4
Infra-estrutura de Implementação: M-Law

Os avanços da tecnologia de agentes dependem do desenvolvimento de modelos, mecanismos e ferramentas para a construção de sistemas de qualidade. O projeto e a implementação destes sistemas ainda é caro e sujeito a erros. Os frameworks de software lidam com esta complexidade reutilizando projetos e implementações de software validadas na implementação de uma família de aplicações. Um framework orientado a objetos é uma aplicação semi-acabada e reutilizável que pode ser especializada para produzir aplicações customizadas (Fayad, Schmidt et al. 1999).

Neste capítulo, apresenta-se o projeto e a aplicação de um *middleware* para a governança de sistemas multi-agentes chamado M-Law (Paes, Almeida et al. 2004; Paes, Carvalho et al. 2004; Paes, Gatti et al. 2006; Carvalho 2007; Paes, Carvalho et al. 2007). M-Law foi projetado como um *framework* orientado a abjetos e os seus *hotspots* permitem conectar infra-estruturas existentes de agentes, mudar o mecanismo de comunicação e adicionar novas funcionalidades. O M-Law 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 8).

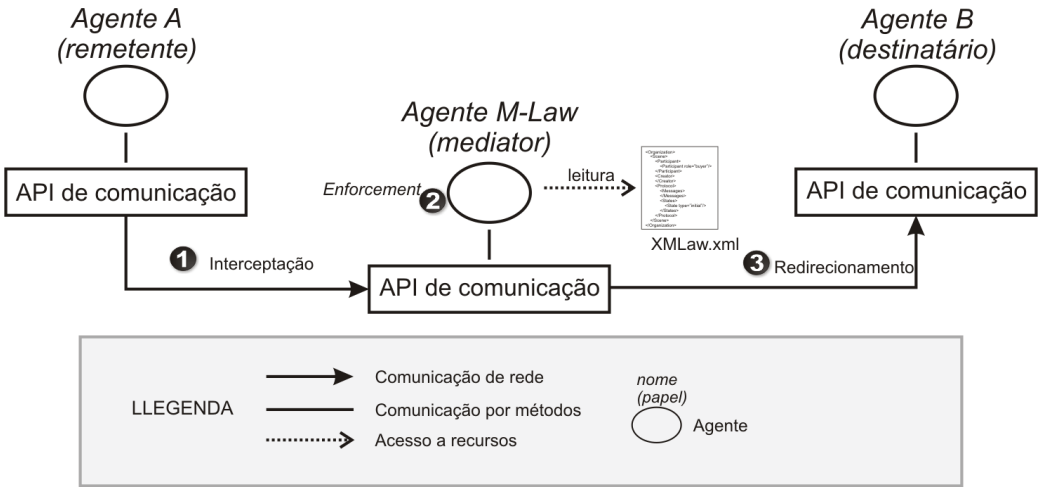

Figura 8 – Arquitetura do Mediador M-Law

Para que o *middleware* M-Law pudesse prover os meios para efetivamente dar suporte à linguagem XMLaw e sua evolução, ele foi desenvolvido com cinco módulos (Figura 9): *communication*, *agent*, *component*, *enforcement* e *event*.

O módulo de *communication* é utilizado para enviar e receber mensagens entre os agentes e o mediador M-Law. O módulo *agent* contém as classes que os desenvolvedores de agentes devem utilizar para implementá-los. Este módulo por sua vez, utiliza o módulo de *communication*. O módulo de *enforcement* é utilizado para mapear elementos de lei descritos usando a linguagem XMLaw para o modelo de execução interno ao M-Law. O modulo *event* 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 *component* 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.

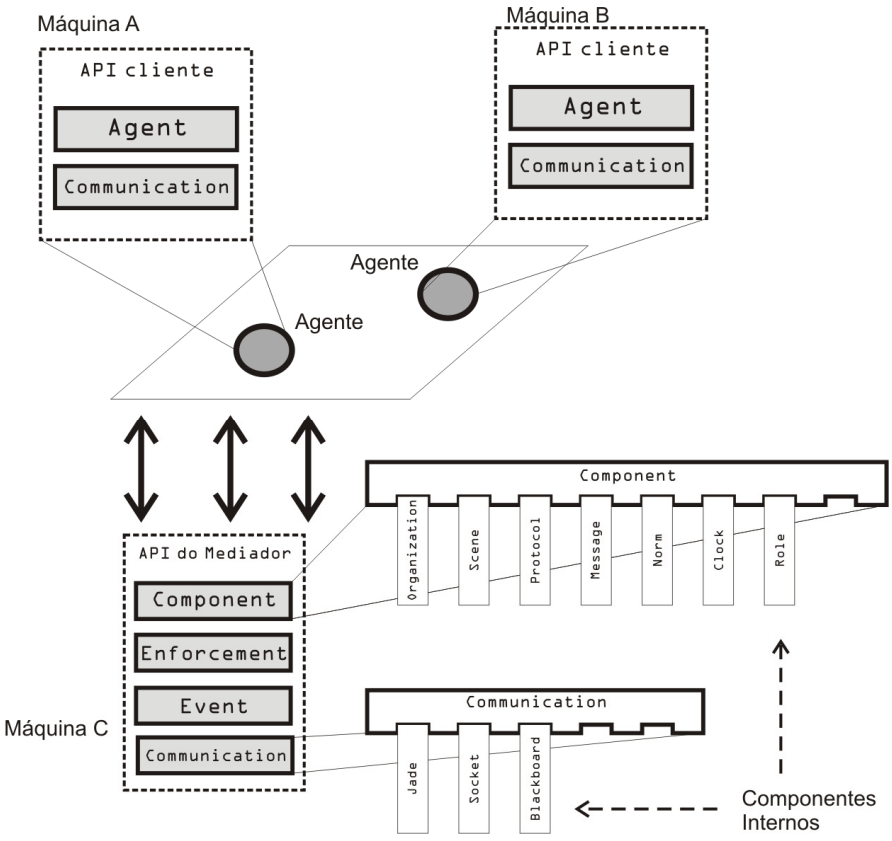

Figura 9 – Componentes Principais do M-Law

Um agente cliente desta solução utiliza os módulos *agent* e *communication*. A máquina de interpretação e *enforcement* de leis, chamada de Mediador, engloba

os outros três módulos, *enforcement*, *event* e *component*, e também utiliza o módulo *communication*. Os módulos de *enforcement*, *event* e *component* 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 *component*. O módulo *component* contém um conjunto de classes que o mediador implementa para representar o comportamento dos elementos da linguagem XMLaw. Por exemplo, a Figura 10 detalha o elemento Scene (cena) em XMLaw e o conjunto de classes que implementam o seu comportamento. Na Figura 10, 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 dos módulos *component* e *enforcement* 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 10, 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 *component*.

