3
Trabalhos Relacionados ao Meta-Modelo

Uma abordagem de governança precisa lidar com dois assuntos importantes: o modelo conceitual (também chamado de linguagem de domínio ou meta-modelo) e mecanismos de implementação que dêem suporte para a especificação e aplicação (*enforcement)* das leis baseadas no modelo conceitual. Nesta seção, discute-se principalmente o primeiro assunto.

Através do modelo conceitual, descrevem-se quais elementos os projetistas podem utilizar para especificar as leis. O modelo especifica o vocabulário e a gramática (regras) que os projetistas podem utilizar para projetar e implementar as leis. O modelo possui um impacto decisivo sobre quão fácil se torna a especificação e manutenção de leis. É a abordagem utilizada para projetar um software que determina boa parte da complexidade resultante do software. Quando um software se torna muito complexo, é difícil entendê-lo e, consequentemente, mudá-lo (Inc. 2007).

Em 1987, Naftaly Minsky publicou as primeiras idéias sobre leis (Minsky and Rozenshtein 1987). Em 2000, ele publicou um artigo seminal sobre o papel das leis de interação em sistemas distribuídos (Minsky and Ungureanu 2000), que foi chamado de *Law-Governed Interaction* (LGI). Desde então, Minsky tem conduzido o seu trabalho de pesquisa e experimentação baseado nestas idéias (Minsky 2003; Murata and Minsky 2003; Minsky 2005).

Embora o LGI tenha sido utilizado em uma variedade de domínios de aplicação, seu modelo conceitual é composto predominantemente de abstrações relacionadas às informações de baixo nível e sobre questões de comunicação. Exemplos de tais elementos são as primitivas *disconnected, reconnected, forward,* e o envio e recebimento de mensagens. Embora seja possível especificar regras de interação bastante complexas utilizando estas abstrações de baixo nível, isto pode não ser adequado para projetar ou especificar leis de sistemas considerados complexos. A inadequação ocorre porque é preciso mapear as leis do domínio que estão em um alto nível de abstração para as várias primitivas de baixo nível. Desta

forma, a idéia original da lei é perdida uma vez que ela fica espalhada em várias primitivas de baixo nível. Esta situação reflete o problema de se ter uma linguagem que fornece abstrações que estão muito desconectadas do domínio do problema que se está querendo representar. O desenvolvimento de sistemas complexos e interativos demanda abstrações de alto nível. Sendo assim, as leis devem ser representadas de tal forma que ajudem a reduzir a complexidade com a conseqüente melhora da produtividade do desenvolvimento de sistemas desta natureza.

As instituições eletrônicas (Esteva 2003) são uma outra abordagem para a utilização de leis de interação. Uma instituição eletrônica (IE) possui um conjunto de abstrações de alto nível que permitem a especificação de leis utilizando conceitos tais como papéis de agentes, normas e cenas. Historicamente, as primeiras idéias apareceram quando os autores analisaram o domínio de um mercado de peixes (Noriega 1997). Os autores perceberam que para obter um nível adequado de regulação sobre as ações dos agentes, o mundo real utiliza instituições que definem um conjunto de regras de comportamento, um conjunto de trabalhadores e um conjunto de observadores que monitoram e garantem o comprimento das regras. Baseando-se nestas idéias, os autores propuseram um conjunto de abstrações e um software para que os sistemas de software pudessem ser construídos utilizando abstrações similares às do mundo real. Entretanto, embora uma IE forneça abstrações de alto nível, o seu modelo conceitual é bastante inflexível quando considerado sob a ótica de evolução do próprio modelo. Em um campo de pesquisa novo, como é a linha de instituições eletrônicas, a flexibilidade de uma abordagem se torna um fator crítico. As pesquisas nesta área estão em evolução constante e, consequentemente, o modelo que representa as abstrações de leis e suas respectivas implementações também devem ser capazes de evoluir para se adaptar a estas mudanças. Um exemplo de evolução é a utilização de leis de interação para incorporar preocupações de fidedignidade. Nesta situação, em geral, a lei é utilizada para especificar comportamentos indesejáveis e a partir da ocorrência destes comportamentos, especificar um respectivo plano de ação para mitigar as situações indesejadas.

Entretanto, como um modelo conceitual de leis pode evoluir se não se conhece a priori qual a natureza das mudanças? Embora não seja possível prever quais partes irão evoluir, é possível construir o modelo sobre uma base que seja

inerentemente flexível. Os sistemas baseados em eventos tendem a ser flexíveis. Isto ocorre porque nestes tipos de sistema evita-se a dependência direta entre os módulos. Ao invés disso, a dependência ocorre entre os módulos e os eventos que eles produzem ou consomem.

Em geral, o projeto de um software baseado em eventos evita qualquer conexão direta entre o componente responsável por executar uma operação e os componentes responsáveis por decidir quando executá-las. A orientação a eventos leva a um baixo acoplamento entre os módulos e vem conseguindo uma boa aceitação principalmente pelo suporte na construção de projetos flexíveis de software (Meyer 2003).

Em uma arquitetura baseada em eventos, os componentes de software interagem gerando e consumindo eventos. Quando um evento em um componente (chamado de componente fonte) ocorre, todos os outros componentes (destinatários) que declararam um interesse no evento são notificados. Este paradigma dá suporte a uma interação flexível e efetiva entre componentes de software altamente reconfiguráveis (Cugola, Di Nitto et al. 1998) e vem sendo aplicado com êxito em domínios bastante variados tais como interfaces gráficas, sistemas distribuídos complexos (Cugola, Di Nitto et al. 1998), sistemas baseados em componentes (Almeida, Perkusich et al. 2006) e integração de software (Meier and Cahill 2005). Muitas destas abordagens usam sistemas baseados em eventos para gerenciar as mudanças que não podem ser antecipadas durante o projeto do software (Batista and Rodriguez 2000; Almeida, Perkusich et al. 2006). Estas mudanças geralmente são motivadas por um melhor entendimento do domínio e por fatores externos tais como fatores estratégicos, políticos ou orçamentários.

Neste capítulo, destacam-se duas características do modelo conceitual de leis proposto nesta tese. A primeira refere-se ao fato dele ser composto de abstrações de alto nível e a segunda diz respeito ao paradigma orientado a eventos que faz parte do modelo conceitual. O objetivo não é convencer o leitor de que as abstrações propostas no modelo conceitual são melhores que as abstrações propostas em outras abordagens. Ao invés disso argumenta-se que as abstrações de alto nível propostas, tornam possível a especificação de leis consideradas complexas. E com o fato do modelo ser baseado em eventos, novos elementos podem ser adicionados ao modelo facilitando a sua evolução.

