4
Arquitetura Proposta

Neste capítulo, a arquitetura proposta será apresentada quanto às extensões no modelo conceitual da linguagem XMLaw, seguida das extensões implementadas no *framework* M-Law a fim de refletir as extensões do modelo conceitual. Depois, serão apresentadas as extensões no *framework* DimaX e, por último, a arquitetura final resultante da integração.

Contudo, uma descrição do problema mais detalhada será exibida antes da apresentação da arquitetura proposta, para que o leitor tenha uma maior compreensão do problema relacionado com os elementos do XMLaw e os conceitos de criticalidade utilizados pelo DimaX.

4.1.
Descrição do Problema

Para ilustrar o problema sobre como calcular dinamicamente a criticalidade de um agente considerando estruturas organizacionais dinâmicas, esta seção apresentará uma cena de negociação baseada no protocolo FIPA-CONTRACT-NET [50] (Figura 10). O objetivo de ilustrar este cenário é descobrir como responder duas questões principais: como e quais elementos (normas, relógios, etc) da linguagem XMLaw poderiam melhorar a análise de criticalidade dos agentes feita pelo DimaX? E, como esta análise poderia ser efetivada, considerando acoplamento, modularidade e reuso?

Neste protocolo, existem dois intervenientes: *Initiator* (o agente que tem o problema a resolver) e *Participant* (o agente que irá resolver o problema ou parte dele). Primeiramente, o *Initiator* solicita pedidos de propostas a outros agentes emitindo uma performativa [50] *call for proposal* (cfp), esta mensagem especifica a tarefa e as condições que o agente impõe para sua execução. As setas denotam uma comunicação assíncrona.

Os agentes que recebem o pedido de propostas, *Participants*, podem responder: (i) recusando fazer uma proposta, (ii) indicando que não percebeu a

mensagem, ou (iii) submetendo uma proposta. Sendo que a proposta do *Participant* inclui as pré-condições que este definiu para a tarefa, tais como o preço, altura em que a tarefa será executada, etc. Ao fim do deadline presente na cfp, as respostas devem ser recebidas pelo *Initiator*, de modo a que este não fique à espera indefinidamente caso o *Participant* falhe uma resposta. E o *Initiator* avalia as propostas e escolhe o(s) agente(s) que irá(ão) desempenhar a tarefa (pode não escolher qualquer agente). Os agentes cujas propostas foram aceitas recebem uma mensagem de confirmação e os outros de rejeição. Quando o *Participant* recebe a confirmação da proposta, informa o *Initiator* da execução da proposta, que pode cancelar a execução da proposta a qualquer instante.

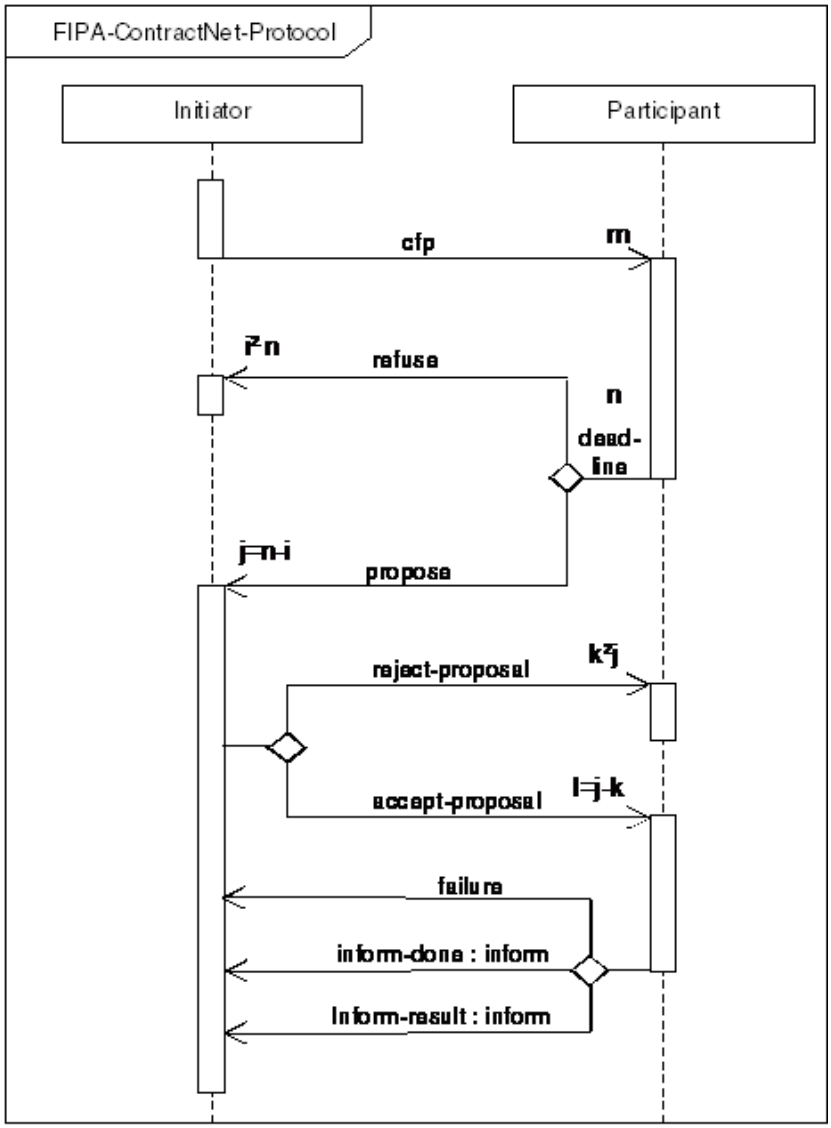

Figura 10 - Protocolo FIPA CONTRACT-NET

Suponha agora que este protocolo tenha sido descrito como uma máquina de estados (Figura 11), onde si é um estado dos protocolos durante a sua execução e

cada sinal de mais (+) ou menos (-) representa a ativação ou desativação de um relógio, respectivamente. O protocolo começa com o estado s0, quando o *Initiator* solicita m propostas de outros agentes e termina com os estados s4, ou s5, ou s7, dependendo do fluxo de execução realizado no protocolo.