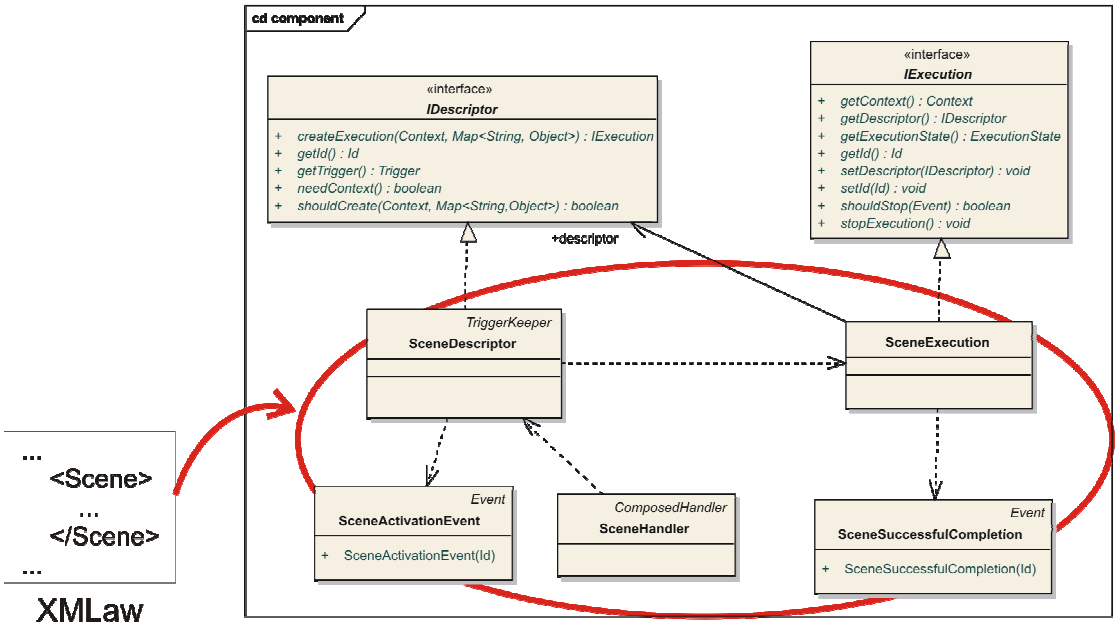

Figura 10 – Projeto do Elemento *Scene*

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.

Para utilizar o M-Law é necessário a execução de quatro passos principais. Primeiro, é preciso escrever as leis utilizando a linguagem XMLaw. Depois o mediador precisa ser iniciado através da execução dos arquivos de script fornecidos com o M-Law. O terceiro passo é indicar ao mediador a localização do arquivo de lei. Finalmente o quarto passo é iniciar os agentes da aplicação.

Desenvolvedores de agentes podem querer estender a classe Agent oferecida pela API cliente M-Law. 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 11 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 M-Law. A implementação padrão de M-Law utiliza Jade (Bellifemine, Poggi et al. 1999) como ambiente de comunicação e desenvolvimento básico de agentes de software.

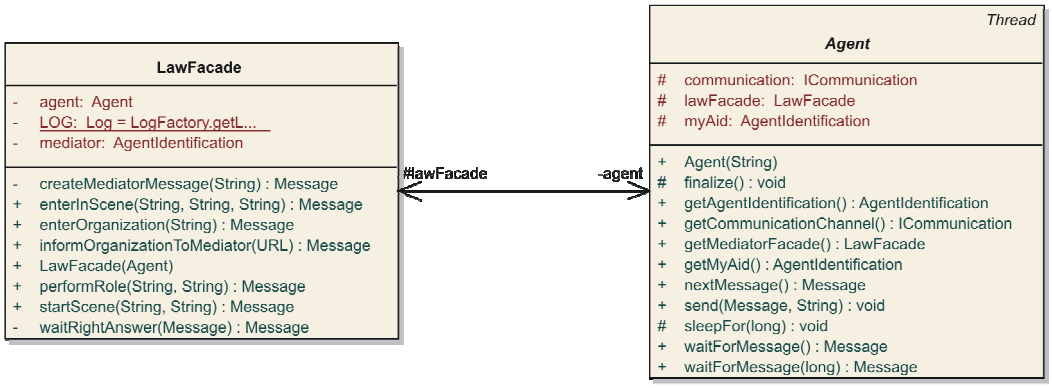

Figura 11 – API cliente: classes LawFacade e Agent

4.1.Ciclo de Vida do Mediador MLaw

Esta seção tem como objetivo explicar as alterações necessárias no mediador M-Law para interpretar leis de interação com pontos de extensão de interação. Conforme explicado, o mediador M-Law 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 12). 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 *enforcement*. Este interpretador verifica se a implementação da lei XMLaw está bem formada e é 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.

Em razão da proposta de pontos de extensão de interação (Carvalho 2007), 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.

