4
Modelo Conceitual

A abordagem de leis é utilizada para construir sistemas multi-agentes abertos, onde um certo grau de controle sobre o comportamento dos agentes é esperado. Agentes possuem propriedades tais como autonomia [25], racionalidade [52], adaptabilidade [27], aprendizado [26], habilidade social [1] [53] e mobilidade [54]. Destas propriedades, pelo menos duas são consenso: autonomia e habilidade social.

Autonomia é a habilidade do agente para (i) agir sem intervenção externa, (ii) dizer sim ou não em resposta a uma requisição de outro agente e (iii) iniciar uma ação por motivações próprias [55]. A abordagem de leis afeta a autonomia das interações dos agentes, pois elas são reguladas por leis. Desta forma, o item (i) é afetado porque o agente sofre a intervenção das leis, e (ii) e (iii) também só ocorrem se estiverem em conformidade com as especificações das leis. Neste sentido, a abordagem deve ser entendida como uma forma de se estabelecer um limite para o grau de autonomia que um agente tem ao interagir com outros agentes.

Neste capítulo, propõe-se um conjunto de conceitos para a especificação da forma como as interações são reguladas em um sistema multi-agente. A Figura 4.1 ilustra estes conceitos. No restante desse capítulo, os conceitos são apresentados e suas representações são ilustradas usando-se XMLaw. XMLaw é uma linguagem declarativa para a especificação de leis de interação em um sistema multi-agente.

4.1
Eventos: relacionando os conceitos

O relacionamento entre os elementos que compõem o modelo de leis é feito dinamicamente por meio de eventos. Os elementos são capazes de gerar e perceber a ocorrência de eventos. É possível interpretar esse relacionamento via eventos como uma cadeia de causas e conseqüências, onde um evento

[1] Agentes interagem com outros agentes.

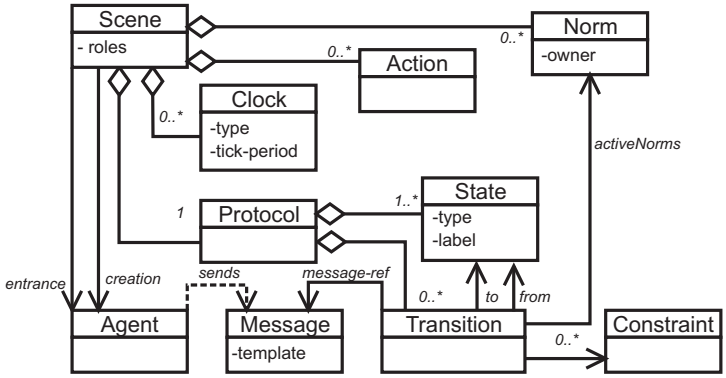

Figura 4.1: Modelo Conceitual

pode ativar um elemento; esse elemento pode gerar outros eventos e assim por diante. Por exemplo, o envio de uma mensagem por um agente gera um evento (*message arrival*), este evento pode ativar uma transição (*transition activation*), que por sua vez ativa um relógio (*clock activation*), que gera um outro evento (*clock_tick*), e finalmente ativa uma norma (*norm activation*) (Figura 4.2).

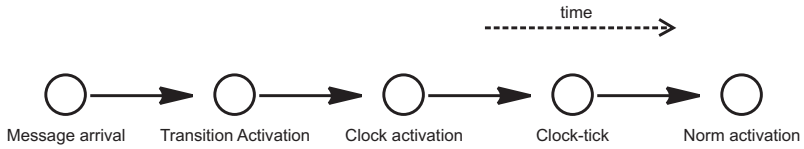

Figura 4.2: Cadeia de Eventos

4.2 Agente e Mensagem

As leis estabelecem uma regulamentação que especifica o comportamento desejado dos agentes. Agentes, por sua vez, podem ser definidos como entidades computacionais autônomas capazes de resolver problemas em ambientes dinâmicos e abertos [56]. Em sistemas multi-agentes abertos, os aspectos internos dos agentes são inacessíveis, a única informação acessível sobre os agentes é o seu comportamento observável por meio das mensagens que eles emitem ao se comunicarem com os outros agentes. Baseado nisto, um agente no modelo conceitual é qualquer entidade computacional distribuída capaz de se comunicar com outros agentes através da troca de mensagens. No modelo conceitual representa-se um agente pelo elemento *Agent* e uma mensagem é representada pelo elemento *Message* (Figura 4.3).