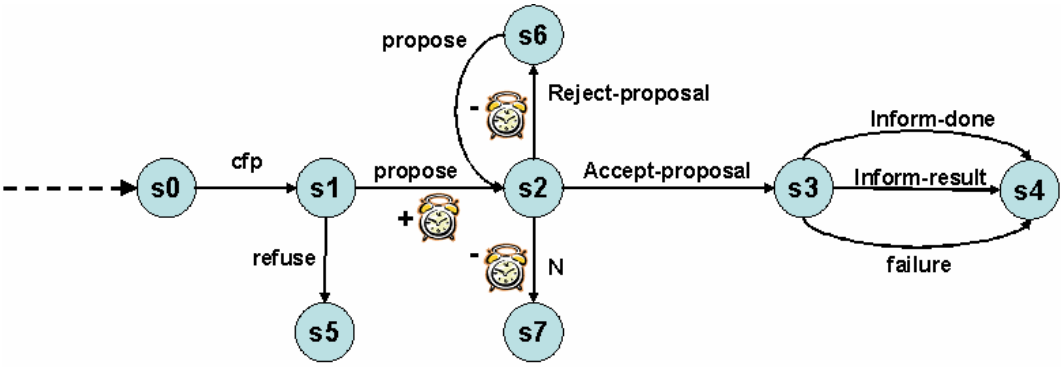

Figura 11 - Máquina de Estados para o Protocolo CONTRACT-NET

Considere esta representação para um outro cenário de negociação que será utilizado para analisar a variação da criticalidade dos agentes. Um agente comprador inicia uma negociação enviando uma proposta de compra de um livro para uma lista de vendedores. Ele informa o preço máximo para o livro e o vendedor pode aceitar com uma proposta ou pode recusar. Se o vendedor aceitar, ele pode enviar propostas com preços menores ou iguais ao preço informado pelo comprador.

Quando o comprador receber a proposta do vendedor, ele terá 20 segundos para decidir se aceitará ou não. Após 20 segundos, se o comprador não tiver respondido, o vendedor poderá oferecer outras propostas do mesmo livro para outros compradores. Senão, o vendedor assume o compromisso de vender aquele livro somente a este comprador. Se o comprador recusar sua proposta, o vendedor pode propor outro preço. Se o comprador aceitar, o vendedor informa o banco onde o pagamento deverá ser efetuado e o comprador assume o compromisso de pagar e enviar o comprovante de pagamento para que possa receber o livro. Quando isso ocorre, a cena acaba.

Na Figura 12, a máquina de estados para o protocolo desta negociação é ilustrada, assim como todos os elementos XMLaw (relógios e normas) que são ativados durante a sua execução.

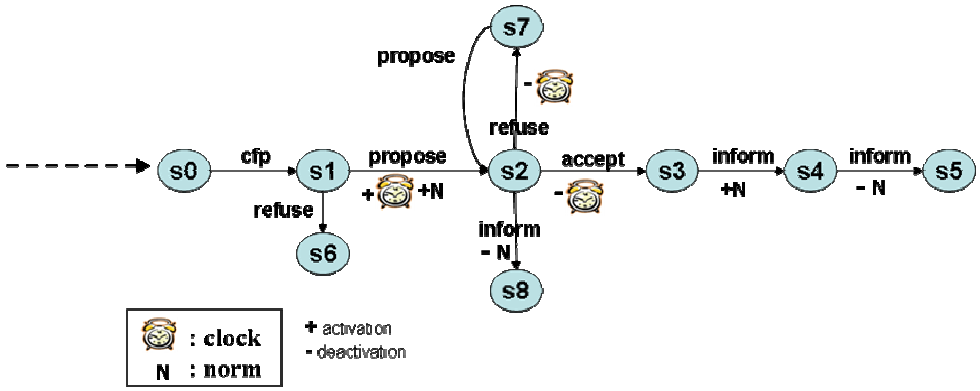

Figura 12 - Máquina de Estados do Protocolo da Negociação

Esta máquina de estados pode ser interpretada segundo a tabela abaixo:

Estado		Mensagem	Descrição	Evento
Inicial	Final			
s0	s1	cfp	Agente comprador inicia uma negociação enviando uma proposta de compra de um livro para uma lista de agentes vendedores.	
s1	s2	propose	Agente vendedor aceita a proposta do agente comprador e envia uma proposta com preço menor ou igual ao preço informado pelo comprador.	-Ativação de relógio -Ativação de norma
s1	s6	refuse	Agente vendedor recusa a proposta do agente comprador e o protocolo termina.	
s2	s3	accept	Agente comprador aceita a proposta do agente vendedor em menos de 20 segundos.	Desativação de relógio
s2	s7	refuse	Agente comprador recusa a proposta do agente vendedor em menos de 20 segundos.	Desativação de relógio
s7	s2	propose	Agente vendedor enviar outra proposta para o agente comprador.	
s2	s8	inform	Agente vendedor informa para o agente comprador que ele está permitido vender o livro para outra pessoa, e o protocolo termina.	Desativação de norma
s3	s4	inform	Agente vendedor envia as informações necessárias para o agente comprador efetuar o pagamento.	Ativação de norma
s4	s5	inform	Agente comprador envia o comprovante de pagamento.	Desativação de norma

Tabela 2 - Descrição da máquina de estados do protocolo

Considerando que, quando um evento (ativação/desativação de relógio ou norma, etc) ocorre durante a execução da cena, a criticalidade do agente pode aumentar ou diminuir, já que ele pode se tornar menos ou mais importante de

acordo com os eventos disparados. Desta forma, cada elemento deve ser levado em consideração para o cálculo da criticalidade do agente.