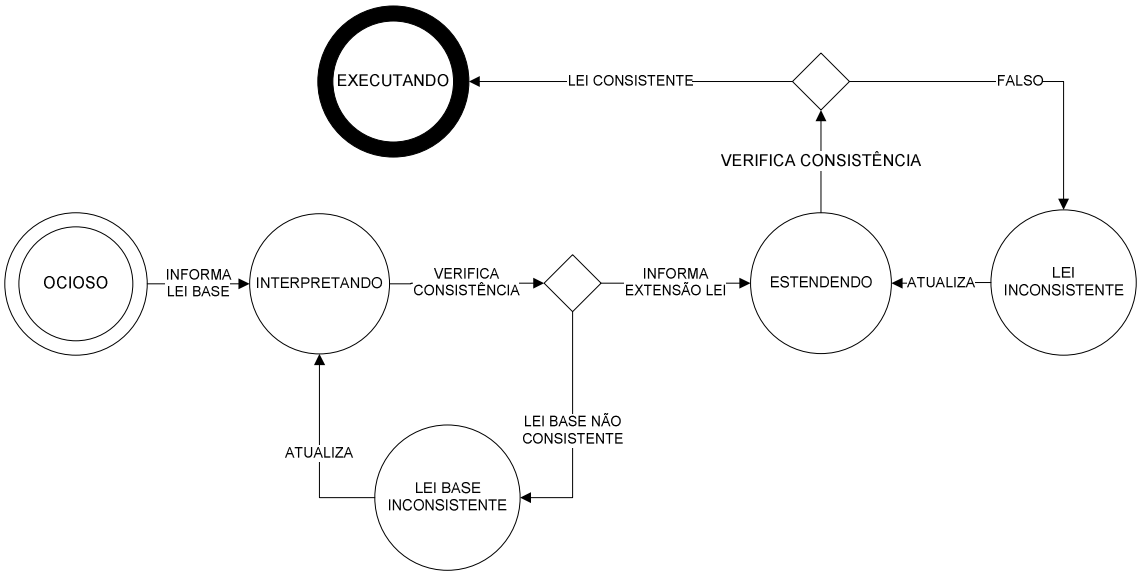

Figura 12 – Ciclo de Vida do Mediador M-Law

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 M-Law 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 10.

4.2.Arquitetura e Implementação

O M-Law é composto por sete pacotes principais, conforme pode ser visto na Figura 13. Alguns destes pacotes são utilizados pelo mediador, enquanto outros

podem ser utilizados pelos agentes da aplicação. A Figura 14 mostra como esta divisão ocorre. No restante desta seção, cada um dos pacotes é descrito e, quando necessário, mostra-se um diagrama de classes contendo as principais classes de cada pacote.

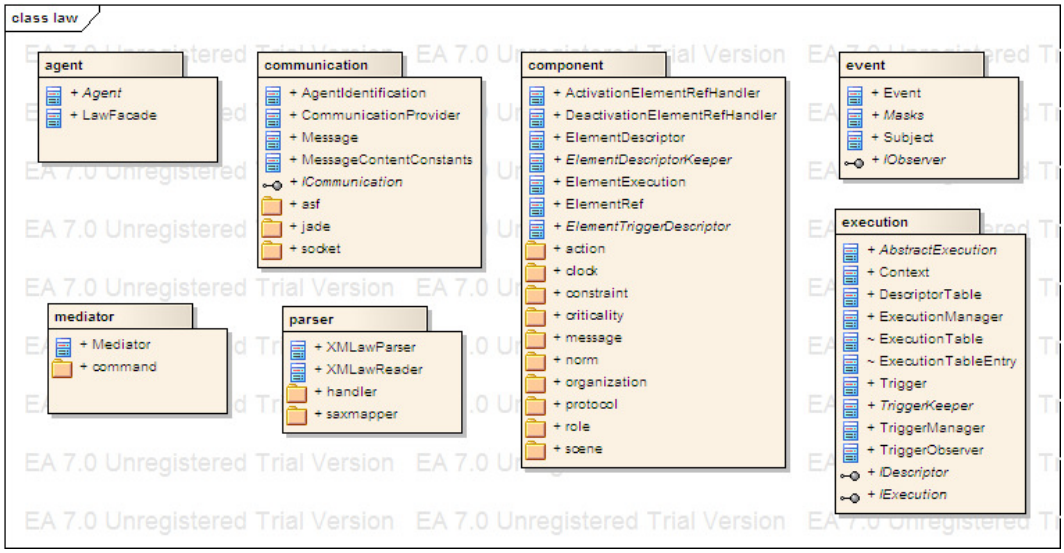

Figura 13 – Diagrama de Pacotes

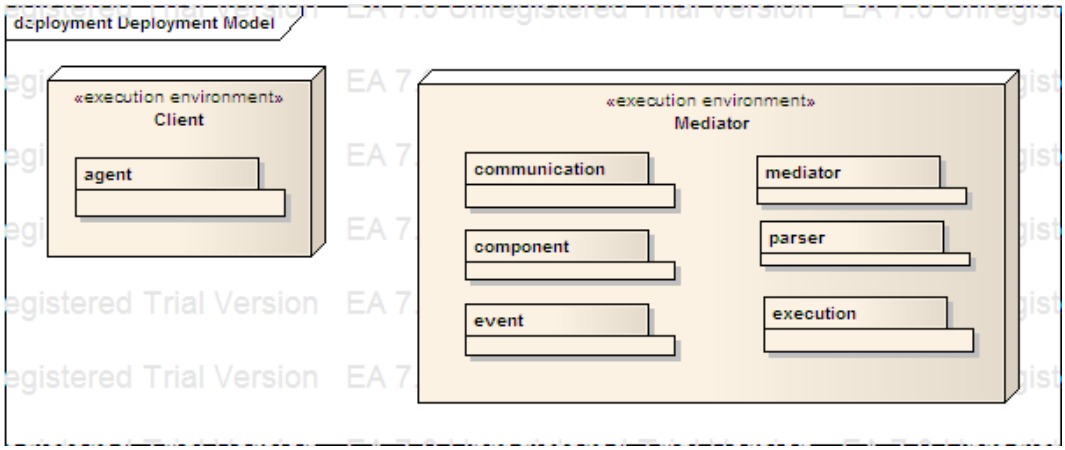

Figura 14 – Diagrama de Deployment

*4.2.1.*Pacote *agent*

Existem duas maneiras de criar um novo agente: construindo-o desde o início ou usando a classe *Agent* disponibilizada por este módulo. Esta classe fornece métodos que auxiliam no envio e no recebimento de mensagens e é completamente integrada com a abordagem de leis. Caso o agente a ser desenvolvido já herde de alguma outra classe, ainda assim é possível utilizar a

classe *Agent* provida pelo módulo através da técnica de delegação de chamada de métodos e, assim, alcançar os benefícios fornecidos pela classe Agent.