No modelo, cada elemento é capaz de ouvir e gerar eventos. Por exemplo, se o modelo possui a noção de normas, então este elemento, a norma, deve gerar eventos que potencialmente podem interessar a outros elementos. Exemplos destes eventos são eventos relacionados ao ciclo de vida, a ativação da norma, a aplicação de sanções especificadas na norma, dentre outros. A norma também é capaz de escutar por eventos gerados por outros elementos do modelo conceitual e então reagir apropriadamente. Por exemplo, se no modelo conceitual existe um elemento que representa a noção de tempo, tal como um relógio, então as normas podem escutar notificações do relógio e então o seu comportamento pode se tornar sensível a variações do tempo. Este mecanismo leva a relacionamentos entre os elementos de lei bastante flexíveis e expressivos. Além disso, se existe a necessidade de se introduzir um novo elemento no modelo, então a maior parte do esforço fica restrito a conectar este novo elemento aos eventos com que ele precisa reagir e descobrir quais os eventos que este novo elemento precisa propagar.

Esta flexibilidade alcançada através de uma abordagem baseada em eventos em conjunto com abstrações de alto nível não está presente em outras abordagens que também utilizam abstrações de alto nível (Esteva 2003; Dignum, Vazquez-Salceda et al. 2004). As vantagens alcançadas com o uso de eventos como auxílio a modelagem também está presente na abordagem LGI (Minsky and Ungureanu 2000), entretanto as abstrações utilizadas são de baixo nível.

Este capítulo ilustra em detalhes como mapear a linguagem de alto nível proposta nesta tese para a abordagem LGI. Este mapeamento é importante por ilustrar que é possível alcançar os resultados produzidos por LGI sem considerar, até o momento, as dimensões de desempenho e segurança.

3.1. Relação do Modelo Conceitual do XMLaw com uma Abordagem de Baixo Nível Baseada em Eventos

Minsky (Minsky and Ungureanu 2000) propôs um mecanismo de coordenação e controle chamado *Law Governed Interaction* (LGI). Este mecanismo é baseado em dois princípios: a natureza local das leis e a descentralização do *enforcement*, ou seja, a descentralização da maneira que se verifica e garante que a lei está sendo cumprida. A natureza local das leis significa que as leis podem regular somente eventos locais em cada *home agent*. Um *home*

agent é um agente que está sendo regulado pelas leis. A descentralização do *enforcement* é uma decisão arquitetural considerada como necessária para alcançar escalabilidade e evitar a presença de um único ponto de falha no sistema.

LGI possui um rico conjunto de eventos que podem ser monitorados em cada controlador. Um controlador é a denominação dada aos agentes responsáveis por realizar o *enforcement*. Cada *home agent* está associado a um controlador. Uma vez que estes eventos são monitorados, é possível utilizar operações para especificar as leis. A união de eventos e operações compõe o modelo conceitual de LGI. Este modelo foi concebido para lidar com decisões arquiteturais para alcançar um alto nível de robustez. Essa decisão levou a um modelo composto por primitivas de baixo nível de abstração. Embora estas primitivas sejam adequadas para muitas classes de problema, algumas vezes é necessário utilizar várias primitivas para alcançar o comportamento desejado. Uma vez que as leis se tornam mais complexas, a manutenção de leis utilizando estas primitivas de baixo nível se torna uma tarefa mais complicada e propensa a falhas.

É possível pensar no LGI como uma máquina virtual altamente escalável cujas instruções são compostas por elementos de lei de baixo nível. Desta forma, seria possível, por exemplo, utilizar as abstrações de alto nível do XMLaw para especificar as leis e em um segundo passo, mapear a especificação para executar sobre a arquitetura LGI. Com o intuito de ilustrar esta idéia, mostra-se como alguns dos elementos que compõem o modelo conceitual do XMLaw podem ser mapeados para várias primitivas do LGI. Embora esta ilustração não seja exaustiva, ela pode ser facilmente estendida para abordar todos os elementos do modelo do XMLaw e a arquitetura LGI. Ainda para facilitar o entendimento e a comparação entre o XMLaw e o LGI, resumiu-se os principais eventos e operações do LGI na Tabela 4 e Tabela 5.

adopted	Representa o momento que um agente adota uma lei LGI e passa a interagir sob suas condições.
arrived	Ocorre quando uma mensagem M enviada por um agente X para o agente Y chega ao controlador de Y. O evento é gerado no controlador do agente Y.
disconnected	Este evento ocorre quando um agente se disconecta de seu

	controlador.
exception	Este evento pode ocorrer uma operação invocada por um agente falha.
obligationDue	Este evento funciona como um relógio que lembra ao controlador que uma obrigação previamente estabelecida expirou. Obrigação são declaradas pela primitiva imposeObligation.
reconnected	Ocorre quando um agente que se desconectou previamente, se reconectou.
sent	Ocorre quando o agente X envia uma mensagem para um agente Y. O evento ocorre no controlador do agente X.
stateChanged	Ocorre quando uma obrigação de estado expira.

Tabela 4 – Lista dos Principais Eventos do LGI

Deliver	Esta operação envia uma mensagem. A estrutura desta operação é dada por: deliver([x,L'],m,y), onde *x* é o agente remetente, *m* a mensagem, L' a lei e *y* o agente destinatário.
Forward	Redireciona uma mensagem.
Add	Adiciona termos ao CS (Control-State).
Remove	Remove termos do CS.
Replace	Substitui termos do CS.
Incr	Esta operação possui a forma incr(f,d) e incrementa o termo *f(n)* de *d* unidades.
Decr	Esta operação é a contraparte da operação Incr.
replaceCS	Esta operação possui a forma: replaceCS(termList) e substitui todo o conteúdo do CS com a lista de termos especificada.
addCS	Esta operação possui a forma: addCS(termList). Adiciona a lista de termos ao CS.
imposeObligation	imposeObligation(oType, dt, timeUnit). Declara que uma obrigação de um determinado tipo expira

	após o intervalo de tempo especificado.
repealObligation	repealObligation(oType). Remove todas as operações do tipo especificado.
imposeStateObligation	imposeStateObligation(termList). Faz com que seja gerado um evento stateChanged quando ocorrer qualquer mudança em algum dos termos do CS.
repealStateObligation	repealStateObligation(all). Contraparte da operação imposeStateObligation.

Tabela 5 – Lista das Principais Operações do LGI

A maioria dos eventos listados na Tabela 4 estão relacionados a informações de baixo nível sobre comunicação (*disconnected*, *reconnected*), envio ou recebimento de mensagens (*sent, arrived*), ou mudanças de estado no controlador de estados (*stateChanged*). Do ponto de vista das operações, elas também estão relacionadas principalmente a instruções de baixo nível tais como *forward, deliver, add* e assim por diante.