Além dos elementos mencionados, relógios e normas, outros elementos e eventos que não são manipulados pelo XMLaw deveriam ser analisados a fim de avaliar suas influências na análise de criticalidade do agente. Então, por exemplo, se um agente começa a desempenhar um papel, sua criticalidade aumenta ou diminui.

No contexto da cena de negociação, quando o agente comprador é obrigado a responder se ele aceitará a proposta do agente vendedor ou não, como o evento de ativação de relógio será ativado, a criticalidade do comprador deveria aumentar, já que ele se tornou importante para o vendedor e para o sistema no geral. E, quando o evento de ativação de relógio fosse desativado, a criticalidade do comprador deveria diminuir.

Uma outra situação neste cenário seria relativa ao pagamento do produto. Já que o comprador possui a obrigação de pagar pelo produto após aceitar a proposta do vendedor, sua criticalidade também deveria aumentar, já que o vendedor e o sistema estão esperando este pagamento para efetuar uma ação, que no caso, seria enviar o produto para o comprador.

Todas essas variações são exibidas na Figura 13. Do lado esquerdo da figura, pode-se ver a execução do protocolo. E, ao seu lado, é exibido um rascunho da principal variação de criticalidade do agente comprador. O resultado principal é baseado na variação da criticalidade associado a cada evento previamente mencionado.

A figura do relógio representa um evento de ativação ou desativação de relógio, e a letra N representa um evento de ativação ou desativação de norma durante a execução do protocolo, de acordo com o sinal de mais ou menos que segue cada figura ou letra.

De forma análoga poderíamos analisar a criticalidade do agente vendedor. Então, por exemplo, sua criticalidade poderia aumentar quando o comprador propusesse um preço para o produto, já que ele tem a obrigação de respondê-lo.

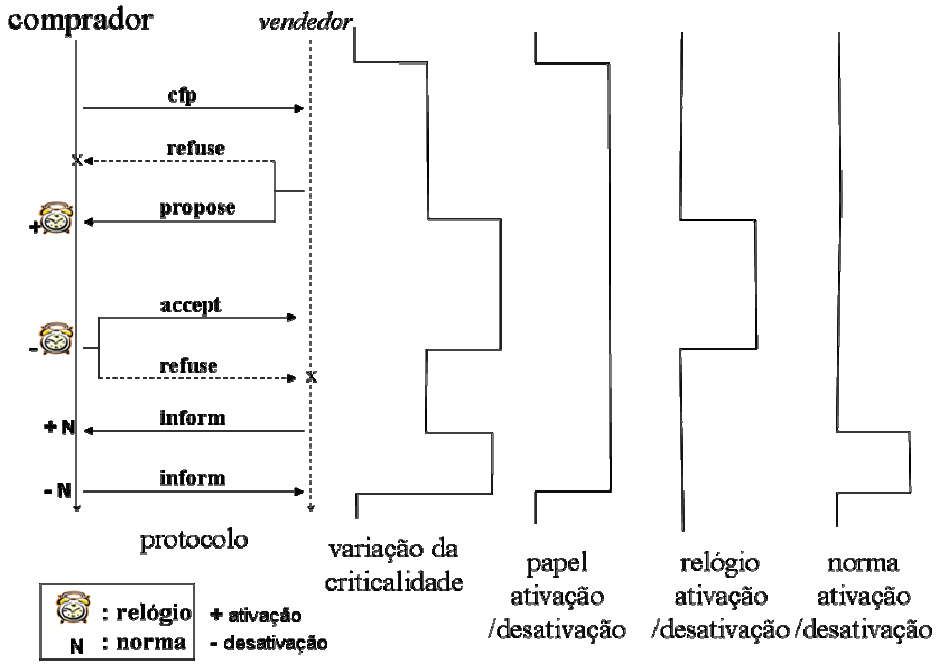

Figura 13 - Variação da Criticalidade

Portanto, entendida a variação da criticalidade de um agente durante a execução de um protocolo, todos os eventos que podem ser disparados durante a execução do mesmo poderia aumentar ou diminuir a criticalidade de acordo com o tipo do evento e sua semântica.

Na próxima seção, a arquitetura proposta será apresentada e, nas seções subseqüentes, serão descritos quais elementos e eventos influenciam esta variação e quais foram os mecanismos implementados para executar tal variação em sistemas multiagentes abertos governados por leis.

4.2. Arquitetura Proposta

Com o objetivo de ilustrar a arquitetura resultante da integração dos *frameworks* M-Law e DimaX, a Figura 14 foi gerada. Dado dois agentes: *Agente A* e *Agente B*. Cada um deles possuirá um agente monitor chamado, respectivamente: *Monitor Agente A, Monitor Agente B*. Suponha que ambos os agentes estão na mesma máquina (ou *host*) e que cada agente monitor se registrará indiretamente em uma porta para comunicações via socket.