O envio e o recebimento de mensagens utilizando a classe Agent é realizado com uma chamada de método. A classe *Agent* possui uma instância da classe *ICommunication* como um de seus atributos. Este atributo representa o canal por onde o agente envia e recebe mensagens. Além disso, a classe *Agent* também possui uma referência para a classe *LawFacade*. Esta classe contém métodos para operações de leis, tais como entrar em uma cena e desempanhar um determinado papel. A Figura 15 mostra as classes *Agent* e *LawFacade*.

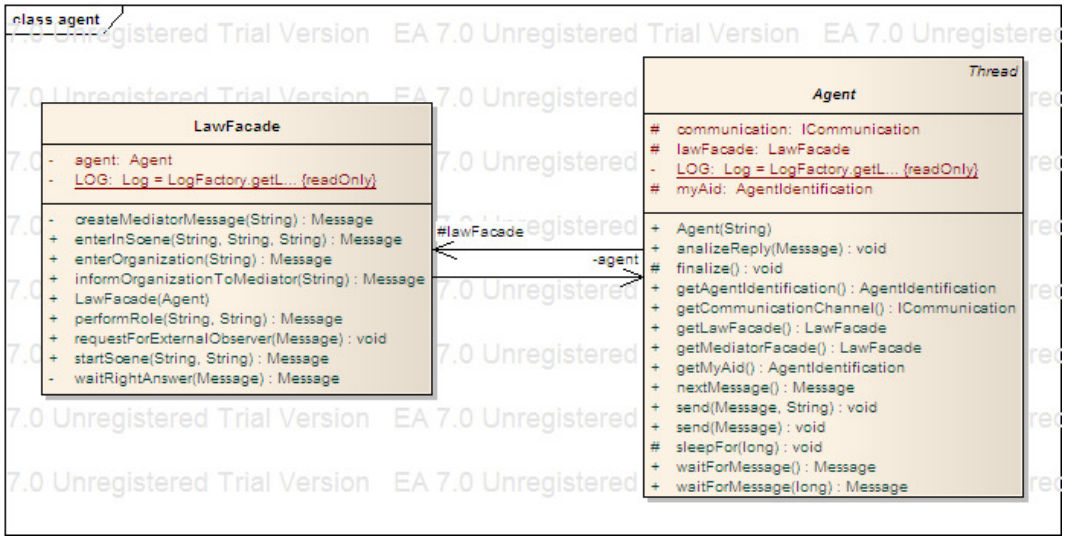

Figura 15 – Diagrama de Classes do Pacote *agent*

4.2.2.Pacote *communication*

Agentes podem utilizar diferentes formas de comunicação. Eles podem utilizar uma infra-estrutura baseada nos padrões SOAP (Box, Ehnebuske et al. 2000), podem implementar infra-estruturas proprietárias utilizando sockets, ou mesmo padrões de comunicação entre agentes tais como os definidos pela FIPA(Fipa 2002). Cada aplicação possui diferentes requisitos de performance, flexibilidade, interoperabilidade, entre outros. Uma camada de comunicação deve se adequar a esses requisitos. Desta forma, a camada de comunicação proposta fornece uma interface que permite que as aplicações mudem a implementação de determinadas partes para que os requisitos específicos de cada aplicação possam ser atendidos.

A interface *ICommunication* (Figura 16) define métodos para o envio e recebimento de mensagens. Por se tratar apenas de uma interface, as

implementações deste método não são fornecidas por ICommunication e, portanto, precisam ser providas por implementações da interface para que as funcionalidades sejam fornecidas. O método de envio de mensagem é o método *send(Message msg)*. O comportamento esperado deste método é o envio de uma mensagem ao destinatário especificado na mensagem. Para o recebimento de mensagens, três métodos foram definidos, sendo dois métodos "bloqueantes" e um método "não bloqueante". O método *waitForMessage():Message* bloqueia a execução do programa que invocou este método até o recebimento de alguma mensagem. Quando uma mensagem for recebida, a chamada é desbloqueada e a mensagem é retornada. Uma versão um pouco modificada deste método é o método *waitForMessage(long milliseconds)*. Este método funciona de forma análoga ao *waitForMessage()* exceto que ele bloqueia o programa que o chamou até que uma mensagem seja recebida ou que o tempo especificado no parâmetro tenha se esgotado. Desta forma, este método retorna a mensagem recebida caso ela tenha chegado ou *null* caso nenhuma mensagem tenha chegado e o tempo tenha expirado. Os métodos *wait* são bloqueantes uma vez que o programa que os invoca têm a sua execução bloqueada. Porém, esta interface define um método denominado *nextMessage():Message*. Este método, ao contrário dos outros dois, não bloqueia a execução, mas retorna a próxima mensagem que ainda não foi lida, ou *null* caso nenhuma mensagem tenha chegado até o momento da invocação do método. Este comportamento do método nextMessage sugere que implementadores da interface *ICommunication* implementem uma estrutura de fila para que todas as mensagens recebidas sejam armazenadas e quando o método *nextMessage* for invocado, a primeira mensagem da fila seja retornada.