O protocolo mostrado na Figura 4 pode ser especificado de forma direta em XMLaw através dos elementos de lei *Protocol*, *State*, *Transition* e *Message*. O Código 23 ilustra como este protocolo é especificado em XMLaw. Para alcançar o mesmo comportamento no LGI, pode-se escrever as leis conforme ilustrado no Código 24. Na lei em LGI, foi necessário introduzir dois novos termos que não fazem parte dos termos pré-definidos do LGI: *currentState* e *event*. O termo *currentState* modela os estados atuais do protocolo e o termo *event* simula a geração de eventos. Por exemplo, na primeira linha da listagem de Código 24, um agente *A* envia a mensagem *m1* para o agente *B*. Se o estado atual é *s0*, então o estado *s0* é removido da lista de estados atuais e o estado s1 é adicionado a esta lista. Após isto, simula-se a geração do evento *transition_activation* e finalmente a mensagem é redirecionada.

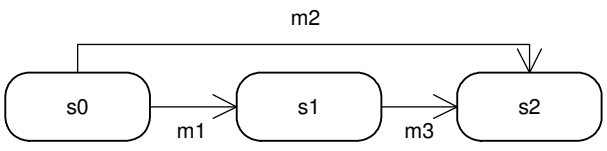

Figura 4 – Exemplo de Protocolo

```
t1{s0->s1, m1}
t2{s0->s2, m2}
t1{s1->s2, m3}
```

Código 23 – Mapeamento do Protocolo da Figura 4 usando XMLaw

```
sent(A,m1,B) -> currentState(s0)@CS,
do(remove(currentState(s0))), do(add(currentState(s1))),
do(add(event(t1,transition_activation))), do(forward).
sent(A,m2,B) -> currentState(s0)@CS,
do(remove(currentState(s0))), do(add(currentState(s2))),
do(add(event(t2,transition_activation))), do(forward).
sent(B,m3,A) -> currentState(s1)@CS,
do(remove(currentState(s1))), do(add(currentState(s2))),
do(add(event(t3,transition_activation))), do(forward).
```

Código 24 – Mapeamento do Protocolo da Figura 4 usando LGI

Os exemplos mostrados na Tabela 6 mostra como situações encontradas no XMLaw podem ser mapeadas para a abordagem LGI. Para isto incorpora-se na semântica expressa em prolog do LGI os termos *event*, *currentState* e *norm*.

1. No recebimento de uma mensagem *m1*, um clock precisa ser ativado para disparar um evento em 5 segundos. O clock deve ser desativado quando a mensagem *m2* chegar.
XMLaw `myXMLawClock{5000,regular, (m1),(m2)}`

LGI
`arrived(X, m1, Y) :- imposeObligation("myLGIclock",5).`
`arrived(X, m2, Y) :- repealObligation("myLGIclock”).`

2. Disparar a transição *t2* quando um clock gerar um evento do tipo *clock_tick*. A transição muda o protocolo do estado *s1* para o estado *s2*.
XMLaw `t2{s1->s2, myXMLawClock}`

LGI
`obligationDue("myLGIclock ") :- currentState(s1)@CS, do(remove(currentState(s1))),`
`do(add(currentState(s2))), do(add(event(t2,transition_activation))), do(forward).`
`/*`
`comentário: o evento obligationDue ocorre quando o tempo especificado a obrigação expirar.`
`*/`

3. Declarar um clock do tipo periódico que precisa gerar um evento a cada cinco segundos. Este clock deve ser ativado pela chegada da mensagem *m1* e desativado pela chegada da mensagem *m2*.

XMLaw myXMLawClock{5000,periodic, (m1),(m2)}
LGI arrived(X, m1, Y) :- imposeObligation("myLGIclock",5). arrived(X, m2, Y) :- repealObligation("myLGIclock”). obligationDue("myLGIclock") :- imposeObligation("myLGIclock ",5). /* comentários: o evento obligationDue ocorre quando o tempo especificado na obrigação expira. Pode-se utilizar um loop na especificação para simular clocks periódicos. Neste exemplo, declara-se um clock e quando o clock expira, um evento *obligationDue* é gerado o que, por sua vez, ativa outra imposeObligation, e assim por diante. */
4. Uma mensagem *m1* ativa a transição *t1*. A transição *t1* muda o estado do protocolo de *s1* para *s2*. Uma norma *n1* precisa ser ativada quando a transição *t1* disparar. A norma é dada para o agente que recebeu a mensagem *m1*.
XMLaw t1{s1->s2, m1} n1{$addressee, (t1) }
LGI sent(A,m1,Addresee) -> currentState(s1)@CS, do(remove(currentState(s1))), do(add(currentState(s2))), do(add(event(t1,transition_activation))), do(forward). imposeStateObligation(event(t1,transition_activation)). stateChanged(event (t1,transition_activation)) :- do(add(event(n1,norm_activation))), do(add(norm(n1,active,valid)). /* comments: Neste caso, o imposeStateObligation causará um evento stateChanged todas as vezes que o termo event(t1,transition_activation) for adicionado ao CS. Então, no terceiro comando, quando este evento stateChanged ocorrer, a norma n1 é ativada. */

Tabela 6 – Mapeamento entre o XMLaw e o LGI

3.1.1.Propriedades Globais

De acordo com (Minsky 2005): “qualquer política que pode ser implementada através de um mediador central e que mantenha o estado global da interação de toda a comunidade de agentes, também pode ser implementado via uma lei LGI”. Como um exemplo de propriedade global, suponha a situação da Figura 5. Neste exemplo, 3 agentes estão interagindo no contexto de um determinado protocolo. O agente A envia a mensagem m1 para o agente B. Como os agentes A e B estão interagindo, seus controladores atualizam o estado atual (cada controlador atualiza o seu próprio estado atual de forma independente). Entretanto, o agente C não participou desta interação e, portanto, seu controlador

não atualizou o estado atual. Isto leva a uma inconsistência entre os estados atuais dos controladores dos agentes A e B e o estado atual do agente C. A causa principal disto é o monitoramento sendo realizado de forma descentralizada e sem nenhuma sincronização explícita.

