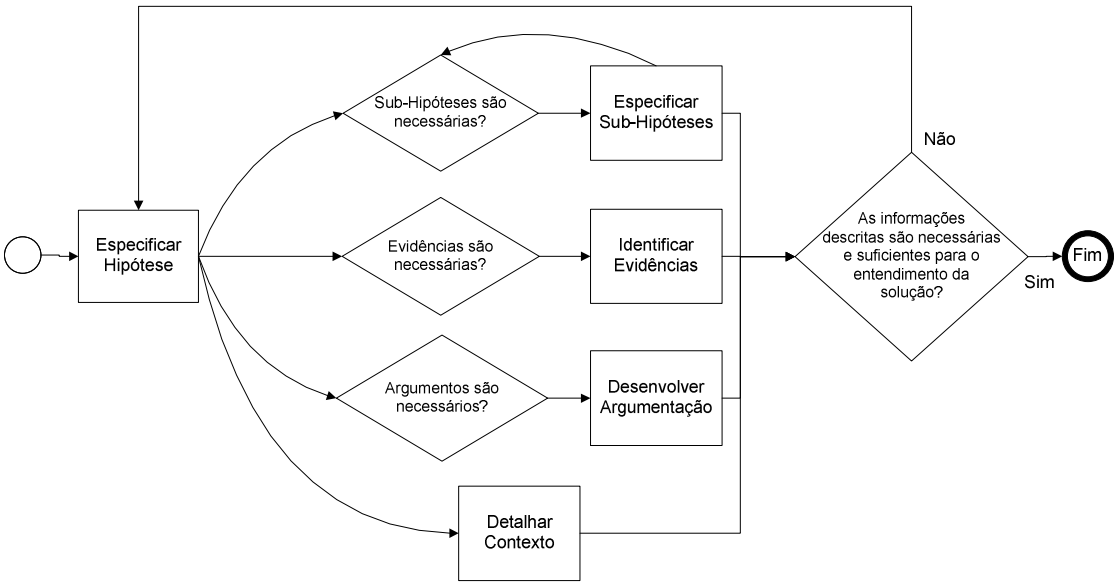

Figura 13 – Desenvolvendo o rationale de um Caso de Leis

Com o objetivo de exemplificar de que forma o *rationale* de um caso de leis é documentado, ilustramos na Figura 14 a especificação da sub-hipótese "o agente fornecedor de peça não pode falhar". Neste Caso de Leis, basicamente, deseja-se impedir que um fornecedor deixe de efetuar uma entrega.

Como foi mencionado anteriormente, para desenvolver Casos de Leis, é necessário utilizar uma abordagem tradicional de documentação de requisitos (por exemplo, Casos de Uso) para gerar um documento de requisitos funcionais do sistema que, por sua vez, vão gerar os bens e serviços disponíveis pelo sistema. Feito isso, a partir dos requisitos não funcionais, o desenvolvedor vai levantar as ameaças para Caso de Uso, quando houver, e vai desenvolver os Casos de Leis iterativamente, baseando-se sempre nos requisitos funcionais, não funcionais e ameaças do sistema. Após a geração dos Casos de Leis, deve ser feita uma análise para gerar os requisitos de leis que, por sua vez, podem ser diretamente mapeados para a linguagem de especificação XMLaw. Outro exemplo de uso de Casos de Leis está transcrito na Figura 14.

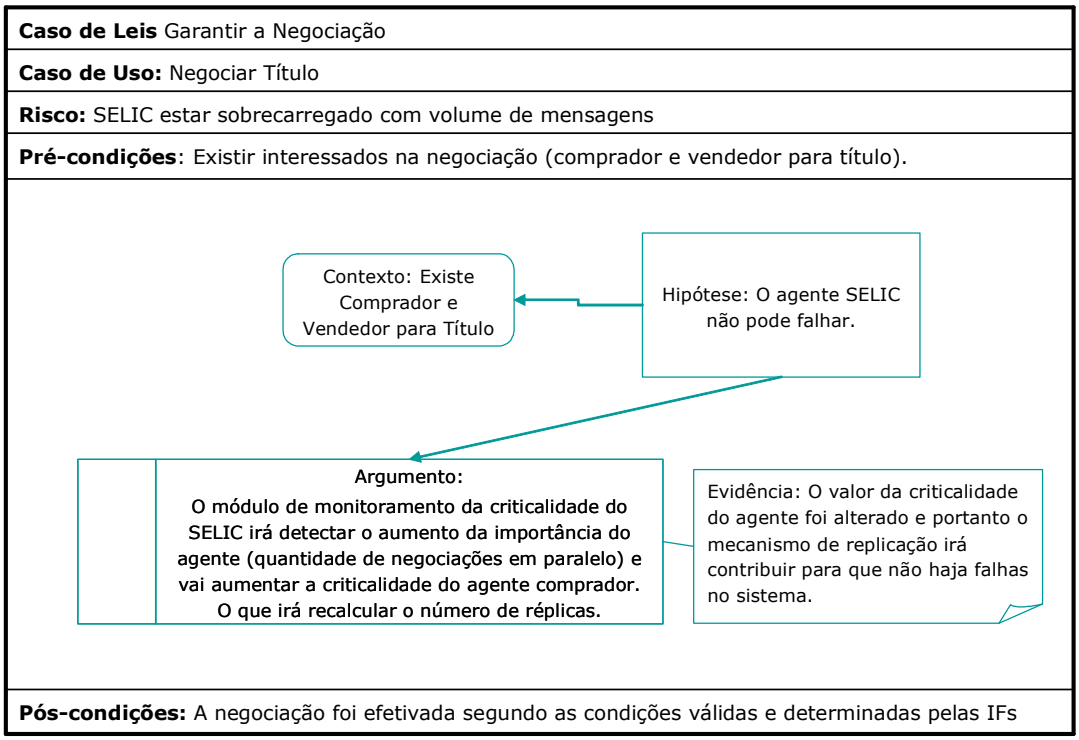

Figura 14 – Exemplo de Caso de Lei

2.5. Conclusão

Neste capítulo, foram apresentadas quatro técnicas desenvolvidas ao longo da tese para apoio a governança de sistemas multiagentes abertos regulados por leis: o Modelo Conceitual, a linguagem declarativa XMLaw, o middleware MLaw e Casos de Leis.

Um modelo conceitual, com abstrações de alto nível, foi proposto para representar leis de interação descritas por uma linguagem associada ao modelo, chamada de XMLaw. Maiores detalhes sobre as melhorias propostas a esta linguagem serão apresentadas adiante. Um middleware foi proposto para implementar o mecanismo de governança instruído a partir da linguagem de especificação de leis XMLaw. O projeto e a implementação deste middleware foram aprimorados para incluir um suporte a interpretação e especialização de leis de interação para dar suporte à técnica de g-frameworks. Por fim, Casos de Leis foram propostos para estruturar e facilitar a descrição de requisitos que pudessem ser mapeados para leis de interação. Neste trabalho utilizaremos esta abordagem com um propósito próprio de apoiar a registrar o racional em decisões de projeto de leis de interação flexíveis e reutilizáveis.