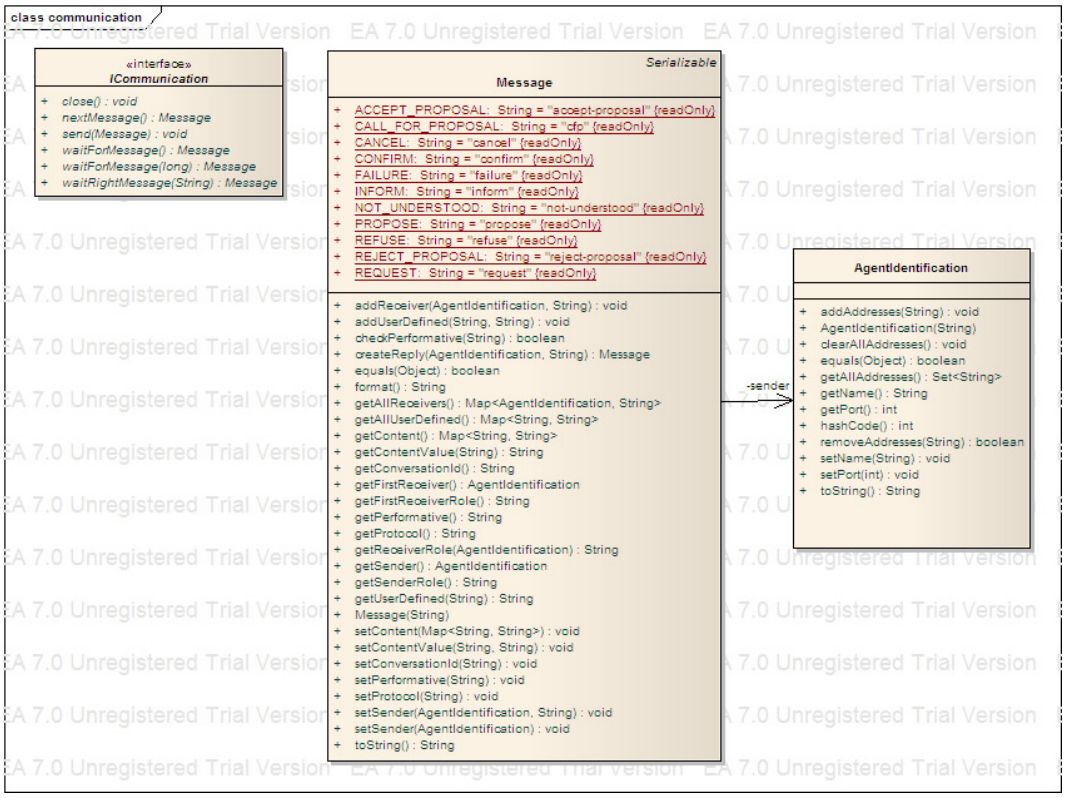

Figura 16 – Diagrama de Classes do Pacote *communication*

A Figura 17, mostra como o framework JADE (Bellifemine, Poggi et al. 1999) foi utilizado para prover as funcionalidades definidas pela interface da camada de comunicação. O JADE implementa um mecanismo de comunicação compatível com os padrões definidos pela FIPA. Este mecanismo é acessível de forma relativamente transparente através da classe *jade.core.Agent*, provida pela implementação JADE. Então, a classe *JadeCommunication* reutiliza a implementação desse mecanismo de comunicação JADE através da delegação dos métodos da interface *ICommunication* para uma instância da classe *jade.core.Agent*.

Também foram realizadas outras implementações da camada de comunicação, sendo que uma utilizou *sockets* e outra o framework ASF (Silva 2004).

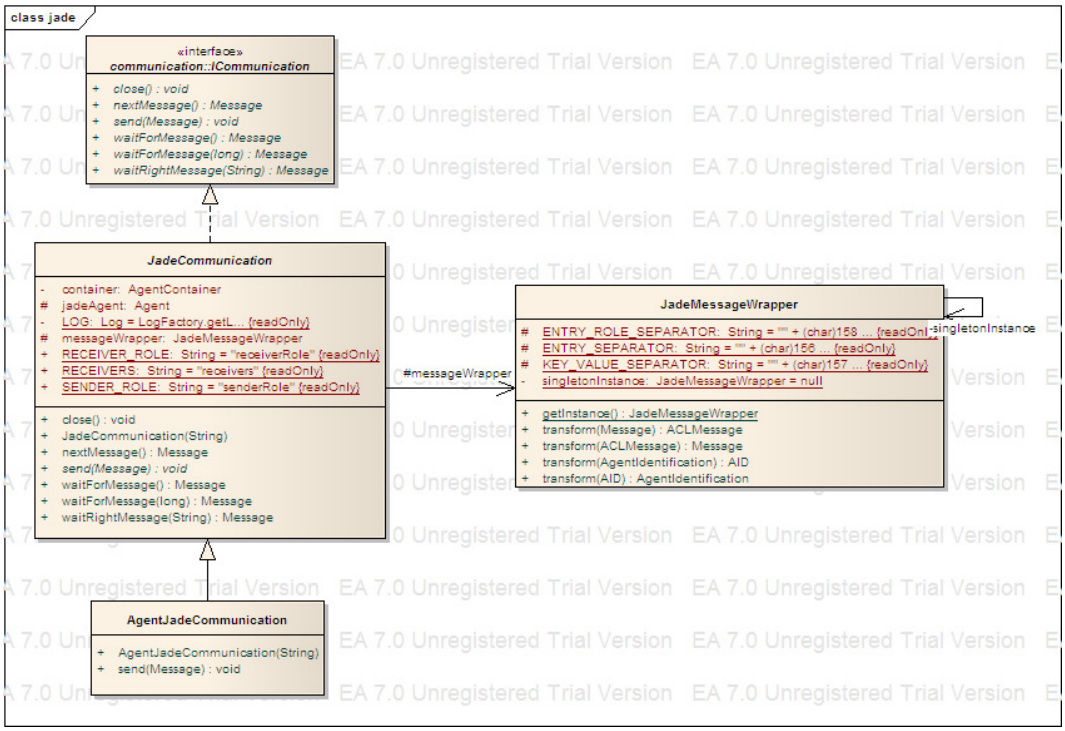

Figura 17 – Diagrama de Classes do Pacote *jade*

*4.2.3.*Pacote e*vent*

Este módulo fornece a implementação para que a comunicação entre os elementos do XMLaw seja baseada em eventos. A classe *Event* representa um evento no sistema. Um evento possui principalmente: um identificador único, o identificador do elemento do xmlaw que originou o evento, um conjunto de variáveis que podem ser adicionadas ao evento e o tipo. O evento pode ser de qualquer um dos tipos definidos na classe *Masks*. A estrutura de notificação e subscrição dos eventos é baseada no padrão *Observer* (Gamma, Helm et al. 1995).