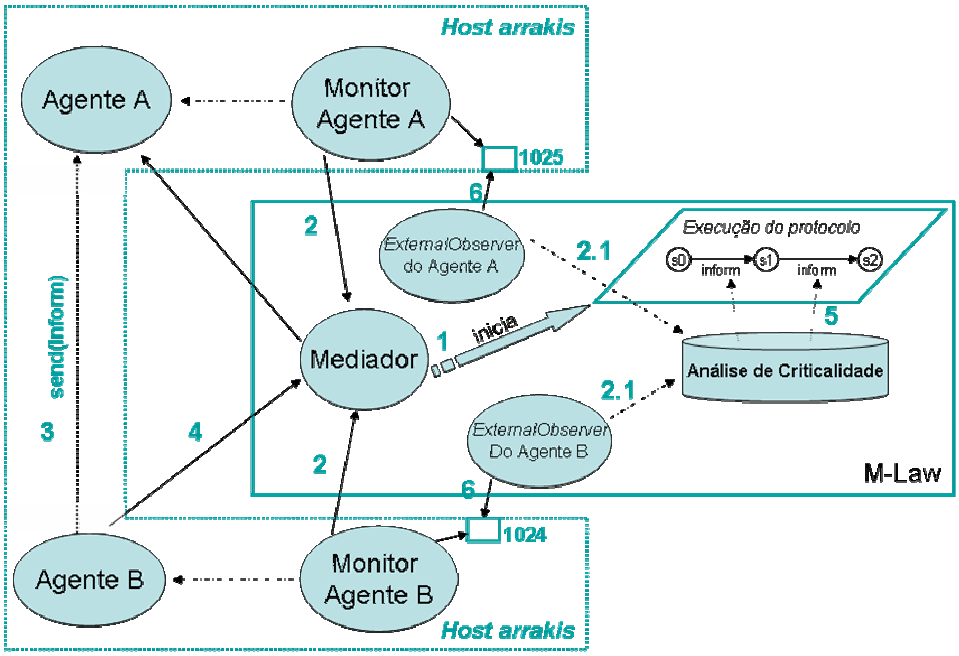

Figura 14 - Arquitetura de Integração

O seguinte fluxo de execução ocorrerá no momento em que os agentes são criados e que, por exemplo, uma cena de interação entre os mesmo é iniciada:

1. O Mediador é iniciado e a lei é carregada;
2. Os agentes monitores (DimaX) requisitam observadores externos ao mediador;
3. O agente *Agente B* envia uma mensagem para o agente *Agente A*;
4. O Mediador intercepta a mensagem e executa o protocolo de aplicação das leis;
5. O módulo de análise da criticalidade monitora os eventos que ocorrem à partir do envio da mensagem e recalcula a criticalidade dos agentes envolvidos; Este módulo dispara um evento a ser percebido pelo observador externo (*ExternalObserver*) de cada agente. Este observador externo pertence ao M-Law e será descrito posteriormente.
6. O observador externo percebe o evento e envia uma mensagem via socket para o monitor do agente.

As próximas subseções irão apresentar cada um dos elementos aqui citados dentro do contexto das extensões realizadas em cada *framework* e na linguagem XMLaw.

4.2.1.
Extensões na Linguagem de Especificação XMLaw

Como visto no capítulo 2, o modelo conceitual de XMLaw é composto por elementos conceituais que permitem a representação dos aspectos de interação de agentes. Dentre estes conceitos existem as cenas, que agrupam as interações em módulos, os protocolos, que definem precisamente a comunicação entre os agentes, as normas, as quais podem ser obrigações, permissões ou proibições, os relógios que adicionam aspectos temporais nas interações permitindo que um determinado comportamento seja válido durante um período de tempo, e ações e restrições, as quais implementam determinados comportamentos em código Java.

Para que a linguagem XMLaw apoiasse mecanismos de monitoramento de criticalidade, foi necessário estendê-la acrescentando mais expressividade e funcionalidade para o elemento Role, que define os papéis das organizações, e criando um novo elemento que define o monitoramento propriamente dito, chamada *CriticalityAnalysis* (veja Figura 15). No antigo modelo conceitual, conceitualmente o elemento Role existia, mas na prática, era somente um atributo referenciado em outros elementos. De fato, os papéis desempenhados pelos agentes tinham que ser informados por eles mesmos quando pediam para entrar em uma determinada organização. E este papel era então associado ao novo nome do agente na organização. Ficava então difícil de manipular eventos de empenho ou desempenho de papel de acordo com as cenas. E além disso o forte acoplamento ao nome do agente, impossibilitava o desempenho de mais de um papel ao mesmo tempo.

Com o elemento Role deste novo modelo, sempre que um agente entra em uma organização ou uma cena, ele precisa informar o papel que deseja desempenhar. Sendo que um agente pode desempenhar um ou mais papéis ao mesmo tempo, de acordo com o número de organizações e/ou cenas as quais pertence. Uma organização também pode ter mais de um papel especificado.

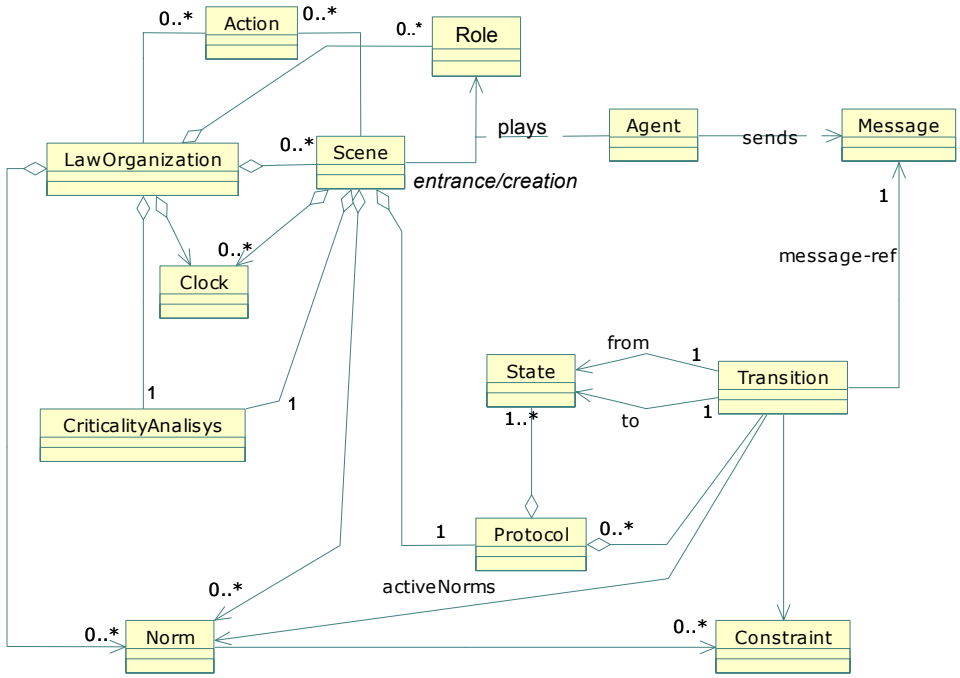

Figura 15 – XMLaw: Novo Modelo Conceitual

Quanto ao novo elemento de criticalidade (*CriticalityAnalysis*), para garantir modularidade, este elemento pode ser adicionado tanto no nível da organização quanto no nível da cena. Desta forma, para cada cena existiria um módulo de criticalidade opcional a ser especificado. Então, por exemplo, se um agente está em duas cenas em execução diferentes e, se em cada cena existe um módulo de criticalidade especificado que contribua para a variação de criticalidade do agente naquele instante, a criticalidade resultante será composta do resultado da criticalidade do agente em cada cena (Figura 16).