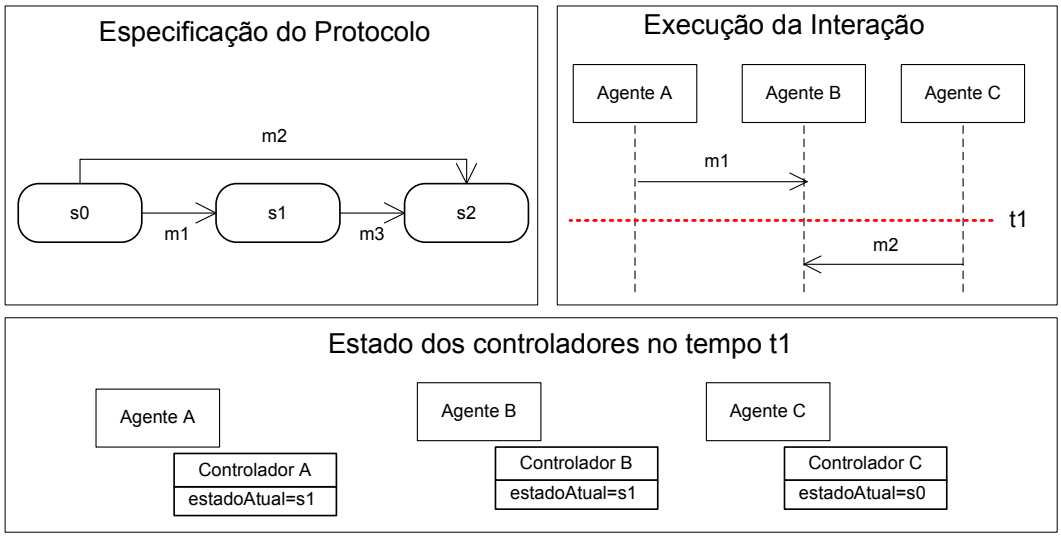

Figura 5 – Exemplo da Necessidade de Sincronização entre os Controladores para a Identificação de Propriedades Globais.

A forma geral para superação deste problema é a utilização de protocolos de sincronização como o *token ring* que foi utilizado na abordagem *Islander* (Esteva 2003). No LGI, este comportamento pode ser alcançado introduzindo-se um coordenador central que recebe todas as mensagens e mantém um estado global consistente. O Código 25 ilustra em LGI como este coordenador (mediador) pode ser especificado na lei. As leis em XMLaw são especificadas de um ponto de vista global (Paes, Carvalho et al. 2007). A abordagem ilustrada no Código 25 possui uma vantagem sobre o mediador centralizado utilizado no XMLaw. Como os mediadores do XMLaw precisam receber todas as mensagens, interpretá-las e só então, eventualmente, descartá-las, eles não podem se proteger contra o congestionamento de mensagens geradas por agentes super-ativos. Isto pode levar o mediador a problemas de negação de serviço (*deny of service)*. Por outro lado, utilizando-se LGI, é possível especificar o limite da freqüência de mensagens emitidas pelos participantes. E o mais importante é que este limite é verificado localmente nos controladores.

```
alias(coordinator,'law-coordinator@les.inf.puc-rio.br').

// any message is forward to the central coordinator
sent(X,M,Y) :- do(forward(X,[M,Y],#coordinator)),
do(forward).

// laws ...and redirection to the real addressee
arrived(#coordinator,A,m1,B) -> currentState(s1)@CS,
do(remove(currentstate(s1))), do(add(currentState(s2))),
do(deliver), do(forward(A,m1,B)).
```

Código 25 – Redirecionamento de Mensagens para um Mediador Central

3.2.Relação do Modelo Conceitual do XMLaw com uma Abordagem de Alto Nível não Baseada em Eventos

As Instituições Eletrônicas (IE) são uma tecnologia para monitorar e garantir o cumprimento de leis em uma sociedade de agentes. Vários estudos de caso foram apresentados utilizando-se esta abordagem. Eles incluem um sistema para um mercado de peixes (Cuni, Esteva et al. 2004), um ambiente de computação em grade (Ashri, Payne et al. 2006), e uma aplicação para controle de tráfego terrestre (Bou, López-Sánchez et al. 2006).

As IEs utilizam um conjunto de conceitos que possuem pontos de contato com os conceitos utilizados no XMLaw. Por exemplo, tanto as cenas e protocolos do IE especificam, como no XMLaw, a interação de um ponto de vista global. O aspecto temporal é representado através de *timeouts*. Os *timeouts* permitem que sejam disparadas transições depois que um certo número de unidades de tempo tenha passado desde que um estado do protocolo tenha sido alcançado. Por outro lado, por causa do modelo de eventos, o elemento *clock* (relógio) proposto pelo XMLaw pode tanto ativar quanto desativar não apenas transições, mas também outros *clocks* e normas. A conexão de *clocks* e normas permite um comportamento normativo mais expressivo uma vez que as normas se tornam elementos sensíveis ao tempo. Além disso, outra diferença entre os dois modelos é a presença do elemento *action* no XMLaw. Este elemento permite a execução de código Java em resposta a alguma situação na interação.

A Tabela 7 compara as abstrações utilizadas no modelo conceitual de ambas as abordagens. O objetivo desta comparação é relacionar precisamente o XMLaw

com uma abordagem existente e amplamente conhecida na área de governança. Através desta comparação é possível analisar as vantagens e desvantagens de cada uma das abordagens quando se considera o uso delas na implementação de governança. A comparação mostra que, embora as abordagens compartilhem um número razoável de conceitos, eles possuem diferenças importantes tal como, por exemplo, a noção de normas que está presente em IE é melhor definida do que no XMLaw. Por outro lado, o conceito de *actions* do XMLaw pode ser bastante útil para tornar o comportamento das leis mais ativo e integrado com serviços providos pelo ambiente. Essencialmente, a maior diferença entre os dois modelos é a maneira pela qual as abstrações se relacionam para compor uma lei.

Na abordagem de IE existe um conjunto fixo de relacionamento entre os elementos e a maneira pela qual estes elementos são utilizados em conjunto é definida a priori. Um bom exemplo desta idéia é o conceito de *timeout* presente em uma IE. Esta abstração é bastante similar ao *clock* do XMLaw. Entretanto, um *timeout* só pode ser utilizado com transições. Se o modelo de comunicação entre os elementos fosse mais flexível, seria possível , por exemplo, fazer o mesmo que o XMLaw faz e conectar o *timeout* a uma norma.

Instituição Eletrônica	XMLaw	Comentários
Illocutory formulas	Message	Possuem estruturas diferentes, mas possuem propósito similar.
EI vocabulário (ontologia)	Define-se o vocabulário na própria definição da mensagem ao invés de separadamente.	IE define uma ontologia explicitamente. No XMLaw esta definição não é realizada de forma explícita.
Papéis Internos	Não considerado	Papéis internos definem um conjunto de papéis que irão ser desempenhados por "agentes internos". Uma IE delega a implementação de alguns serviços para estes agentes internos. Um agente externo não pode desempenhar um papel interno.
Papéis Externos	Papel (Role)	
Relacionamento entre papéis	Não considerado	
Controle sobre quem desempenha um papel	Controle sobre quem desempenha um papel	Ambas as abordagens fornecem mecanismos de controle sobre o número mínimo e máximo de agentes que podem desempenhar papéis em uma cena.
Cena (Scene)	Cena (Scene)	As duas abordagens possuem o conceito de Scene. Em uma IE, é necessário especificar quando os agentes entram e saem das cenas. No XMLaw só é necessário especificar o momento de entrada.
Performative Structure	Não considerado	É um tipo especial de cena (Scene) que aceita transições de outras cenas e possui transições para cenas. Eles