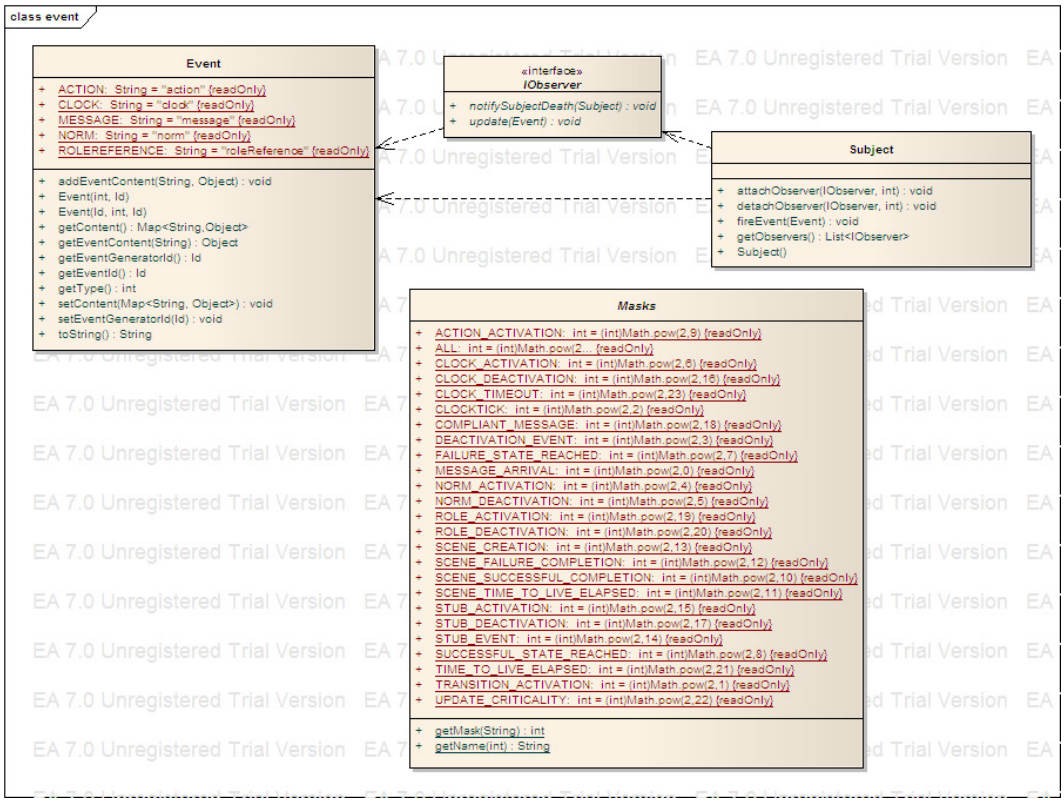

Figura 18 – Diagrama de Classes do Pacote *event*

4.2.4.Pacote *execution*

Este pacote contém as classes que implementam a lógica de execução dos elementos. O diagrama de classes é exibido na Figura 19.