O elemento *Message* permite especificar quais são as mensagens que os agentes podem trocar. Este elemento possui um atributo denominado *template* que é utilizado para especificar o padrão, ou formato das mensagens permitidas.

Figura 4.3: Modelo Conceitual: Agente e Mensagem

Este atributo é composto de um conjunto de campos que toda mensagem precisa ter, são eles:

- *performative* - o campo *performative* serve para especificar a intenção dos agentes em uma interação com outros agentes. Os valores possíveis para este campo não está restrito a nenhum conjunto de valores específicos. A comunicação entre os agentes é geralmente baseada na teoria de atos da fala [29]. Um grande número de linguagens de comunicação entre os agentes se baseiam nessa teoria [57, 58]. Elas fornecem um vocabulário que permite aos agentes expressar suas intenções. O campo *performative* permite a especificação destas intenções.
- *SenderName* e *SenderRole* - estes campos especificam respectivamente o agente que está enviando a mensagem e o papel que ele está desempenhando na comunicação.
- *ReceiverName* e *ReceiverRole* - permite especificar o agente e o papel do receptor da mensagem, respectivamente.
- *content* - este campo contém a informação que está sendo trocada entre os agentes. A complexidade das informações e a forma como elas estão representadas varia de acordo com a aplicação. Por exemplo, para algumas aplicações os agentes podem representar e estruturar as informações utilizando a linguagem Prolog [31], em outras o uso de DAML+OIL [59] pode ser mais indicado. Este campo não especifica qual o formato de representação do conteúdo.

A estrutura do elemento *Message* é ilustrado na Tabela 4.1. Através deste elemento, é possível por exemplo, especificar que a única mensagem válida em um determinado ponto da conversação entre os agentes é aquela onde a performativa é igual a *request*, o papel do agente remetente é igual a *Customer* e o papel do agente receptor é igual a *Seller* (Tabela 4.2).

```
<Message id="aMessageId"
         template="message(performative,
                  sender(SenderName,SenderRole),
                  receiver(ReceiverName,ReceiverRole),
                  content(SomeContent)).">
```

Tabela 4.1: XMLaw da messagem

```
template="message(request,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣sender(Any,Customer),
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣receiver(Any,Seller),
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣content(Any))."/>
```

Tabela 4.2: Exemplo do Atributo *template*

Entre os campos do atributo *template* estão o *SenderRole* e o *ReceiverRole*. Estes campos estão relacionados com o elemento *Role* definido no modelo conceitual. Este elemento representa como um agente é visto em um nível organizacional, ou seja, qual o seu papel na organização. Ao interagir, um agente precisa escolher um papel ao qual ele se associará, embora possa estar associado a vários papéis simultaneamente.

4.3 Protocolo, Estado e Transição

As mensagens isoladamente não permitem especificar muita coisa em uma interação. É preciso estabelecer a ordem em que elas ocorrem e quais as mensagens válidas em um determinado momento da interação. Protocolos de interação representam justamente padrões de interação entre os 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 [36], onde estados representam pontos na execução do protocolo e transições são as conexões entre os estados.

Além de estados e transições, autômatos também possuem um conjunto finito de símbolos que são capazes de processar. No contexto dos protocolos, este conjunto de símbolos é representado por elementos *Message*. Ou seja, as transições entre os estados são ativadas por mensagens trocadas entre os agentes. Na Figura 4.4, destaca-se o elemento *Protocol* do modelo conceitual.