No caso do módulo de criticalidade ser especificado no contexto da organização, ele só poderá conter referências para eventos que ocorram no nível da organização. Então, por exemplo, os eventos lançados por elementos tais como normas, relógios, ações e restrições que foram especificados no nível da organização, poderão ser percebidos pelo módulo de criticalidade. Porém os eventos lançados no nível das cenas não poderão ser percebidos, e, portanto, não poderão ser especificados no módulo de criticalidade pertencente à organização.

O mesmo, entretanto, não ocorre no sentido contrário. Quando um elemento definido no contexto da organização lançar um evento no contexto de uma cena, o seu módulo de criticalidade poderá percebê-lo e alterar a criticalidade do agente.

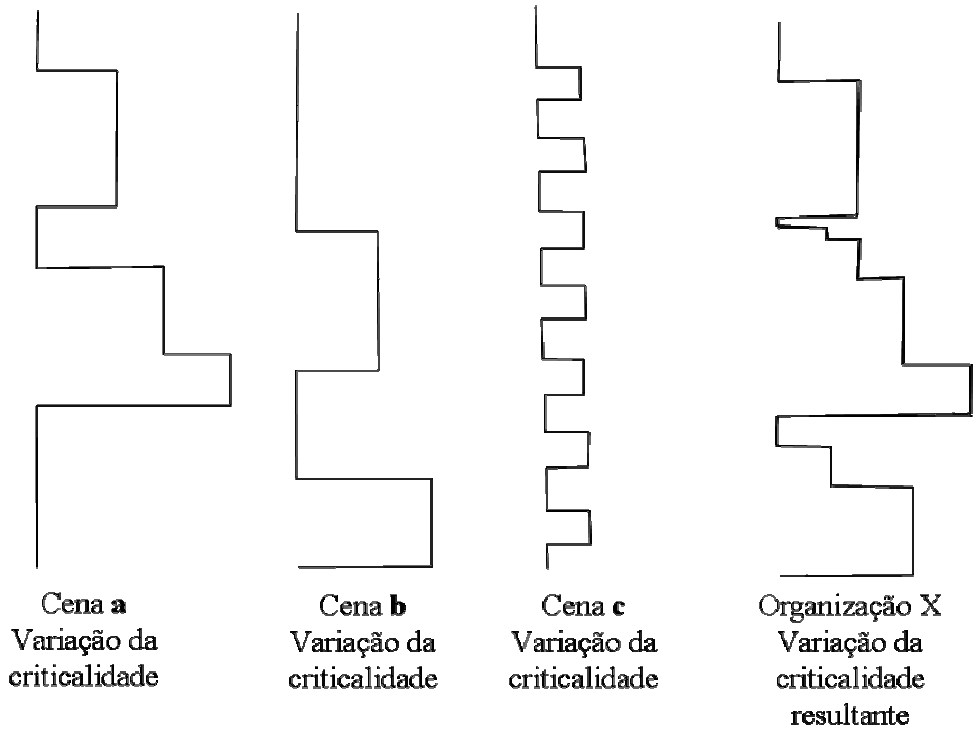

Figura 16 - Variação de Criticalidade de um agente entre as Cenas

Uma vez definido o escopo do módulo de criticalidade em cada nível, é necessário entender como se especifica tal módulo para que mecanismo que implementa seu comportamento funcione corretamente.

Cada módulo de criticalidade contém três elementos principais: *Weigth*, *Increases* e *Decreases* (Figura 17). O primeiro elemento define o peso para cada tipo de evento que o será especificado nos outros elementos. Caso ele não seja especificado, um valor padrão será atribuído de forma que o mecanismo funcione corretamente. Sendo assim, eventos do tipo mensagem (*message_arrival*) recebem valor 0.05. Eventos do tipo transição (*transition_activation*) recebem valor 0.05. Eventos do tipo relógio (*clock_activation, clock_deactivation*) recebem valor 0.4. Eventos do tipo papel (*role_activation, role_deactivation*) recebem valor 0.1. E, finalmente, eventos do tipo norma (*norm_activation, norm_deactivation*) recebem valor 0.4. Caso o projetista do monitoramento da criticalidade não deseje, por exemplo, que eventos do tipo mensagem sejam monitorados, basta especificar neste elemento o valor para 0.

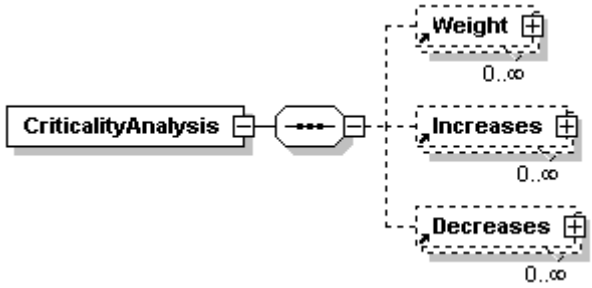

Figura 17 - Elemento: *CriticalityAnalysis*

Os outros dois elementos especificam as informações necessárias para o módulo de monitoramento perceber um determinado evento e tratá-lo a fim de recalcular a criticalidade de um determinado agente. O elemento *Increases* contém a lista de todas as informações para monitoramento que resulte em um aumento de criticalidade. E o elemento *Decreases* contém a lista de todas as informações para monitoramento que resulte em uma diminuição de criticalidade. O módulo de criticalidade pode conter zero ou mais elementos que definam um aumento ou diminuição.

