2
Fidedignidade de Sistemas Multiagentes Abertos

Neste capítulo serão apresentados os principais conceitos sobre *software dependability* [16][17], e sua importância para sistemas multiagentes abertos. Deve-se ressaltar que o termo *dependability* será utilizado neste trabalho como **fidedignidade** que pode ser encontrado em dicionários como "qualidade daquilo ou de quem é fidedigno", e fidedigno, por sua vez, significa "digno de fé, merecedor de crédito"

Considerando que um sistema de software é uma entidade que interage com outras entidades, que são sistemas, incluindo hardware, software, pessoas e o mundo físico propriamente dito. Que a função de um sistema é para o que o sistema foi feito. Que o comportamento do sistema é o que o sistema faz para implementar suas funções. E, finalmente que um serviço disponível do sistema é o seu comportamento percebido pelo seu usuário, isto é, outro sistema. A fidedignidade de sistemas de software é a capacidade que o mesmo tem de entregar serviços corretos de forma justificável. E um serviço correto é aquele que implementa a função do sistema.

Para que um sistema de software seja fidedigno e atenda às definições especificadas, de acordo com o domínio e os requisitos do sistema, este deve estar em conformidade com uma série de atributos que serão descritos na próxima seção. A partir das definições dos conceitos gerais de fidedignidade, a fidedignidade de SMA's abertos será descrita na seção 2.2 do ponto de vista de tolerância a falhas e de governança, seguidas da apresentação de duas abordagens relacionadas a dois atributos de fidedignidade: confiabilidade e disponibilidade, as quais são o foco deste trabalho.

2.1.
Fidedignidade e seus atributos[3]

A principal noção de fidedignidade de sistemas de software compreende três conceitos: seus atributos, suas ameaças e o meio pelo qual é alcançada.