		possibilitam a especificação da sequência de execução esperada entre as cenas. No XMLaw, alcança-se efeito similar através de normas: ao término de uma cena, uma norma pode ser ativada e verificada como pré-requisito para iniciar uma cena.
Protocolo (Protocol)	Protocolo (Protocol)	
Estado (State)	Estado (State)	Estados são bastante similares, exceto pelo fato de que no XMLaw existem dois tipos de estados finais: *failure* e *success.*
Arco direcionado (Directed edge)	Transition .	Arcos direcionados de uma IE podem ser ativados por uma *Illocutory formula*, uma *timeout* ou *constraints*. No XMLaw transições podem ser ativadas por qualquer evento.
Constraint	Constraint	Em uma IE as *constraints* são implementadas como expressões boleanas utilizando-se a forma (op expr1 expr2). No XMLaw, elas são implementadas utilizando-se código Java.
Time-out	Clock	Time-outs permitem o disparo de transições após um determinado período de tempo. Por outro lado, um *clock* é um elemento de propósito mais geral que também pode ser utilizado para disparar transições. Mas ele pode ser utilizado para outros fins tal como fazer com que uma norma expire após um determinado período.
Regras normativas (Normative rules)	Normas (Norms)	Ambas as abordagens possuem as noções de obrigação, permissão e proibição. As regras normativas de uma IE verificam: "QUANDO uma mensagem é enviada SE a mensagem satisfaz determinadas condições ENTÃO outra mensagem com determinadas condições precisa ser emitida no futuro". Por outro lado, a norma no XMLaw pode ser utilizada para evitar o disparo de transições, ativar ações ou qualquer outro elemento do XMLaw.
Não considerado	Actions	As *actions* podem ser utilizadas para "plugar" serviços no mediador. Elas podem ser ativadas por qualquer evento do XMLaw e são implementadas através de código Java.
Não considerado	Law	No XMLaw, o elemento *Law* é o contexto global onde se compartilha informações entre cenas, normas, clocks e actions.

Tabela 7 – Relacionamento entre os Modelos Conceituais de XMLaw e IE

3.3.Moise +

O modelo Moise+ estabelece quais os componentes que formam uma organização. Este modelo foi desenvolvido com o objetivo principal de auxiliar o processo de reorganização. O Moise+ é composto por três dimensões: a estrutura (papéis), o funcionamento (planos globais) e as normas (obrigações) da organização. O aspecto estrutural atém-se aos componentes elementares da organização (papéis) e como estão relacionados (ligações entre papéis, grupos de papéis, hierarquias). O aspecto funcional especifica como os objetivos globais

podem ser atingidos (planos globais, missões e metas). Por fim, o aspecto deôntico relaciona os dois anteriores, indicando quais as responsabilidades dos papéis nos planos globais. Os principais elementos que compõem o seu modelo conceitual estão descritos na Tabela 8.

Papel	É um tipo primitivo sobre o qual são estabelecidas restrições estruturais e funcionais.
Missão	É parte de um esquema social e, como tal, é uma estrutura de decomposição de metas através de planos. A relação de obrigação e permissão também existe, não é definida na própria missão, mas na sua relação com a estrutura social.
Ligações	São restrições sobre os papéis.
Grupo	Conjunto de papéis, ligações e subgrupos com as cardinalidades.
Instanciação	É um conjunto de pares (agente, papel), um conjunto de grupos instanciados e um conjunto de ligações instanciadas.

Tabela 8 – Elementos do Moise+

O Moise+ é composto por abstrações de alto nível e possui associado um mecanismo de implementação que permite a verificação da conformidade com as leis (Hübner, Sichman et al. 2006). O Moise+ também possui a noção de eventos, entretanto, os eventos são utilizados para comunicar mudanças organizacionais aos agentes. Os eventos não são utilizados para realizar a composição entre os elementos do modelo conceitual, como ocorre no XMLaw.

3.4.Estudos de Caso

Nesta seção, foram escolhidos dois exemplos publicados na literatura para ilustrar a aplicabilidade do modelo do XMLaw e facilitar a comparação com os trabalhos relacionados. O primeiro exemplo já foi publicado utilizando-se a abordagem LGI e o segundo exemplo foi publicado utilizando-se a abordagem IE. Estes dois exemplos foram escolhidos propositadamente para facilitar a análise dos prós e contra das diferentes abordagens.

3.4.1.Estudo de Caso 1: Equipe de Compradores

Este estudo de caso foi publicado em (Minsky and Murata 2004). Para esta tese, a descrição do problema foi ligeiramente modificada, eliminando, por exemplo, a necessidade de uma autoridade de certificação. O exemplo é descrito a seguir:

"Considere-se uma loja de departamento que possui uma equipe de agentes cujo objetivo é prover a loja com as mercadorias que ela precisa. Esta equipe consiste de um gerente e um conjunto de empregados (ou agentes de software que os representam) que estão autorizados como compradores e possuem um orçamento próprio para as compras. Sob circunstâncias normais, a operação da equipe será bem sucedida se todos os membros seguirem a seguinte política:"

1. A equipe de compradores é inicialmente gerenciada por um agente especial chamando *firstMgr*. Mas qualquer gerente desta equipe pode, a qualquer momento, escolher outro agente autenticado como um empregado como o seu sucessor. Uma vez feita a escolha, o gerente perde todos os seus poderes gerenciais e o escolhido os adquire;
2. Caso o agente possua recursos suficientes, é permitido que um comprador emita ordens de compra (POs). O custo da ordem será subtraído do orçamento do agente que emitiu a ordem. A cópia de cada PO emitida precisa ser enviada para o gerente da equipe;
3. Qualquer empregado pode receber um orçamento do gerente. Além disso, cada empregado também pode doar parte do seu orçamento para outros empregados, recursivamente. O gerente pode reduzir o orçamento de qualquer empregado *e*. Como conseqüência, o orçamento do empregado *e* é bloqueado, ou seja, ninguém pode doar orçamento para ele enquanto o gerente não mudar.

Mensagens. O item 1 da política acima é realizado quando o agente desempenhando o papel de gerente envia a mensagem *transfer* para o empregado que irá ser o sucessor. O item 2 acontece quando o comprador envia a mensagem *purchaseOrder(Amount)*, onde *Amount* é o valor da ordem de compra que irá ser reduzido do orçamento do comprador. Em relação ao item 2, um agente atribui um orçamento para outros através da

mensagem *giveBudget(Amount)*. Então, o concedente do orçamento terá o seu orçamento reduzido pelo valor de *Amount* e o destinatário terá o seu orçamento aumentado do mesmo valor. Finalmente, os gerentes podem enviar a mensagem *removeBudget(Amount)*. O efeito desta mensagem é reduzir o valor de *Amount* do orçamento do destinatário.