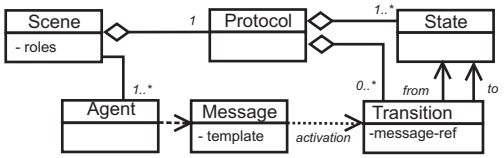

Figura 4.4: Modelo Conceitual: Foco no Protocolo

Protocolos são representados especificando seu conjunto de estados e transições utilizando-se XMLaw conforme ilustrado na Tabela 4.3.

```
<Protocol>
  <States> ... </States>
  <Transitions> ... </Transitions>
</Protocol>
```

Tabela 4.3: XMLaw do Protocolo

Os estados de um protocolo indicam pontos na execução do protocolo e, portanto, podem ter diferentes significados. Entretanto, é possível identificar algumas categorias de estados: *initial*, *execution*, *success* ou *failure*. Um estado classificado como *initial* representa o estado inicial do protocolo de interação e somente um estado inicial é permitido por protocolo. Um estado *execution* significa que a execução do protocolo ainda não terminou. Estados *success* ou *failure* são estados finais e representam, respectivamente, que o protocolo foi concluído com sucesso ou com falha. Estas categorias de estados são representadas na linguagem XMLaw através do atributo *type*, como ilustrado na Tabela 4.4.

```
<State id="anStateId"
       type="success"
       label="A␣state␣label"/>
```

Tabela 4.4: XMLaw do Estado

As transições (*Transition*) conectam dois estados. Esta conexão é feita no XMLaw através dos atributos *from* e *to*, que mantêm referências para estados do protocolo. Porém, transições somente são ativadas mediante à ocorrência de uma mensagem representada pelo atributo *message-ref*. Este atributo é uma referência a uma mensagem especificada no XMLaw (elemento *Message*). Na Tabela 4.5 ilustra-se a representação XMLaw de uma transição.

```
<Transition id="anId"
            from="anStateId"
            to="anotherStateId"
            message-ref="aMessageId"/>
```

Tabela 4.5: XMLaw da Transição

O disparo de uma transição, além de ser condicionado à ocorrência de uma mensagem, pode ser também condicionado a um conjunto de normas que precisam estar ativas e a um conjunto de restrições[2] (*Constraints*). Em outras palavras, uma transição pode requerer que uma determinada norma esteja ativa para que ela possa disparar. Este conjunto de normas é representado pelo atributo *activeNorms* do elemento *Transition*. XMLaw representa este conjunto de normas através da tag XML <*ActiveNorms*>, ilustrada na Tabela 4.6.

```
<Transition id="anId"
            from="anStateId"
```

[2]Restrições são explicadas na Seção 4.9

```
            to="anotherStateId"
            message-ref="aMessageId">
  <ActiveNorms>
    <Norm ref="aNormId"/>
  </ActiveNorms>
</Transition>
```

Tabela 4.6: XMLaw do Atributo *activeNorms*

4.4 Relógio

Aspectos temporais de um sistema podem ser bastante relevantes em muitas classes de sistemas tais como sistemas de tempo real, simuladores, comércio eletrônico e sistemas interativos. Muitas outras abordagens baseadas em leis [45, 12, 13] não consideram a influência do tempo para regular as interações dos agentes. Neste trabalho, argumenta-se que as leis devem ser sensíveis ao tempo. Em outras palavras, se um agente é capaz de realizar uma ação a_1 no tempo t_1, ele pode ser proibido de executar a mesma ação se o tempo for t_2 ($t_1 < t_2$).

O modelo conceitual fornece um elemento denominado *Clock* (Relógio) que provê 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 ser ativado por transições ou mesmo por normas.

O ciclo de vida de um relógio é ilustrado na Figura 4.5. Uma vez ativado, o relógio entra no estado *ativado*. É possível haver múltiplas instâncias de um mesmo relógio executando simultaneamente e independentemente. Após esgotado o período de tempo especificado, o relógio gera eventos de *tick*. O relógio entra no estado desativado em duas situações: alguma outra condição na lei o desativa, como por exemplo a ocorrência de uma determinada transição; ou quando não há mais eventos de *tick* para gerar. A especificação de se há ou não mais eventos de *tick* para gerar é dada pelo tipo de relógio, que pode ser *regular* ou *periodic*. O primeiro significa que o relógio gerará apenas um *tick* e entrará no estado *desativado* e o segundo significa que o relógio gerará um *tick* toda vez que o intervalo de tempo especificado se esgotar.