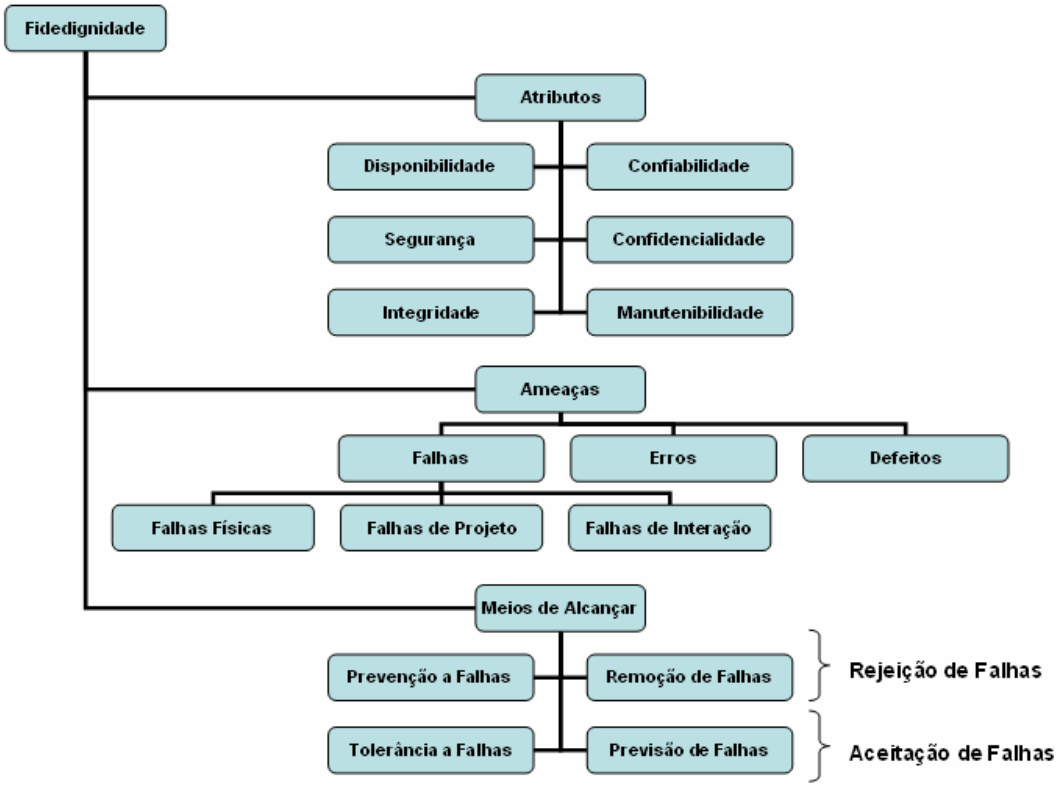

Figura 1 - Fidedignidade e seus atributos

Os atributos de fidedignidade são:

- Disponibilidade: é a capacidade que o sistema possui de entregar um serviço correto dado um tempo;
- Confiabilidade: é a capacidade que o sistema possui de entregar continuamente um serviço correto em um período de tempo;
- Confidencialidade: é a ausência de informação não autorizada aberta/disponível.
- Integridade: é a ausência de alterações de sistema impróprias;
- Manutenibilidade: é a capacidade de efetuar modificações e reparos;
- Segurança: é o conjunto dos seguintes itens de forma paralela: (i) a disponibilidade somente para usuários autorizados, (ii)

[3] Todos os conceitos definidos nesta seção foram retirados de [16] e [17] e tratam de fidedignidade de sistemas de software de uma forma geral.

confidencialidade, e (iii) integridade com um objetivo impróprio não autorizado.

As ameaças para a fidedignidade de sistemas consistem em erros, faltas e falhas. Uma falha no sistema ocorre quando um serviço que é entregue difere do serviço correto. Um erro é o estado do sistema que levou ou pode levar a uma falha. E, finalmente, uma falta é a causa suposta ou real de um erro. Uma falta está ativa/latente quando ela produz erros, senão diz-se que ela está dormente.

Dado que é necessário especificar os requisitos de acordo com os atributos de fidedignidade, e dado os tipos de ameaças que existem para se alcançar a fidedignidade de um sistema, faz-se necessário saber quais são os meios para se alcançá-la efetivamente.

Como exibido de forma sucinta na Figura 1, existem quatro técnicas que devem sem empregadas ou implementadas para se alcançar a fidedignidade:

Técnica	Descrição
Prevenção de falhas	Provê meios de prevenir a ocorrência ou introdução de falhas
Remoção de falhas	Provê meios de reduzir as falhas em número e severidade
Tolerância a falhas	Provê meios de evitar defeitos em serviços na presença de falhas
Previsão de falhas	Provê meios de estimar o número presente, a incidência futura, e as conseqüências das falhas

Tabela 1- Técnicas de Tolerância a Falhas

Como exposto na Tabela 1, as técnicas de prevenção de falhas e tolerância a falhas objetivam prover a habilidade de entregar um serviço que pode ser confiável, enquanto que as de remoção de falhas e previsão de falhas objetivam alcançar a confiança no sentido de que as especificações funcionais e de fidedignidade são adequadas e que o sistema as implementa.

Este trabalho, entretanto, foca somente em duas técnicas para diferentes tipos de erros. A técnica de tolerância a falhas, que é a entrega de serviços corretos na presença de falhas, será utilizada para determinados tipos de falhas. E a técnica de prevenção de falhas será utilizada para outros tipos de falhas, considerando sistemas abertos.

Desta forma, na próxima seção, serão apresentados os tipos de falhas que podem ocorrer em um sistema multiagente aberto, algumas técnicas de tolerância existentes na literatura, e as técnicas utilizadas neste trabalho para implementar os

meios de se alcançar a fidedignidade no que diz respeito à disponibilidade e confiabilidade.