Solução XMLaw. XMLaw possui abstrações para decompor o problema em informações menores e mais gerenciáveis. Além disso, também é possível estruturar os etapas de interação de um protocolo de conversação complexo. Neste exemplo, as interações não seguem uma ordem pré-definida e o protocolo não é complexo o suficiente para justificar uma decomposição em várias partes menores. Sendo assim, especificou-se as leis utilizando apenas uma cena que encapsula o protocolo de interação e um conjunto de normas, *actions* e *constraints*. A especificação completa pode ser vista no Código 26 e o código para as *actions* e *constraints* utilizadas na especificação podem ser vistas nas listagens de código: Código 27, Código 28, Código 29 e Código 30.

A explicação da solução começa por uma descrição sucinta da sintaxe e da dinâmica dos elementos. Na Figura 6, também apresenta-se uma notação gráfica do protocolo baseado nos diagramas de estado da UML (Group 2007).

Existem cinco tipos de mensagens que podem ser trocadas pelos agentes, estas mensagens estão especificadas das linhas 02 até as linhas 06. O formato da mensagem é *message-id{sender,receiver,content}*. O símbolo * denota qualquer valor. Também é possível manipular variáveis. As variáveis são armazenadas no contexto. Cada cena possui o seu próprio contexto e também existe um contexto global da lei. Os contextos formam uma hierarquia.

A mensagem especificada na linha 02 significa que a identificação do agente remetente será atribuída à variável *budgetOwner*, o destinatário pode ser qualquer agente e o conteúdo tem a forma *purchaseOrder(valor)*, onde o valor será atribuído a variável *amount*. As mensagens são usadas para ativar cinco transições do protocolo (linhas 09 até 13). Entretanto, as transições t1, t2, t3, e t4 possuem *constraints* que serão verificadas antes de serem disparadas. As transições só disparam se as *contraints* forem satisfeitas. As *constraints* estão

especificadas nas linhas 14 e 15. Uma vez que as transições disparem, algumas delas podem ativar *actions*, conforme pode ser visto nas linhas 16 até 18. Finalmente a transição t2 precisa que a norma especificada na linha 19 esteja ativa para que a transição dispare.

A transição *t1* controla as ordens de compra descritas no item 02 da política. Para ser ativada, a *constraint enoughMoney* referenciada na linha 09 verifica se o remetente identificado pela variável *budgetOwner* (linha 02) possui dinheiro suficiente para emitir a ordem (Código 29). A transição t2 controla o item 03 da política. Ela se refere a situação onde um empregado ou gerente atribui um orçamento a outro empregado. Então, a constraint *enoughMoney* referenciada na transição t2 verifica se o remetente possui dinheiro suficiente para efetivar a transferência (Código 29). Depois disso, verifica-se se o empregado que está prestes a receber o dinheiro tem permissão para recebê-lo, conforme especificado na política 03. Esta verificação é feita através da norma *increaseBudgetProhibition* (linha 19). Esta norma é atribuída a um empregado quando o gerente envia a mensagem *removeBudget* e esta mensagem ativa a transição t3. No XMLaw, isto pode ser visto na linha 19 onde t3 é a transição que ativa a norma e t4 é a transição que a desativa. Portanto, se a norma está ativa, a transição t2 não é disparada. Se o agente não possui esta norma ativa associada a ela, então a transição t2 irá disparar. Uma vez que a transição t2 dispare, a *action changeBudget* é ativada (linha 18). O código desta *action* pode ser visto no Código 30. A transição t4 é ativada quando o gerente envia a mensagem *transfer* para um empregado. No protocolo, a *constraint checkTransfer* (Código 29) garante que o remetente da mensagem é de fato o gerente. Se a transição t4 disparar, a *action switchManager* (Código 28) é executada e a norma *increaseBudgetProhibition* é desativada. A *action switchManager* atualiza o gerente atual e uma vez que a norma *increaseBudgetProhibition* não esteja ativada, os empregados que tinham essa norma associada a eles estão agora livres novamente para receber orçamentos através da mensagem *giveBudget*.

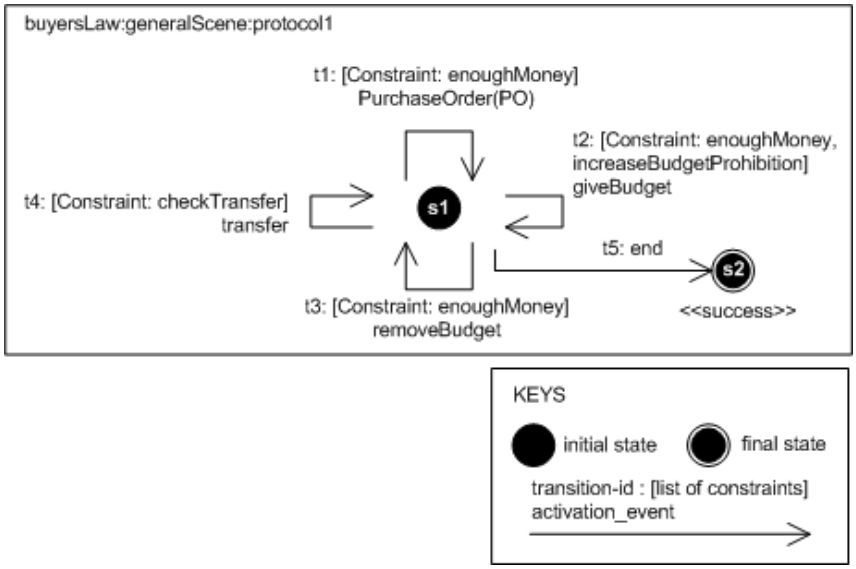

Figura 6 – Protocolo de Interação do Estudo de Caso da Equipe de Compradores

```
01: generalScene{
02:    PO{$budgetOwner,*,purchaseOrder($amount)}
03:  removeBudget{manager,$budgetOwner,removeBudget($amount)}
04:    giveBudget{$budgetOwner,$receiver,giveBudget($amount)}
05:    transfer{$manager,$employee,transfer}
06:    end{$sender,$receiver,end}

07:    s1{initial}
08:    s2{success}

09:    t1{s1->s1, PO, [enoughMoney]}
10:    t2{s1->s1, giveBudget,
[enoughMoney],[increaseBudgetProhibition]}
11:    t3{s1->s1, removeBudget, [enoughMoney]}
12:    t4{s1->s1, transfer, [checkTransfer]}
13:    t5{s1->s2, end}

14:    enoughMoney{br.pucrio.EnoughMoney}
15:    checkTransfer{br.pucrio.CheckTransfer}

16:    forwardMessage{(t1), br.pucrio.ForwardMessage}
17:    switchManager{(t4), br.pucrio.SwitchManager}
18:    changeBudget{(t2,t3), br.pucrio.ChangeBudget}

19:    increaseBudgetProhibition{$budgetOwner, (t3),(t4)}
20:}
```