A representação de um relógio utilizando-se XMLaw é ilustrada na Tabela 4.7. Os elementos <*Activations*> e <*Deactivations*> representam as condições de ativação e desativação de um relógio. Estas condições podem ser quaisquer eventos gerados pelos elementos do modelo conceitual de leis. Na Seção 5.3.1

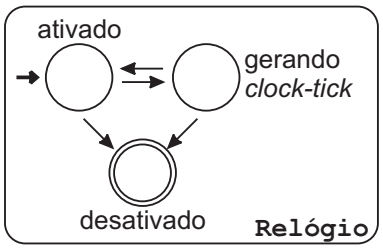

Figura 4.5: Ciclo de Vida do Relógio

apresenta-se uma lista detalhada de todos os eventos que podem ser gerados, e quem são os elementos que o geram.

```
<Clock id="aClockId"
       type="regular"
       tick-period="1000">
  <Activations>
    <Element ref="anId"
             event-type="anEventType"/>
  </Activations>
  <Deactivations>
    <Element ref="otherId"
             event-type="anEventType"/>
  </Deactivations>
</Clock>
```

Tabela 4.7: XMLaw do Relógio

4.5 Norma

Normas (*Norms*) descrevem quais comportamentos dos agentes são permitidos, quais são obrigados e quais são proibidos. Em outras palavras, elas descrevem as permissões, obrigações e proibições dos agentes. As normas são geralmente adquiridas pelos agentes durante o 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.

Normas contêm um conjunto de condições de ativação e desativação. Essas condições especificam quando uma norma deve ser ativada e quando ela deve ser desativada e, assim como o *Clock*, este processo de ativação e desativação também é baseado em eventos. Além disso, normas possuem um atributo chamado *owner* que representa o "possuidor da norma". Quando uma norma é ativada, ela é válida para um agente ou papel de agente específico.

```
<Norms>
 <Permission id="aPermissionId">
```

```
    <Owner>AgentOrRole</Owner>
    <Activations>
        <Element ref="anId"
                 event-type="anEventType"/>
    </Activations>
    <Deactivations>
      <Element ref="anotherId"
               event-type="anEventType"/>
    </Deactivations>
 </Permission>
 <Obligation
    id="anObligationId"> ... </Obligation>
 <Prohibition
    id="aProhibitionId"> ... </Prohibition>
</Norms>
```

Tabela 4.8: XMLaw de *Normas*

4.6 Cena

Sistemas multi-agentes podem ter muitas interações complexas ocorrendo simultaneamente. Sistematizar essas interações é necessário para um melhor entendimento e gerenciamento desses sistemas. O modelo conceitual utiliza a abstração de cenas para auxiliar na organização e modularização das interações. A idéia de cenas é análoga a uma cena de peça de teatro. Em peças de teatro os atores atuam de acordo com scripts bem definidos e a peça é composta de várias cenas conectadas seqüencialmente. Esta analogia foi introduzida por Esteva [9] e é utilizada nesse trabalho com algumas modificações.

Cenas são representadas pelo elemento do modelo conceitual *Scene*. Este elemento especifica quais agentes e quais os papéis de agentes 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 (*Clock*). 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. Na Tabela 4.9 ilustra-se a estrutura de uma cena representada em XMLaw.

```
<Scene id="negotiation" time-to-live="infinity">
 <Creators>
   <Creator agent="any" role="any"/>
 </Creators>
 <Entrance>
   <Participant agent="any" role="anAgentRole">
     <StatesRef>
       <StateRef ref="aState"/>
     </StatesRef>
   </Participant>
```

```
        ...
  </Entrance>
  <Messages>
        <!-- All possible messages patterns -->
        ...
  </Messages>
  <Protocol>
        <!-The interaction protocol for this scene -->
        ...
  </Protocol>
  <Clocks>
      <!-All the clocks for this scene -->
      ...
  </Clocks>
  <Norms>
        <!-- All the norms for this scene -->
      ...
  </Norms>
  <Actions>
        <!-- All the actions for this scene -->
      ...
  </Actions>
</Scene>
```

Tabela 4.9: XMLaw da *Scene*

Em muitos casos é necessária a existência de muitas instâncias de uma mesma cena executando concorrentemente, por exemplo, especifica-se uma cena para um processo de leilão virtual e vários leilões estão ocorrendo ao mesmo tempo. Por essa razão, podem existir várias instâncias de cena executando simultaneamente, o que reflete no seu ciclo de vida. Cada instância de uma cena possui um ciclo de vida composto de três estados: *initialization*, *running* and *finalization*.