2.2. Tolerância a falhas em SMA's Abertos

Existem quatro características principais que fazem com que sistemas multiagentes (SMA's) precisem de mecanismos que garantam sua fidedignidade [19]. Primeiramente, um SMA é composto de agentes autônomos, capazes de tomar ações sem o conhecimento do ambiente, alterando o estado do mesmo. Segundo, SMA's possuem nenhum ponto centralizado de controle, pela própria natureza distribuída do sistema. A terceira característica é o dinamismo pelo qual o ambiente passa constantemente. Isto dificulta a capacidade de se manter o estado do mesmo no caso de recuperá-lo em uma necessidade de restauração. E, finalmente, os agentes colaboram entre si o tempo inteiro para alcançar seus objetivos, de forma coordenada ou não, e em sistemas abertos ou não. Logo, a falha de um deles se propaga rapidamente pelo sistema caso não seja tratado corretamente.

Todas as características descritas contribuem para um estado de defeito. E, como mencionado anteriormente, um defeito ocorre quando o sistema produz resultados que não atendem às especificações. Portanto, já que uma falha é a suposta ou real causa de um erro que, por sua vez, pode levar o sistema para um estado de defeito, faz-se necessário identificar os tipos de falhas que poderiam ocorrer e tratá-las com um mecanismo de tolerância a falhas. De forma que, se elas efetivamente ocorrerem, o sistema poderá ser levado para um estado correto e não para um estado de defeito. E ainda como um fator agravante é necessário ressaltar que, quando uma falha ocorre em um SMA, as interações dos agentes podem propagá-la pelo sistema de uma forma imprevisível.

Para que seja possível aplicar alguma técnica de tolerância a falhas, faz-se necessário identificar os tipos de falhas. Assim, dentre os tipos de falhas que podem ocorrer em sistemas multiagentes, encontra-se: (i) estados não previstos, isto é, o sistema não sabe manipular um estado particular e a equipe de teste, quando existente, não realizou testes para estes estados; (ii) falhas de processador, que podem ser um travamento no sistema (permanente/silenciosa), ou a falta de

recursos de sistemas; (iii) falhas de comunicação que podem ocorrer por uma diminuição de velocidade, defeitos ou outros problemas de comunicação; (iv) um ou mais agentes podem entrar em *deadlock*; (v) falhas de interação intencionais ou maliciosas que poderiam ser originadas por um agente malicioso que pretendesse destruir ou prejudicar o sistema; e ainda, (vi) comportamento emergente do sistema não esperado, que pode ser proveitoso ou prejudicial.

Como exibido na Figura 1 na seção anterior, existem três grupos de técnicas de tolerância a falhas que podem ser aplicadas: tolerância de falhas físicas, tolerância de falhas de projeto e tolerância de falhas de interação. Este trabalho foca na primeira e na terceira, já que estas resolvem as falhas (ii), (iii), (iv), (v), (vi), listadas anteriormente.

Para que estas falhas não levem o sistema para um estado de defeito, existem algumas abordagens - [20], [21], [22], [23], [2], [4], [24] - quanto à tolerância a falhas em SMA's que podem ser classificados em dois grupos: a centrada no agente, e a centrada no sistema. A abordagem centrada no agente trata a tolerância a falhas e/ou tratamento de exceções internamente no agente. E a centrada no sistema implementa algum tipo de monitoramento e restauração em uma entidade separada de software [19]. Dentro deste contexto, existe uma técnica que vem sendo pesquisada nos últimos anos relacionada a replicação de agentes. Esta técnica utiliza as duas abordagens mencionadas: centrada no agente e centrada no sistema.

A técnica de replicação já é antiga para os grupos de pesquisa de tolerância a falhas de sistemas distribuídos. É uma técnica considerada eficaz se for bem manipulada para resolver o problema no contexto de sistemas multiagentes. O ato de replicar agentes é o ato de criar uma ou mais duplicatas (ou réplicas) de um ou mais agentes. E o que vai definir o grau de replicação, isto é, quantos agentes devem ser replicados é, principalmente, o grau de criticalidade do mesmo e o quão complexo é adicionar ou remover membros no grupo de réplicas já existente.