Tanto o elemento *Increase*, quanto o elemento *Decrease* são especificados através de três atributos e um elemento *Assignee* (Figura 18). O atributo *event-id* especifica o identificador do evento que será percebido, o atributo *event-type* especifica o tipo do evento definido pelo atributo *event-id*, e o atributo *value* representa o valor que aquele evento contribui para o aumento ou diminuição da criticalidade do agente. E, finalmente, o elemento *Assignee* contém informações sobre o agente que terá a sua criticalidade alterada pelo evento especificado. Estas informações são: o papel do agente e a variável que define a instância do mesmo naquela organização.

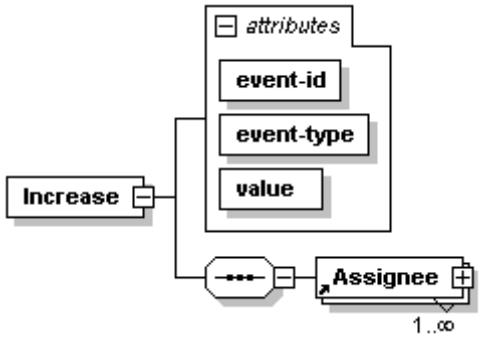

Figura 18 - Elemento: *Increase*

Considerando o exemplo da descrição do problema, a Tabela 3 mostra a especificação resultante para o monitoramento de criticalidade da cena

especificada como um exemplo de especificação utilizando os elementos descritos. Note que, pelo exemplo, os eventos de chegada de mensagem não serão monitorados. Já os do tipo papel, serão monitorados com um valor diferente do padrão, isto é, 0.2. E todos os outros serão com o valor padrão.

```
<CriticalityAnalysis>
    <Weight ref="role" value="0.2"/>
    <Weight ref="message" value="0"/>
    <Increases>
      <Increase event-id="customer" event-type="role_activation" value="0.3">
            <Assignee role-ref="customer" role-instance="$customer.instance"/>
      </Increase>
      <Increase event-id="seller" event-type="role_activation" value="0.7">
            <Assignee role-ref="seller" role-instance="$seller.instance"/>
      </Increase>
      <Increase event-id="time-to-decide" event-type="clock_activation" value="0.5">
            <Assignee role-ref="customer" role-instance="$customer.instance"/>
      </Increase>
      <Increase event-id="customer-payment-voucher" event-type="norm_activation"
      value="0.8">
            <Assignee role-ref="customer" role-instance="$customer.instance"/>
      </Increase>
    </Increases>
    <Decreases>
      <Decrease event-id="customer" event-type="role_deactivation" value="0.3">
            <Assignee role-ref="customer" role-instance="$customer.instance"/>
       </Decrease>
      <Decrease event-id="seller" event-type="role_deactivation" value="0.7">
             <Assignee role-ref="seller" role-instance="$seller.instance"/>
      </Decrease>
      <Decrease event-id="seller-permission-to-cancel" event-type="norm_activation"
      value="0.5">
             <Assignee role-ref="seller" role-instance="$seller.instance"/>
      </Decrease>
    </Decreases>
</CriticalityAnalysis>
```

Tabela 3 - Exemplo de Especificação de Criticalidade

Podemos ver neste exemplo também que, quando um agente começa a desempenhar o papel de comprador (*customer*), sua criticalidade deve ser recalculada utilizando o valor 0.3 e atualizada. O mesmo ocorre quando um agente começa a desempenhar o papel de vendedor (*seller*), sua criticalidade deve ser recalculada utilizando o valor 0.7 e atualizada. Estas ações são executadas quando o evento de ativação de papel (*role_activation*) for disparado.

4.2.2. Extensões no Framework M-Law

Nesta seção são apresentadas as mudanças necessárias para implementar o comportamento do módulo de monitoramento de criticalidade incluído na

linguagem XMLaw, e como o mecanismo funciona em tempo de execução no *framework* M-Law. Além disso, serão apresentadas também as alterações implementadas para o elemento de papel (Role) no modelo conceitual.

4.2.2.1. Papéis

Os papéis são representados pela classe *RoleDescriptor*. Esta classe por si só, não apresenta muita funcionalidade. O que este conceito trouxe de importante para o XMLaw e M-Law foi o conceito de desempenhar um papel em organização e/ou cena.

Como mencionado anteriormente, uma organização pode ter um ou mais agentes logados (agentes que pediram para desempenhar um determinado papel na organização) em um determinado instante. E o mesmo se dá para a cena. A Figura 19 ilustra este cenário. A organização é representada pela classe *OrganizationExecution* e a cena, *SceneExecution*.

É importante ressaltar que, quando um agente pede para entrar na organização com um determinado papel, a única verificação implementada é quanto ao fato de o agente já estar desempenhando um papel na mesma. Entretanto, quando um agente pede para entrar em uma cena desempenhando um determinado papel, uma série de verificações é feita a fim de validar a entrada. Primeiro o sistema verifica se a cena permite a entrada de um agente com o papel especificado. Segundo, verifica-se se o agente já não entrou na cena antes com este papel. Terceiro, verifica-se se já existe o número máximo de agentes desempenhado o papel solicitado no momento do pedido. Em caso afirmativo, nega-se a entrada. Por último, verifica-se se o estado corrente da cena permite a entrada do agente. Estas verificações foram implementadas no método *canAgentEnter*. Estando todas satisfeitas, o método *enterScene* é executado e o agente entra na cena. Vale ressaltar que algumas dessas verificações já estavam implementadas utilizando a forma *ad hoc* de especificação de papel, e só foram descritas aqui para um melhor entendimento do mecanismo.