O estado *initialization* ocorre logo após a criação da instância de uma cena e em seguida move-se para o estado *running*. Conforme ilustrado na Tabela 4.9 através da tag <*Entrance*>, cenas também especificam pontos do protocolo de interação onde agentes podem entrar na cena. Um agente só pode entrar em uma cena quando ela estiver no estado *running*.

A entrada no estado *finalization* do ciclo de vida de uma cena é detalhada a seguir. De acordo com o modelo conceitual, uma cena é composta por um protocolo de interação e este protocolo de interação é representado por um autômato finito. Autômatos são capazes de reconhecer se uma certa entrada é válida. Quando a entrada tem um tamanho limitado ou ao menos previsível, reconhecer se uma determinada entrada é válida ou não é somente uma questão de executar o autômato. Por exemplo, considere-se um autômato que aceita todas as cadeias de caracteres que começam com o caracter ‘a’. Então, para a entrada “abbbc” o autômato responderá que a entrada é válida, mas para a entrada “baaaa”, o autômato responderá que a entrada é inválida. O importante a

ressaltar neste exemplo é que o autômato "sabe" quando a entrada terminou, ele "sabe" quando não existem mais caracteres para ler e, portanto, é capaz de dizer se a entrada é válida ou não.

Quando se considera o uso de um autômato na verificação da interação de um sistema multi-agente, os caracteres são, de fato, mensagens trocadas pelos agentes e ao contrário do exemplo anterior, o autômato não tem como saber se a entrada terminou, isto é, quando não existem mais mensagem para ler. Para superar este problema, duas decisões foram tomadas:

– não existem transições conectando um estado final a qualquer outro estado - com transições originadas de estados finais, o autômato teria que esperar para que todas as mensagens fossem enviadas para responder se a seqüência é válida ou não. Porém, como os agentes são autônomos, o autômato não tem como saber quando um agente enviará uma mensagem e não poderá responder se o protocolo terminou com sucesso ou falha. Portanto, evitar que transições conectem estados finais a outros estados elimina o problema de saber quando a entrada terminou, pois uma vez que um estado final é alcançado, o protocolo terminou.
– tempo de vida - um tempo de vida é atribuído a cada instância de uma cena. Este tempo é especificado em milisegundos e pode ter o valor especial *infinity*, que significa que a cena nunca expira. Este tempo de vida começa a decrementar a partir do momento da criação da cena e uma vez que 0 é alcançado a cena é movida para o estado *finalization*.

Em suma, uma cena é movida para o estado *finalization* quando o tempo de vida expira ou quando algum estado final é alcançado.

4.7 Navegando entre as Cenas

Cenas descrevem um contexto de interação entre agentes. Estes contextos não existem de maneira isolada, eles geralmente representam etapas de um processo maior. Por exemplo, em um processo de leilão eletrônico poderiam existir uma cena de negociação e uma cena de pagamento. Estas duas cenas acontecem em uma seqüência, ou seja, primeiro os agentes negociam e caso haja sucesso na negociação, deve-se iniciar a cena de pagamento para efetivar a negociação. Além disso, agentes podem estar participando de muitas cenas em paralelo, por exemplo, um mesmo agente pode estar envolvido em duas instâncias de uma cena de negociação e uma outra instância de uma cena de pagamento simultaneamente.

Para atender a estes requisitos, a entrada de um agente em uma cena pode ser condicionada à existência de determinadas normas. Desta forma, normas podem ser atribuídas a agentes dando permissão para a entrada em determinadas cenas. Por exemplo, pode ser concedido ao agente que concluiu uma negociação a permissão para que ele possa iniciar a cena de pagamento.