A criticalidade de um agente significa o quão importante um agente é para o sistema, isto é, qual o potencial impacto que um agente defeituoso pode causar. Ela pode ser definida de acordo com o domínio do sistema e/ou de acordo com as interações e troca de mensagens efetuadas pelos agentes, e é considerada uma medida deste potencial impacto como um valor numérico dentro do intervalo

[0,1], onde 0 significa que o agente no momento não pode causar nenhum impacto, e 1 significa o máximo de impacto que o agente pode causar.

De fato, existem dois casos de estimativa de criticalidade que precisam ser diferenciados, quando: 1) a criticalidade do agente é estática, e 2) a criticalidade do agente é dinâmica. No primeiro caso, normalmente, o sistema multiagentes possui estruturas organizacionais fixas, comportamentos imutáveis dos agentes e um número reduzido de agentes interagindo. Desta forma, a criticalidade dos agentes pode ser identificada pelo projetista e passada para o programador antes da execução do sistema. No segundo caso, a criticalidade do agente não pode ser definida *a priori* caso o sistema multiagentes possua estruturas organizacionais dinâmicas, comportamentos dinâmicos dos agentes, e um número muito grande de agentes. Neste caso, é importante determinar estas estruturas dinamicamente a fim de avaliar e estimar a criticalidade dos agentes [25].

Definido o que é a técnica de replicação e criticalidade de agentes, faz-se necessário descrever os dois tipos principais de protocolos de replicação existentes: a replicação ativa e a replicação passiva. A replicação ativa ocorre quando todas as réplicas processam concorrentemente todas as mensagens recebidas. E a replicação passiva ocorre quando somente uma réplica processa concorrentemente e, de tempos em tempos, envia seu estado atual para as outras réplicas do grupo de réplicas de um determinado agente, a fim de manter a consistência.

Em [2], é proposto um *framework* adaptativo e dinâmico tolerante a falhas para SMA's, chamado DimaX. Este *framework*, que será apresentado na próxima seção, utiliza a técnica de Replicação de Agentes como uma forma de implementar tolerância a falhas em sistemas multiagentes. E o seu mecanismo é mantido de forma transparente para a aplicação.

2.2.1. Visão Geral: DimaX

O objetivo desta seção é apresentar uma visão geral do DimaX e de seus serviços para desenvolvimento de sistemas multiagentes tolerantes a falhas. DimaX [30] é o resultado da integração entre um *framework* para sistemas multiagentes, chamado DIMA [27], e uma *middleware* de replicação dinâmica,

chamada DarX [28][29]. Como resultado da integração, obteve-se uma plataforma multiagentes tolerante a falhas (Figura 2).

DimaX possui três camadas: (i) sistema, isto é, o *middleware* DarX; (ii) aplicação, isto é, os agentes; (iii) e o monitoramento. No nível da aplicação, Dima provê um conjunto de bibliotecas para construir sistemas multiagentes. No nível do sistema ou *middleware*, DarX provê os mecanismos necessários para distribuição, e replicação de agentes e serviços. Existe ainda um serviço de observação presente em ambas as camadas (sistema e aplicação). E, finalmente, o nível de monitoramento contem o controle da replicação de forma automática implementada através da cooperação e do serviço de observação [2].