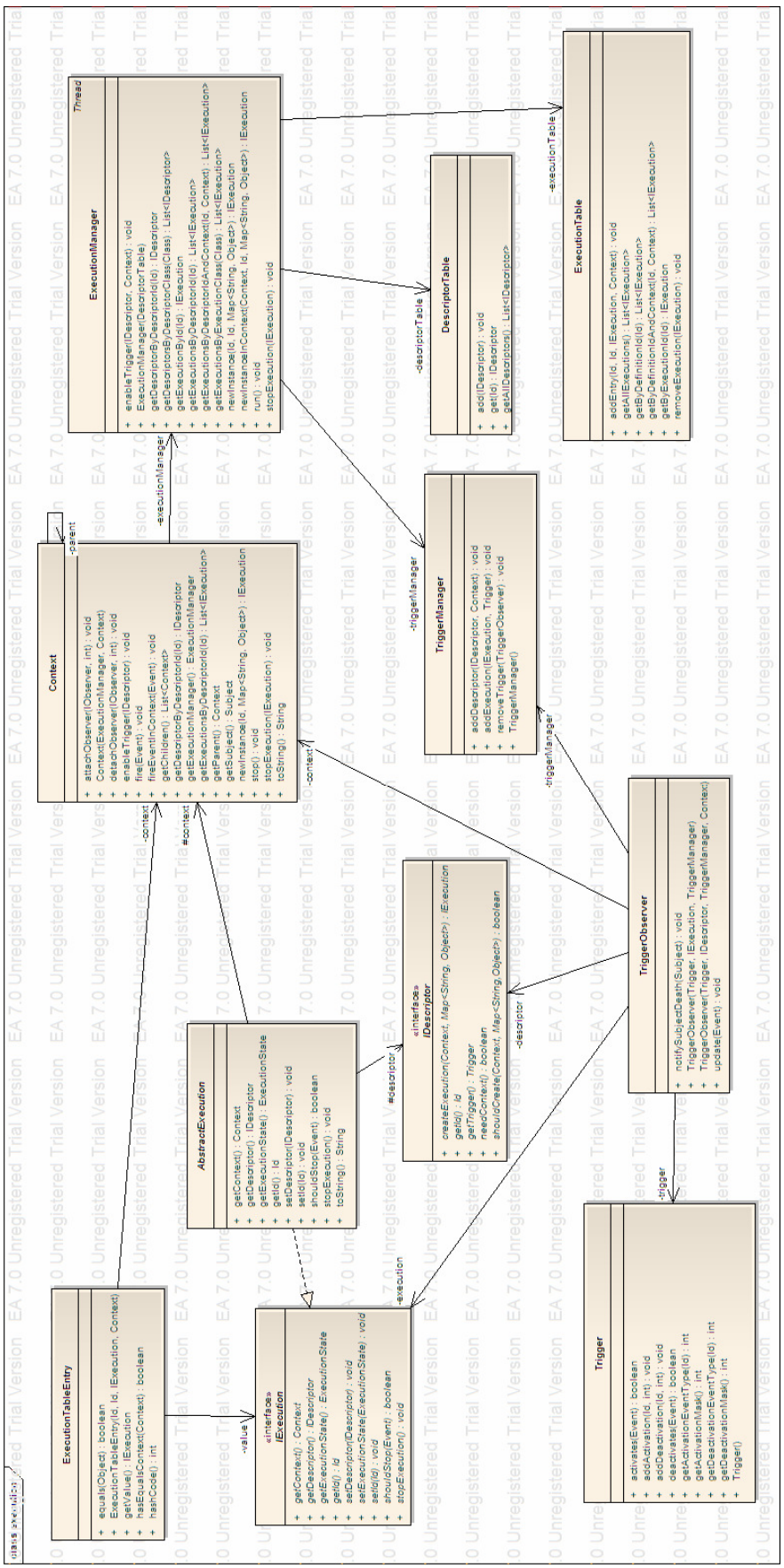

Figura 19 – Diagrama de classes do Pacote *execution*

4.2.5.Pacotes *component*, *mediator* e *parser*

É no pacote *component* que os elementos do XMLaw estão localizados. Cada elemento é um pacote contido dentro do pacote *component*. O pacote *parser* implementa um analisador léxico e sintático para da linguagem XMLaw. Atualmente existem duas implementações deste pacote: uma para a sintaxe em XML e outra para a sintaxe proposta nesta tese. Finalmente, o pacote *mediator* contém o agente mediador. É este agente que recebe as mensagens redirecionadas para ele e inicia a interpretação das mensagens em função da lei.

4.3.Trabalhos relacionados ao Middleware

4.3.1.AMELI

Para dar suporte à especificação de uma instituição eletrônica (Esteva 2003) foi desenvolvido um middleware chamado AMELI (Esteva, Rosell et al. 2004) e um editor gráfico que permite a especificação através da combinação de elementos gráficos e textuais. Além da especificação, este editor também permite a verificação de algumas propriedades, mais especificamente: integridade, que permite verificar as auto-referências entre os elementos da linguagem estão corretas; *liveness*, que permite verificar se algum agente estará bloqueado na *performative structure* e que o estado final das cenas sempre é alcançado, corretude do protocolo; e finalmente a verificação se os agentes podem cumprir as normas especificadas.

O middleware verifica se a especificação da instituição eletrônica está sendo cumprida durante a interação dos agentes. Este mecanismo utiliza o JADE (Bellifemine, Poggi et al. 1999) como camada de comunicação e é composto de 3 elementos principais:

- *Institution Manager* é o elemento responsável por autorizar os agentes a entrar na instituição, iniciar o sistema e controlar a participação dos agentes entre as cenas. Uma vez autorizados, é associado a cada agente um agente *Governor*.
- O *Scene Manager* é responsável por controlar a execução de uma cena, identificando quais os agentes podem participar da cena.

- Cada *Governor* é responsável por mediar a comunicação o agente e o resto da instituição. Eles dão aos agentes toda a informação de que eles precisam para participar de uma instituição.

Um dos pontos importantes que vale ser ressaltado nesta abordagem é que o protocolo é especificado sob um ponto de vista global. Desta forma, é possível saber qual o exato estado de execução deste protocolo. Na abordagem, os protocolos são especificados através de uma máquina de estados finita, onde não existe a idéia de concorrência. Desta forma, os *governors* (responsáveis por atualizar o estado da execução do protocolo ou cena) precisam se coordenar para que somente um deles atualize o estado do protocolo por vez. Além disso, uma vez que a atualização do estado de execução do protocolo é feita, todos os governors precisam atualizar a sua visão do estado do protocolo para poder executar alguma mudança de estado. Uma transição que era válida em um estado anterior pode não ser mais válida no novo estado.

Em uma possível solução para esta questão, poderia existir um elemento centralizador que mantém o estado atual da execução de uma cena e, portanto, quando cada *governor* precisar atualizar o estado, é preciso "travar" o estado atual, solicitar uma cópia deste estado, realizar a interpretação da transição de estados, possivelmente modificar o estado e enviar de volta o novo estado destravando-o. Desta forma, a cada mensagem que o agente envia, existem mais duas, uma para a recuperação e outra para a atualização do estado. Somente este fato não ocasiona muitos problemas em termos de ordem de complexidade do algoritmo. Entretanto, introduz-se um único ponto de falha no sistema. Ou seja, caso este elemento que detém as informações sobre o estado da cena se torne indisponível, todos os agentes que participam da cena passam a não poder se comunicar.

Na abordagem, existe um elemento centralizador que mantém o estado atual da execução de uma cena. Porém, várias cópias do estado atual estão distribuídas entre os *governors*. Esta solução elimina o problema de se ter apenas um único ponto de falha, porém, introduz um custo de comunicação relacionado a manter os estados dos *governors* atualizados. Cada mensagem enviada pelos agentes que muda o estado da execução do protocolo tem como conseqüência a atualização do estado deste protocolo em todos os *governors*. Ou seja, a cada mensagem existe o custo de um *broadcast* para atualização dos estados. Logo, se *x* mensagens forem

enviadas entre os agentes, o número total de mensagens do sistema é de aproximadamente: $x \times n$, onde n é o número total de agentes em uma cena. Este alto custo de coordenação torna esta solução imprópria para sistemas onde existam um grande número de agentes interagindo em uma mesma cena.