A criação de cenas ou participação em uma cena condicionada a existência de uma determinada norma, é representada em XMLaw através da tag <*Active-Norms*>, incluída nas tags <*Creator*> e <*Participant*>. A inclusão desta tag é ilustrada na Tabela 4.10, onde a permissão *successful-negotiation* é concedida ao agente desempenhando o papel de *Buyer* da cena de negociação no momento que a cena é concluída com sucesso e a criação da cena de pagamento requer que o agente comprador possua esta permissão.

```
<LawOrganization id="auction" name="Auction Organization"/>
    <Scene id="negotiation" time-to-live="infinity">
        <Creators>
            <Creator agent="any" role="Buyer"/>
        </Creators>
        <Entrance>
            <Participant agent="any" role="Buyer">
                <StatesRef>
                    <StateRef ref="initial"/>
                </StatesRef>
            </Participant>
            <Participant agent="any" role="Seller">
                <StatesRef>
                    <StateRef ref="negotiation-started"/>
                </StatesRef>
            </Participant>
            ...
        </Entrance>
        ...
    </Scene>

    <Norms>
        <Permission id="successful-negotiation">
            <Owner>BuyerAgent</Owner>
            <Activations>
                <Element ref="negotiation"
                    event-type="successful_scene_completion"/>
            </Activations>
            <Deactivations>
                ...
            </Deactivations>
        </Permission>
    </Norms>

    <Scene id="payment" time-to-live="infinity">
        <Creators>
            <Creator agent="any" role="Buyer">
                <ActiveNorms>
                    <Norm ref="successful-negotiation"/>
                </ActiveNorms>
            </Creator>
        </Creators>
        <Entrance>
            ...
        </Entrance>
    </Scene>
</LawOrganization>
```

Tabela 4.10: Restringindo a Navegação entre as Cenas

4.8 LawOrganization

O elemento *LawOrganization* representa e especifica as leis de uma organização multi-agente. Este elemento contém cenas e normas, e é ilustrado na

Tabela 4.11. Este elemento deve ser considerado como o ponto de entrada na especificação, tendo como função principal a conjunção dos demais elementos previstos no modelo.

```
<LawOrganization id="anOrgId"
    name="Organization name"/>
  <Scenes> ... </Scenes>
  <Norms> ... </Norms>
</LawOrganization>
```

Tabela 4.11: XMLaw do LawOrganization

4.9 Restrições

Restrições (*Constraints*) possibilitam a verificação das informações das mensagens. Mensagens transportam informações que são verificadas de várias maneiras. O atributo *message pattern* (Section 5.2.2) verifica a forma da mensagem, descrevendo qual o padrão em que a mensagem é esperada. Entretanto, este atributo não descreve quais são os valores permitidos para determinadas partes da informação. Por exemplo, o *message pattern* para o campo *content* da mensagem poderia ser expresso como: content*(television,brand(AnyBrand),price(Amount))*. Porém, por alguma razão qualquer uma determinada aplicação requer que o valor da variável *Amount* possua um valor entre 50 e 100. Somente com o *message pattern* especificado acima este requisito não pode ser especificado. O elemento *Constraint* permite não só a especificação deste requisito mas também outras restrições relacionadas a valores contidos nas mensagens.

Aplicações podem requerer requisitos complexos relacionados aos valores contidos nas mensagens. Mesmo se fosse construída uma linguagem declarativa altamente expressiva para representar essas restrições (*Constraints*), esta linguagem seria complexa o suficiente para ser não-prática, e mesmo assim inexpressiva para representar algumas restrições mais específicas. Por essa razão, optou-se por utilizar uma abordagem imperativa ao invés de declarativa para especificar as restrições. Mais especificamente, uma restrição é especificada utilizando-se um componente em Java [60]. Desta forma, os desenvolvedores estão livres para construir restrições tão complexas quanto necessário, limitando-se apenas ao poder de expressão da linguagem Java.