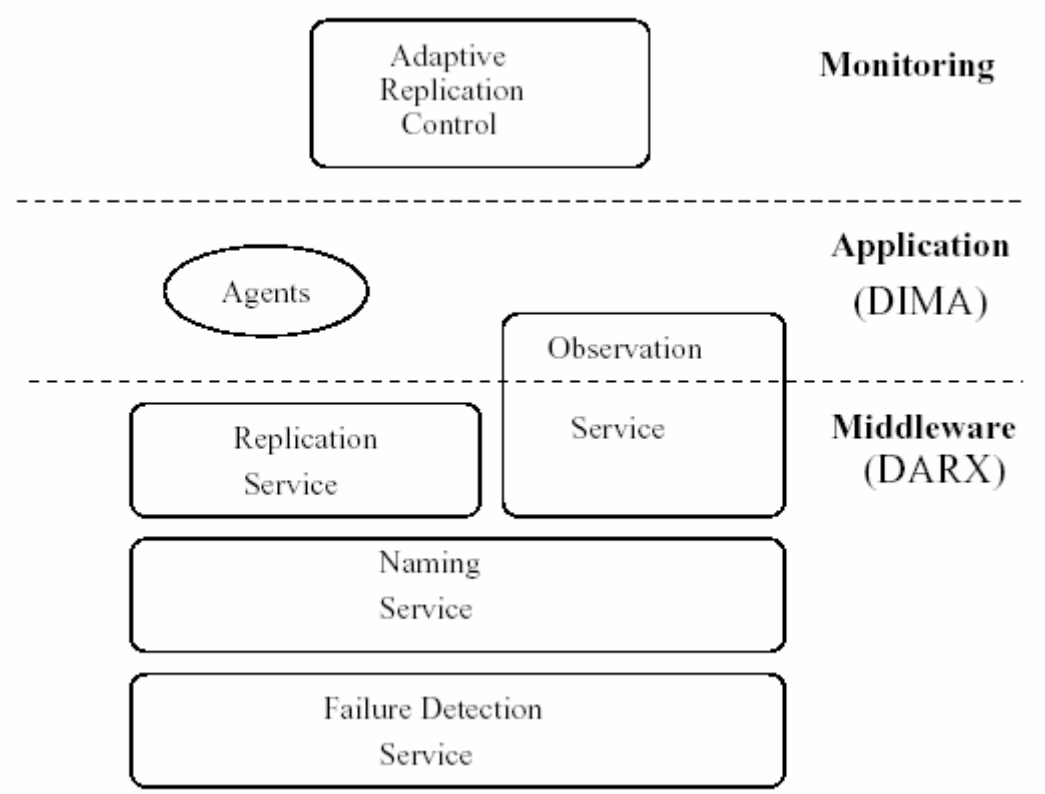

Figura 2 - Visão Geral do DimaX [30]

A fim de implementar o mecanismo de replicação de forma distribuída, DarX provê endereçamento de nomes globais (*global naming)*, e cada agente possui um nome global que é independente da localização corrente das réplicas dos agentes.

Quanto ao serviço de observação (Figura 3), existem dois níveis de coleta de dados: nível do sistema e nível da aplicação. No nível do sistema, DimaX coleta os dados referentes ao ambiente de execução do sistema multiagentes, tais como tempo de CPU e tempo médio de defeitos. No nível da aplicação, DimaX coleta

informações relativas às características dinâmicas como, por exemplo, eventos de interação entre os agentes. O serviço de observação possui um monitor para cada host do sistema multiagente distribuído que vai coletar e processar os dados de observação a fim de calcular informações locais, como o número de mensagens trocadas entre dois agentes, dado um período de tempo, por exemplo. Estes agentes monitores trocam também as informações entre si a fim de disponibilizar informações globais a respeito do sistema.