Código 26 – XMLaw do Estudo de Caso da Equipe de Compradores

```
class CheckTransfer implements IConstraint{
    public boolean constrain(ReadonlyContext ctx){
        String actualManager = ctx.get(“actualManager”);
        String currentMgr = ctx.get(“manager”);
        if (! actualManager.equals(currentMgr)){
            return true; // constrains, transition should not
fire
        }
        return false;
    }
}
```

Código 27 – *Constraint* que Verifica se o Agente é de Fato o Gerente Atual

```
class SwitchManager implements IAction{
    public void execute(Context ctx){
        String employee = ctx.get(“employee”);
        ctx.put(“actualManager”,employee);
    }
}
```

Código 28 – *Action* que Troca o Gerente Atual

```
class EnoughMoney implements IConstraint{
    public boolean constrain(ReadonlyContext ctx){
        String budgetOwner = ctx.get(“budgetOwner”);
        double currentBudget =
Double.parseString(ctx.get(budgetOwner));
        double amount = ctx.get(“amount”);
        double diff = currentBudget – amount;
        if ( diff < 0){
            // constrains, not enough money. Transition
should not fire
            return true;
        }
    }
}
```

Código 29 – *Constraint* que Verifica se o Agente que está Cedendo Dinheiro Realmente tem Dinheiro para Ceder

```
class ChangeBudget implements IAction{
    public void execute(Context ctx){
        String budgetOwner = ctx.get(“budgetOwner”);
        double currentBudget =
Double.parseString(ctx.get(budgetOwner));
        double amount = ctx.get(“amount”);
        // update the owner`s budget (the one who is given
money)
        budget.put(budgetOwner, currentBudget – amount);

        String receiver = ctx.get(“receiver”);
        if (receiver!=null){ // if there is a receiver
```

```
            double receiverBudget =
Double.parseString(ctx.get(receiver));
            // update the receiver`s budget
            ctx.put(receiver, receiverBudget+amount);
        }
    }
}
```

Código 30 – *Action* que Atualiza os Orçamentos e que é Utilizado nas Mensagens de *giveBudget* e *removeBudget*

Discussão. A parte mais importante deste estudo de caso é o Código 26. Este código contém os elementos do modelo conceitual do XMLaw. O código apresentado é predominantemente declarativo e é composto de abstrações de alto nível tais como protocolos de interação, *actions*, *constraints* e normas. Foi possível expressar as regras em vinte linhas de comandos que são relativamente simples de entender mesmo para aqueles não estão acostumados com o XMLaw. Mesmo com o código Java necessário para implementar as *actions* e *constraints*, na maior parte do tempo o projetista pode focar na especificação das leis do Código 26 e utilizar as *actions* e *constraints* como componentes para alcançar a funcionalidade desejada. O modelo de comunicação entre os elementos baseado em eventos está presente na maior parte das declarações. Por exemplo, a linha 19 usa a notificação de eventos para especificar que a norma *increaseBudgetProhibition* é ativada pelo evento *transition_activation* gerado pela transição t3. Além disso, ela é desativada pelo evento *transition_activation* gerado pela transição t4. Outro exemplo é a transição t1 na linha 09 que é ativada pelo evento *message_arrival* gerado pela mensagem *PO*. É claro que a sintaxe da linguagem abstrai a maior parte dos detalhes do projetista e permite que o modelo de eventos funcione "por trás das cenas". Embora este estudo de caso seja relativamente pequeno, ele é bastante útil para tornar as idéias apresentadas neste capítulo mais concretas. O uso de um estudo de caso que já foi publicado e implementado em outra abordagem torna possível uma comparação bastante direta.

Quando comparado a solução publicada em (Minsky and Murata 2004), as leis do XMLaw fornecem um nível maior de abstração quando analisados sob a ótica de descrição do problema até a sua solução. Por exemplo, a restrição descrita no item 2 da política: "Caso o agente possua recursos suficientes, é permitido que

um comprador emita ordens de compra (POs)" é mapeada diretamente para a *constraint enoughMoney* do XMLaw utilizada na linha 09.

3.4.2.Estudo de Caso 2: Centro de Conferências

Este estudo de caso foi implementado utilizando-se Instituições Eletrônicas e foi apresentado em (Rodriguez-Aguilar 2001; Esteva 2003). O exemplo é descrito como se segue:

"Uma conferência ocorre em um centro de conferências. Diferentes atividades ocorrem simultaneamente em diferentes locais protagonizados por pessoas que desempenham papéis diferentes (palestrante, *chair*, organização, etc.). Durante a conferência, as pessoas se orientam pelos seus interesses movendo-se pelos diferentes locais e se engajando em atividades diferentes. O agente pessoal é um agente que está imerso em um ambiente virtual e que é responsável por auxiliar um participante da conferência na procura de informações de interesse e no diálogo com outros agentes de software."

O exemplo apresentado em (Esteva 2003) estruturou esta aplicação em seis cenas diferentes: *Information Gathering*, *Context*, *Appointment Proposal*, *Appointment Coordination*, *Advertiser*, e *Delivery*. Entretanto, a única cena em que foram apresentados detalhes suficientes foi a *Appointment Proposal*. Desta forma, esta cena será o foco de comparação neste segundo estudo de caso.

Os participantes desta cena são dois agentes pessoais (PRA). O objetivo desta cena é alcançar um consenso sobre um conjunto de tópicos para discutir em um encontro. A cena se desenvolve de acordo com os seguintes passos:

1. Um dos PRAs (PRA1) toma a iniciativa e envia uma proposta contendo um conjunto de tópicos iniciais para marcar um encontro com o outro PRA (PRA2). Esta proposta possui um tempo que define a sua validade.
2. PRA2 avalia a proposta e pode aceitar, rejeitar ou enviar uma contraproposta para PRA1 com o conjunto diferente de tópicos. A contraproposta também possui uma validade.
3. Quando o PRA1 recebe a contraproposta de PRA2, o PRA1 avalia a contra proposta e pode aceitar, rejeitar ou enviar outra

contraproposta para PRA2. Esta fase de negociação termina quando um consenso for alcançado ou quando um deles desistir de negociar.

Como foi dito anteriormente, os PRAs participam de um ambiente virtual representando um participante da conferência. Quando um PRA alcança um consenso sobre um conjunto de tópicos, ele precisa informar o participante.

Solução XMLaw. A Figura 7 mostra uma representação gráfica do protocolo de interação para a cena *Appointment Proposal*. A especificação XMLaw é mostrada no Código 31. Existem três tipos de mensagens: *propose, accept,* e *decline* (linhas 02 até 04). Estas mensagens são utilizadas para disparar a maior parte das transições (linhas 08, 09, 10, 11, 13 e 14). A transição t5 é ativada pelo *clock* na linha 16. Este *clock* representa o tempo de validade das propostas. Ele é ativado todas as vezes que as transições t1, t4 ou t6 disparam, e é desativado quando as transições t2, t3, t4 ou t5 dispararem. Este *clock* está configurado para gerar um evento de *clock_tick* 5000 milisegundos após a sua ativação. A linha 15 declara a norma *app-notification*. Esta norma significa que foi alcançado um consenso na negociação e, portanto, os participantes precisam comparecer ao encontro. Esta norma pode, por exemplo, ser utilizada em outras cenas para impedir que agentes que não tenham cumprido a sua obrigação interajam. A norma é dada ao PRA que aceita a proposta dos tópicos. A aceitação ocorre na transição t2.