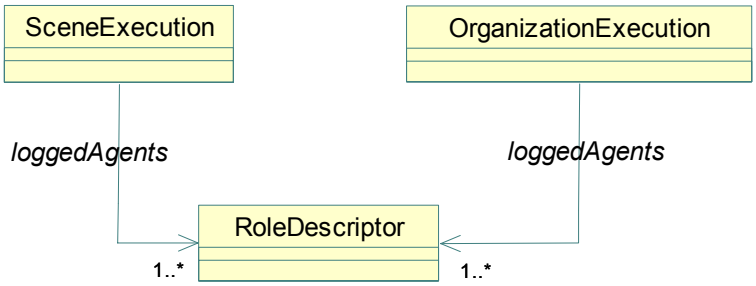

Figura 19 - Módulo de Papel

4.2.2.2. Análise de Criticalidade

No XMLaw, elementos de leis estão inter-relacionados de modo que seja possível especificar protocolos de interação usando restrições de tempo, normas ou mesmo normas sensitivas ao tempo. E a composição e o inter-relacionamento entre os elementos de leis são alcançados através dos eventos. Um elemento de lei pode gerar eventos para avisar algo para outros elementos. Do mesmo modo, outros elementos podem sentir eventos para muitos propósitos, como por exemplo, para ativar ou desativar a si mesmos.

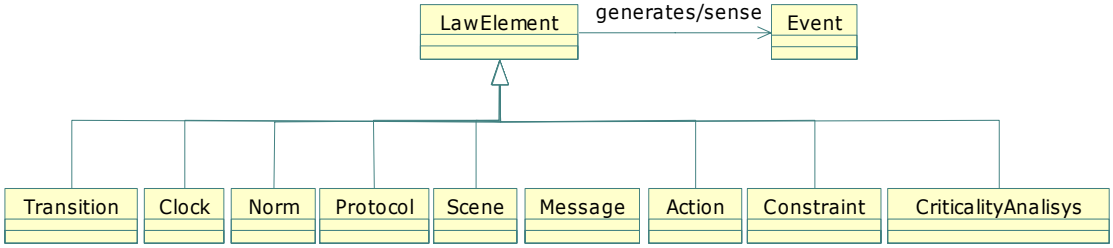

Figura 20 – Eventos

Para que o novo elemento *CriticalityAnalysis* (descrito na seção anterior) fosse capaz de implementar o monitoramento da criticalidade dos agentes, foi necessário implementá-lo como os outros principais elementos XMLaw. Desta forma, ele é capaz de gerar e perceber os eventos do sistema e, a partir de então, não sobrecarregar o tempo de execução do mesmo (Figura 20).

Assim, existe uma classe *CriticalityAnalysisExecution* que implementa a interface *IObserver* (Figura 21). A interface *IObserver* define o comportamento de consumidores de eventos. Consumidores registram seus interesses em certos tipos de eventos utilizando o método *attachObserver*. Desta forma, quando a classe

CriticalityAnalysisExecution é instanciada, ela se subscreve para cada evento especificado na lista de *Increases* e *Decreases* do elemento *CriticalityAnalysis*. Além de se subscrever nos eventos do tipo *message_arrival*.

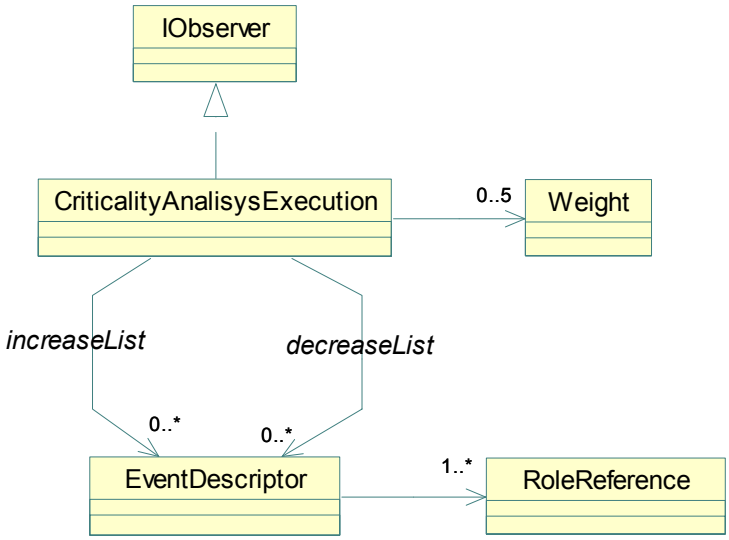

Figura 21 - Módulo de Criticalidade

Quando a instância do objeto *CriticalityAnalysisExecution* está em execução e recebe um evento para o qual se cadastrou, ela verifica se os dados do evento coincidem com os dados previamente especificados quanto ao agente que terá sua criticalidade calculada. Para isto, compara-se a referência para o *RoleReference* passado no evento com o *RoleReference* esperado definido pelo elemento *Assignee* do XMLaw.

Uma vez que um evento de alteração de criticalidade ocorreu, existe um método chamado *updateAgentCriticality* que recebe como parâmetro o identificador do agente, o peso do evento, o valor do evento, a operação (adição ou subtração) e informações adicionais sobre o evento para escrita de logs.

Como ilustrado no diagrama de seqüência abaixo (Figura 22), este método não atualiza a criticalidade propriamente dita do agente, pois quem o fará será o DimaX. Ele somente recebe os parâmetros, calcula a criticalidade local do agente, cria e lança um evento com tais informações e com o tipo de evento igual a *update_criticality*. Este evento, por sua vez, será percebido por uma entidade responsável por informar ao DimaX a alteração da criticalidade dos agentes, chamado observador externo e cuja classe chama-se *ExternalObserver* (Figura 23).