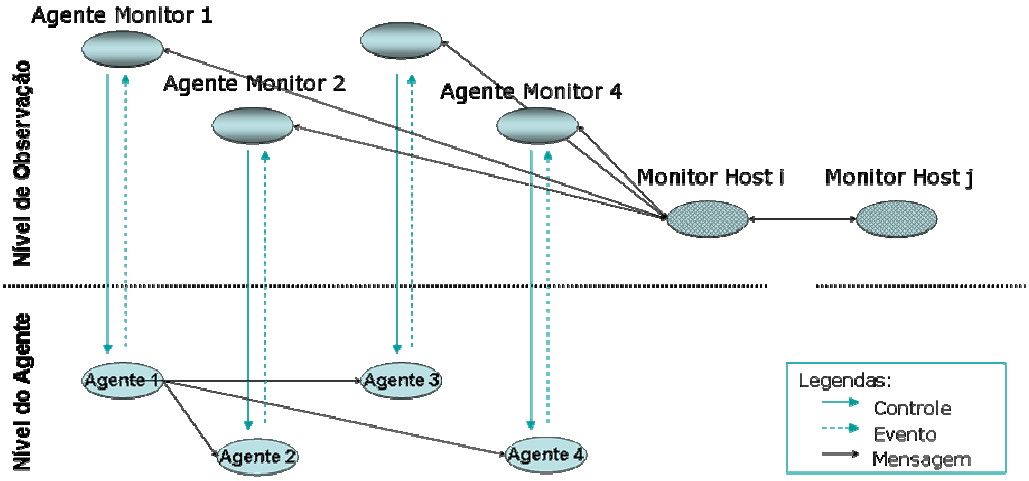

Figura 3 - DimaX: Arquitetura do Serviço de Observação

Apesar de o mecanismo adaptativo e dinâmico do DimaX implementar diversas estratégias de estimativa de criticalidade dos agentes, e isto irá definir o grau de replicação de tais agentes, DimaX não provê um mecanismo de monitorar a criticalidade dos agentes com um grau de granularidade maior. Não seria possível, por exemplo, especificar regras organizacionais que pudessem variar a criticalidade dos agentes através dos mecanismos de especificação de variação de criticalidade propostos na arquitetura até o presente momento. E seria muito custoso implementar tal mecanismo no DimaX, já que o mesmo não possui um mecanismo desacoplado de monitoramento de conformidade dos agentes com as regras de interação

Entretanto, DimaX fornece para o desenvolvimento e implementação de tolerância a falhas de sistemas multiagentes de larga escala características essenciais: escalabilidade, reusabilidade, robustez e adaptação para replicação. E, considerando que DimaX já foi avaliado e validado em diversos experimentos [2][4], este *framework* foi o considerado o mais adequado para o desenvolvimento do trabalho proposto nesta dissertação.

2.3. Governança de SMA's Abertos

Como no desenvolvimento de sistemas multiagentes abertos, os agentes são implementados de forma independente, faz-se necessário prover um meio para se atingir um sistema coerente. Além disso, considerando o desafio apresentado pelo comportamento emergente, leis de interação podem ser usadas para mitigar estes riscos que poderiam gerar diversos tipos de falhas [31]. Leis são regras de interação que especificam a integridade de um sistema aberto através de um conjunto de propriedades que devem ser obedecidas. Uma analogia com assertivas pode ser feita, de forma que é possível afirmar que leis procuram interceptar falhas dinamicamente.

De forma centralizada ou distribuída, o controle da integridade das leis ao executar o sistema é feito por mediadores que interceptam as mensagens ou informações de um ambiente e estes mediadores aplicam as leis ou penalidades previstas e especificadas. Alguns trabalhos foram propostos como forma de assegurar a governabilidade de sistemas através de leis de interação, dentre eles [10], [11], [12], [32], [33], [31], [38], [34], [35] e [36].

Este trabalho utiliza a abordagem proposta por [35] através da linguagem declarativa de especificação chamada XMLaw, associada ao *framework* (M-Law) que implementa o mecanismo de regulação das interações dos agentes fazendo verificação em tempo de execução. Por questões de foco e espaço, as vantagens que o XMLaw apresenta em relação às demais abordagens não serão discutidas, e podem ser encontradas em [35]. Entretanto, para haver um maior entendimento da solução proposta, a próxima seção irá descrever uma visão geral da linguagem XMLaw e do *framework* M-Law.

2.3.1. Visão Geral: XMLaw e M-law

Em SMA's, os aspectos internos dos agentes são inacessíveis, e a única informação disponível sobre os agentes é o comportamento observável por meio das mensagens que eles emitem ao se comunicarem com os outros agentes. A figura abaixo ilustra o modelo conceitual do XMLaw para regular suas interações, baseado em elementos que compõe uma lei.