Entretanto, embora o uso de Java possibilite a definição de restrições bastante expressivas, ele esconde o significado da restrição. Se não existir nenhum mecanismo adicional para externalizar um mínimo de semântica da restrição, seria necessário analisar o código Java para entender o que a restrição faz. Por

essa razão, restrições possuem um campo denominado *Semantic* que permite a especificação da semântica de uma determinada restrição.

Restrições são representadas usando-se o XMLaw de acordo com a Tabela 4.12. Como restrições são utilizadas por transições (Seção 4.3), elas são definidas dentro da tag *Transition*. Uma restrição é representada pela tag *Constraint*, que define os atributos *class* e *Semantic*. No atributo *class* define-se a classe Java que implementa a restrição. Esta classe é chamada quando a transição é verificada, e basicamente verifica se os valores das mensagens são válidos. O atributo *Semantic* é o local onde a semântica da restrição deve ser definida. Não é feita nenhuma restrição sobre o formalismo a ser usado.

```
<Transition id=''a-transition-id'' from=''from-state''
to=''to-state'' message-ref=''rfq''>
    <Constraints>
        <Constraint class=''ajavapackage.AConstraintClass''>
            <Semantic>Semantic for this constraint</Semantic>
        </Constraint>
    </Constraints>
</Transition>
```

Tabela 4.12: Constraint Language Definition

4.10 Ações

Um requisito importante para sistemas de software é a habilidade de se auto-adaptar dinamicamente em resposta a atributos como disponibilidade de recursos, necessidades do usuário e faltas no sistema [61]. Estudos estimaram que atualmente empresas gastam de um terço à metade do custo total de um software consertando ou se preparando contra falhas no sistema [62]. Ao mesmo tempo, existe uma crescente pressão para identificar e resolver os problemas de software rapidamente mesmo em ambientes de software cada vez mais complexos. Recuperação automática de software (Automated self-healing) está surgindo em resposta a esses problemas. Recuperação automática pode ser definida pela habilidade de um sistema automaticamente detectar, diagnosticar e reparar problemas de software e hardware [63].

Ações (*Actions*) são os elementos de lei que dão suporte à recuperação automática de software. Ações são componentes Java específicos para cada aplicação, que executam integrados com as especificações em XMLaw. Uma vez que ações são elementos do modelo conceitual de leis, elas podem ser ativadas por eventos tais como a ativação de uma transição, norma, relógio ou mesmo a ativação de uma outra ação.

A recuperação automática utilizando ações pode ser vista como sendo composta por três atividades: monitoramento, detecção e recuperação. O monitoramento é realizado através da especificação das leis no XMLaw, ou seja, através da especificação das leis pode-se monitorar por determinadas condições no software, por exemplo, um agente que está demorando demais para responder. A detecção ocorre quando a condição que está sendo monitorada ocorre e a ação é acionada. Isto se dá porque uma ação é capaz de monitorar os eventos gerados por outros elementos de leis e iniciar o seu comportamento quando um determinado evento ocorrer. Finalmente, a última atividade é a recuperação e esta é realizada pelo código da ação. Uma vez que a ação é um componente Java, qualquer código pode ser executado após a ativação da ação. Estas atividades estão ilustradas na Figura 4.6.

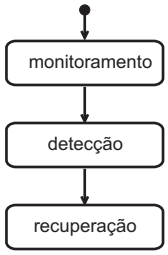

Figura 4.6: Atividades de uma Ação

A especificação de uma ação usando o XMLaw é ilustrada na Tabela 4.13. O atributo *class* especifica a classe Java responsável por implementar a operação de recuperação. A tag *Element* faz referências para os eventos de lei que podem ativar esta ação. Além disso, é possível declarar várias tags *Element*, o que significa que a ação pode ser ativada por vários eventos diferentes. É importante notar que as ações são definidas dentro de uma cena (tag *Scene*), logo as ações atuam no contexto de cenas específicas e não no contexto global.

```
<Actions>
  <Action id="anActionId" class="apackage.ActionClass">
    <Semantic>Semantics for this action</Semantic>
    <Element ref="generatorReference" event-type="type"/>
  </Action>
</Actions>
```

Tabela 4.13: XMLaw de uma ação