4.3.2.S-Moise+

O S-Moise+ (Hübner 2003; Hübner, Sichman et al. 2006) propõe uma arquitetura (Figura 20) com algumas similaridades em relação à utilizada pelo MLaw. A Tabela 9 compara as duas arquiteturas.

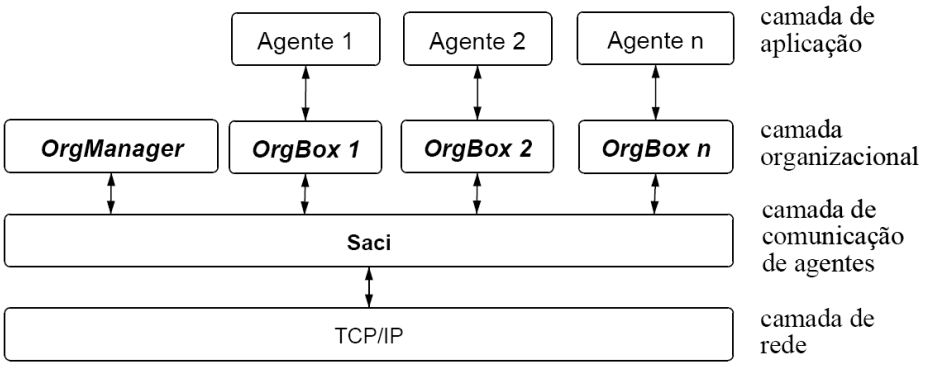

Figura 20 – Arquitetura do S-Moise+

S-Moise+	M-Law	Comentário
OrgBox	Módulo Agent	O OrgBox é uma interface que os agentes utilizam para acessar a organização e os demais agentes. O Módulo Agent no XMLaw possui esta mesma finalidade
Saci	Módulo Communication	Todos os dois elementos fornecem uma abstração para a camada de comunicação. Nas duas abordagens é possível trocar a camada de comunicação.
OrgManager	Mediador do M-Law	Ambos os elementos possuem a função principal de verificar se a especificação está coerente com a execução.

Tabela 9 – Comparação entre os elementos da arquitetura do M-Law com o S-Moise+

4.4.Considerações Finais e Exemplos de Utilização do M-Law

A primeira versão do M-Law foi criada em 2004 (Paes, Almeida et al. 2004; Paes, Carvalho et al. 2004) e desde então vem sofrendo sucessivas evoluções. Em uma dissertação de mestrado defendida em 2005 (Paes 2005), o M-Law foi detalhadamente documentado. No capítulo 05 desta dissertação, encontra-se um guia de desenvolvimento de agentes utilizando o framework. Optou-se por reapresentar o M-Law como contribuição desta tese porque ele passou por consideráveis modificações, dentre as quais é possível destacar:

- Mudança da interpretador para a nova sintaxe da linguagem – originalmente as leis eram escritas em XML. Entretanto a ausência de uma ferramenta de auxílio a especificações das leis tornava a tarefa de escrever as leis bastante suscetível a erros além de tornar a leitura mais difícil. Assim, optou-se por criar uma nova linguagem de especificação. O impacto desta mudança no framework foi a construção de uma nova camada de *Handlers* e *Descriptors* que reconhecessem a nova sintaxe.
- Adição de novos eventos – a inserção do agente detector de falhas, fez com que o M-Law precisasse reconhecer esse agente como um agente especial e consequentemente gerenciasse toda a geração e propagação de eventos relacionados a ele.

No restante desta seção, mostra-se a utilização do M-Law em vários domínios diferentes.

4.4.1.Protótipo do Sistema Especial de Liquidação e Custódia do Banco Central (SELIC)

O SELIC foi escolhido por se tratar de fato de um sistema distribuído de governança (Carvalho 2007). Atualmente em sua versão de produção, mais de 4000 instituições financeiras utilizam este aplicativo para negociar títulos públicos e solicitar a efetiva liquidação e alteração de custódia. As instituições financeiras implementam a abstração de agentes de software autônomos, que podem entrar e sair do sistema e que se comunicam via troca de mensagens; caracterizando

portanto um cenário de sistema aberto. O sistema SELIC foi mapeado diretamente para um sistema multi-agente governado por leis, pois neste contexto existe uma instituição reguladora que funciona como mediadora nas interações de compra e venda de títulos, e também em outras operações relacionadas a instituições financeiras. Em (Carvalho 2007), mostrou-se uma estratégia para a especificações das leis utilizando XMLaw neste domínio, e o M-Law foi utilizado como plataforma de implementação.

4.4.2.Teste de Integração

O trabalho relatado em (Rodrigues, Carvalho et al. 2005) possui como principal foco a estratégia de como escrever casos de teste e receber os resultados da execução destes testes. O sistema é considerado como uma composição de vários subsistemas distribuídos e cada subsistema é visto como um agente. Neste contexto, o XMLaw foi utilizado para especificar o comportamento esperado do sistema e o M-Law foi utilizado para monitorar o comportamento que efetivamente estava ocorrendo. O M-Law foi integrado com outros software que permitiam a escrita dos casos de testes e a geração de relatórios. Os resultados deste experimento ilustraram a capacidade de integração do M-Law com outras soluções de software.

4.4.3.Criticalidade de Agentes

A técnica de replicação de agentes é definida como a ação de criar uma ou mais cópias de um ou mais agentes de um sistema multi-agente. Esta técnica é considerada por muitos como uma maneira de implementar tolerância a faltas em sistemas multi-agentes. O experimento realizado em (Gatti, Lucena et al. 2006) reaproveitou a estrutura de monitoramento do M-Law para identificar a variação de criticalidade dos agentes. A medida que os agentes iam se tornando críticos, o M-Law envia uma mensagem para uma estrutura de replicação que se encarrega de criar as réplicas. Para que este comportamento fosse alcançado, foi preciso modificar o XMLaw para a inclusão de novos elementos. Estes novos elementos foram adicionados através do módulo *component* do M-Law.