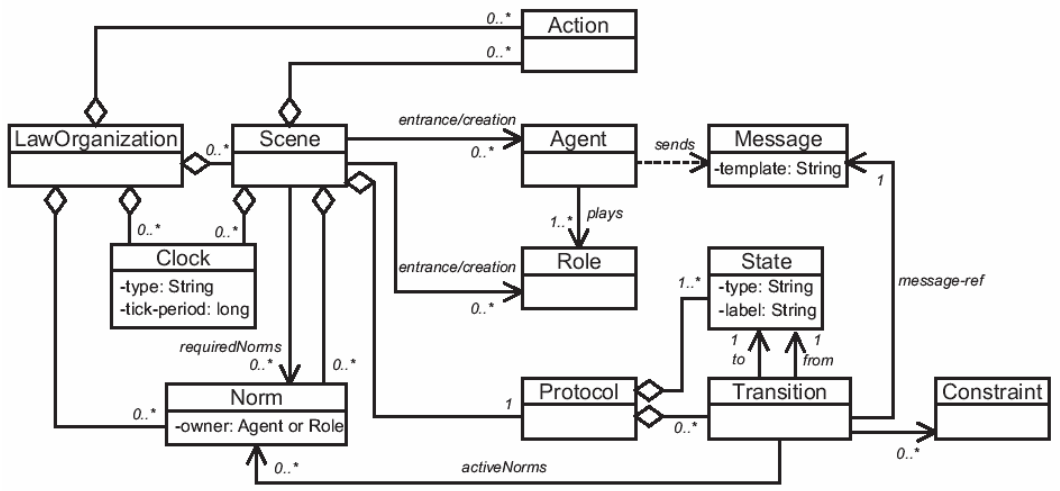

Figura 4 - XMLaw: Modelo Conceitual [35]

Dentre os elementos do modelo conceitual, o elemento *Message*, isoladamente, não permite especificar muitas restrições em uma interação. É preciso estabelecer a ordem em que elas ocorrem e quais são 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 [37], onde estados (elemento *State*) representam pontos na execução do protocolo e transições (elemento *Transition*) são as conexões entre os estados.

O modelo conceitual fornece um elemento denominado Relógio (*Clock*) 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.

Normas (*Norms*) descrevem quais comportamentos dos agentes são permitidos, quais são obrigados e quais são proibidos. 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.

O modelo conceitual utiliza a abstração de cenas para auxiliar na organização e modularização das interações, e 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.

Estes elementos compartilham um contexto comum de interação definido pela cena. Isto significa que uma norma definida no contexto de uma cena é somente visível naquela cena. Para que um agente inicie ou participe de um protocolo de interação de uma cena, é necessário que ele desempenhe algum papel (role) previamente definido na organização e especificado no contexto da cena.

Como mencionado anteriormente, além do modelo conceitual apresentado, existe um *framework*, chamado M-Law, que garante a regulação das interações dos agentes através dos elementos do modelo conceitual e do modelo de interação baseado em eventos. Este mecanismo intercepta as mensagens enviadas por agentes e verifica sua conformidade com as leis (Figura 5).

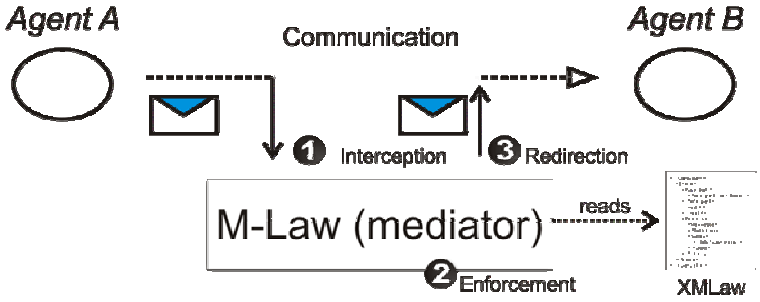

Figura 5 – Arquitetura M-Law [39]

Para que o *framework* M-Law pudesse prover os meios para efetivamente suportar a linguagem XMLaw e sua evolução, ele foi desenvolvido com quatro módulos (Figura 6). O módulo *Agent* contem as classes que os desenvolvedores de agentes devem utilizar para implementá-los. Este módulo por sua vez, utiliza o módulo de Comunicação para enviar e receber mensagens.