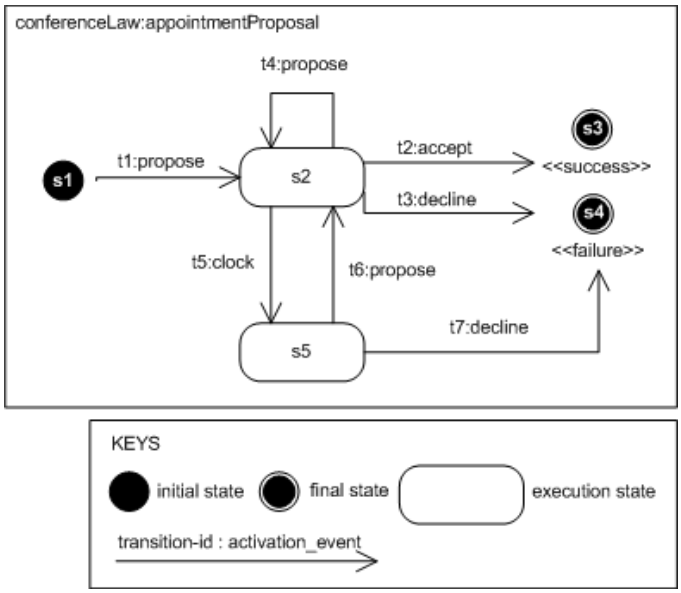

Figura 7 – Protocolo de Interação do Estudo de Caso Centro de Conferências

```
01: appointmentProposal{
02:    propose{$PRA1,$PRA2,$topics}
03:    accept{$PRA1,$PRA2,$topics}
04:    decline{$PRA1,$PRA2,$reason}

05:    s1{initial}
06:    s3{success}
07:    s4{failure}

08:    t1{s1->s2, propose}
09:    t2{s2->s3, accept}
10:    t3{s2->s4, decline}
11:    t4{s2->s2, propose}
12:    t5{s2->s5, clock}
13:    t6{s5->s2, propose}
14:    t7{s5->s4, decline}

15:    app-notification{$PRA1, (t2),()}

16:    clock{5000,regular, (t1,t4,t6),(t2,t3,t4,t5)}

17:}
```

Código 31 –XMLaw da Cena *appointmentProposal*

Discussão. Quanto comparada com a solução apresentada em (Esteva 2003), a solução apresentada aqui possui algumas diferenças importantes: (i) o conjunto de definição de mensagens foi reutilizado muitas vezes na definição do protocolo. Isto levou a um protocolo muito mais simples (por exemplo, o número de transições diminui de 13 para 7); (ii) por causa do modelo baseado em eventos, o clock é conectado na lei para disparar as transições. Quando comparado com a IE, a própria transição possui um elemento de *timeout*. Em outras palavras, a transição possui a funcionalidade do *clock*. Esta separação leva a uma melhor separação de conceitos (*concerns*), e a uma melhor reutilização, uma vez que os *clocks* podem ser compostos com outros elementos; (iii) como a norma também está conectada ao evento de modelos, a sua ativação é bem simples, basta que se especifique o evento que ativa a norma.

3.5.Considerações Finais

Neste capítulo, mostrou-se que o modelo conceitual do XMLaw é composto por abstrações de mais alto nível quando comparado com as primitivas do LGI. Também foi mostrado que a noção de eventos leva o XMLaw a um modelo mais flexível para acomodar mudanças e compor os elementos quando comparado com as instituições eletrônicas.

Para ser mais preciso, tanto XMLaw e LGI lidam com a troca de mensagens entre agentes e não são sensíveis ao comportamento interno dos agentes e nem às mudanças no estado interno destes agentes. Em geral, LGI é mais efetivo para leis que são naturalmente locais enquanto o XMLaw é mais efetivo quando as leis são naturalmente globais. Em ambas as abordagens as leis não foram projetadas para especificar todos os detalhes da interação entre os agentes, ao invés disso, as leis especificam as restrições sobre a interação. Do ponto de vista conceitual, o LGI fornece uma abstração do estado de interação de cada um dos agentes (*control state)*, um conjunto de eventos relacionados a comunicação e um conjunto de operações para manipular o *control state*. Esse estado funciona basicamente como uma *hashtable* onde os termos são armazenados. Não existe restrição sobre os termos que podem ser utilizados. Esta liberdade pode levar a uma grande flexibilidade que pode ser útil para adaptar a abordagem a diferentes domínios de aplicação. Entretanto, um pequeno conjunto de predicados de alto nível poderia ser mais útil para auxiliar a coordenação e *enforcement* das leis sem a complexidade de ter que criar novos termos. O mapeamento dos elementos do XMLaw para LGI pode ser visto como a tarefa de criar um modelo baseado em Prolog (Clocksin and Mellish 1984; Sterling and Shapiro 1994) do XMLaw. Isto porque os termos do LGI são termos Prolog e como as primitivas do LGI são de baixo nível, pouco poderia ser reutilizado para dar a semântica dos elementos do XMLaw no LGI. O mapeamento pode ser utilizado se existe uma necessidade de uma arquitetura descentralizada, tal como a presente em LGI. Também é importante destacar que embora propriedades globais possam ser implementadas em LGI, se a solução geral apresentada em (Minsky 2005) é usada como ilustrado no Código 25, não estaria sendo feito o uso da descentralização disponibilizada pelo LGI. Por outro lado, também é possível especificar leis que façam uso de informações bem específicas do domínio para sincronizar o estado dos

controladores somente quando for necessário. Entretanto, esta abordagem introduz complexidade para a especificação das leis e ainda traz para a lei conceitos de sincronização de sistemas distribuídos.

O modelo flexível de eventos presente no XMLaw pode contribuir para a construção de modelos de governança de leis mais preparados para acomodar mudanças. Isto é importante, especialmente quando se considera que o próprio XMLaw pode evoluir para incorporar mudanças em prol de expressividade e facilidade de especificação das leis. Um exemplo desta evolução seria a inclusão da noção de normas conforme descrito em (Garcia-Camino, Noriega et al. 2005).

Em resumo, o XMLaw é uma abordagem alternativa às abordagens atuais para a especificação de leis de interação em sistemas multi-agents abertos. As suas principais características são as abstrações de alto nível e um modelo de comunicação entre seus elementos baseado em eventos.