O Mediador, propriamente dito, por sua vez, engloba os outros dois módulos, Evento e Componente, e também utiliza o módulo de Comunicação. Os módulos de Evento e Componente não são visíveis para os desenvolvedores de agentes, mas podem ser estendidos para evoluir as funcionalidades do Mediador.

O modulo de Evento implementa a notificação e propagação de eventos. E o módulo de Componente define um conjunto de classes abstratas e concretas, além

de interfaces, que permitem que novas funcionalidades sejam inseridas. De uma forma geral, os componentes implementam o comportamento dos elementos da linguagem XMLaw. Os principais objetivos do M-Law são simplicidade, flexibilidade e reuso.

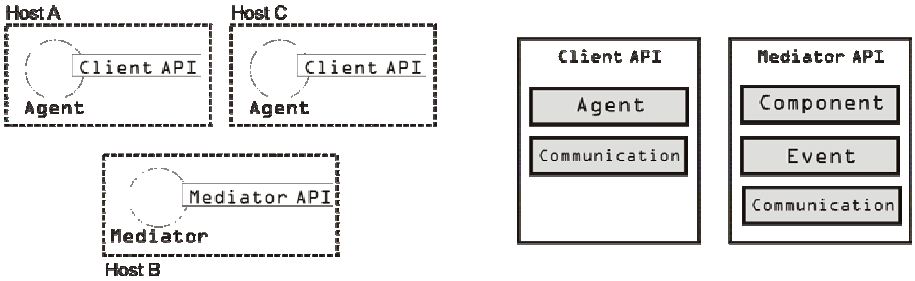

Figura 6 - Componentes do M-Law [39]

Na próxima subseção, o processo de especificação das leis tal como é feito atualmente de forma *ad hoc* será apresentado para que fique mais clara a descrição da aplicação da nossa proposta nessa abordagem apresentada no Capítulo 5.

2.3.1.1. Especificação das Leis

Como visto, para especificar as Leis em XMLaw é necessário descrever elementos de leis como os papéis, normas, relógios, cenas, protocolos e mensagens. Cada cena especifica o protocolo de interação que os agentes devem seguir e, conseqüentemente, quais são as mensagens válidas. A partir do entendimento dos riscos e requisitos associados a um SMA, leis podem ser especificadas.

A seguir descrevemos os passos usuais para a especificação de leis:

1. Crie a organização (*LawOrganization*) e estabeleça as normas (*Norms*), ações (*Actions*), restrições (*Constraints*) e relógios (*Clocks*) que podem existir nela.
2. Crie as possíveis cenas (*Scenes*) nas quais os agentes desempenhando os papéis especificados poderiam interagir dentro da organização.

- Defina o identificador de cada cena e, caso necessário, o seu tempo máximo de execução.

- Defina os protocolos com as suas mensagens, transições e estados possíveis de acordo com uma máquina de estados. Caso seja necessário reutilizar um protocolo existente em outra cena (ou até mesmo em outro arquivo), utilize a tag *xi:include* para especificar onde se encontra o protocolo a ser incluído.

- Defina as normas que regulam a interação e as normas que devem estar ativadas ou desativadas quando do início da execução da cena, caso existam para a cena em questão.

- Defina os relógios do contexto da cena, caso existam.

3. Associe as normas e os relógios para cada mensagem ou transição previamente especificadas no protocolo.

4. Refina as normas, relógios, ações e restrições da organização a partir das cenas especificadas.

Apesar de parecer trivial especificar uma lei em XMLaw, muitas vezes o processo de derivação de tais elementos (normas, relógios, ações e restrições) não é trivial, aumentando assim o esforço inicial. Além disso, nem sempre as leis geradas e especificadas satisfazem os requisitos do sistema, e não existe nenhuma proposta de como fazê-lo. Desta forma, assim como é difícil elicitar e projetar requisitos de leis, o mesmo se aplica para a especificação da criticalidade que será vista no capítulo 4. Por isso, este trabalho propõe no capítulo 5 uma abordagem para apoiar o processo de derivação de leis e, conseqüentemente, de análise de criticalidade.