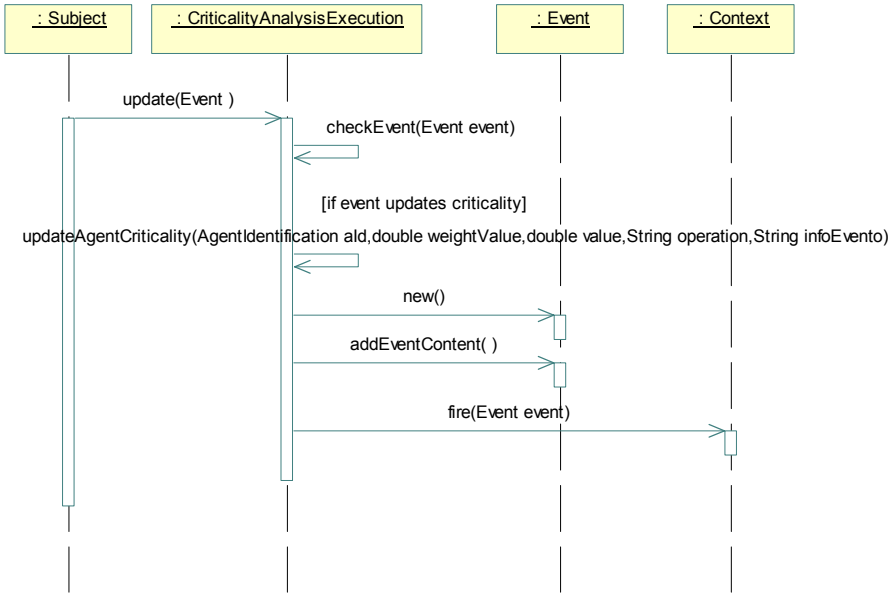

Figura 22 – Diag. de Seqüência para a execução da variação da criticalidade

A classe *ExternalObserver* implementa a interface IObserver (Figura 24) assim como a classe *CriticalityAnalysisExecution*, e possui com canal de comunicação via socket (*SocketCommunication*), para poder enviar mensagens para o DimaX, já que o M-Law utiliza como canal de comunicação a plataforma Jade [51] e o DimaX não.

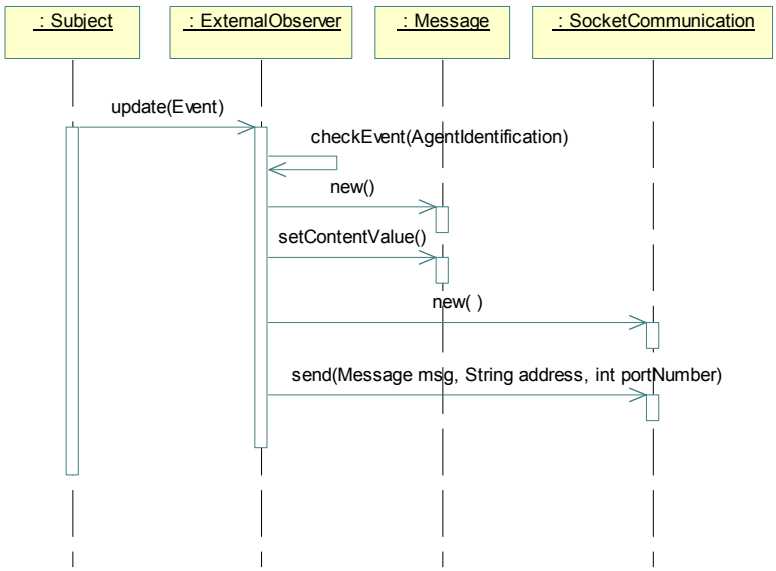

Figura 23 – Diag. de seqüência da execução do observador externo

Vale ressaltar que este observador externo existe para cada agente que está interagindo em alguma cena. Assim que ele é inicializado (que é quando um

agente monitor do agente é criado), ele se cadastra para eventos do tipo *update_criticality* e, quando recebe eventos deste tipo, verifica se é destinado para o agente ao qual possui referência. Ele foi implementado desta forma, porque, como visto no capítulo 2, existe um agente monitor para cada agente no DimaX. Logo, para que o observador externo pudesse enviar as informações relativas a alteração de criticalidade de um determinado agente através do canal socket, seria necessário ter o endereço do monitor do agente em questão. Portanto, foi necessário mapear um agente monitor no DimaX para cada observador externo no M-Law.

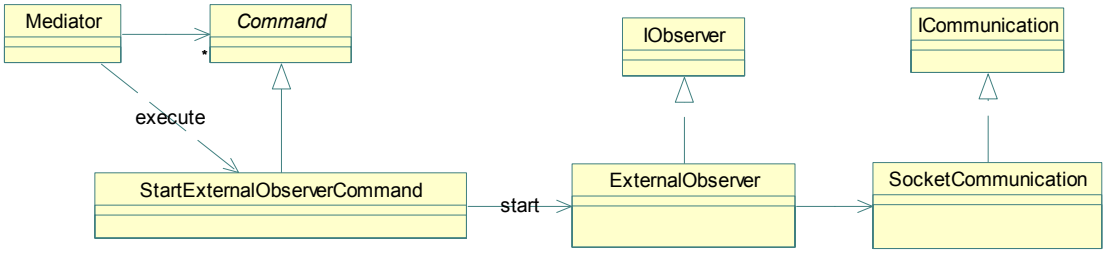

Figura 24 - Observador externo

4.2.3. Extensões no Framework DimaX

Este trabalho apresenta o mesmo raciocínio realizado em [4] e [26] para a atualização da criticalidade dos agentes. Existe uma classe chamada *XMLawBasedMonitor* que estende a classe do DimaX, chamada *AgentMonitor* (Figura 25). Além disso, para que o monitor do agente não tivesse o seu tempo de execução comprometido esperando mensagens do observador externo do M-Law, existe uma classe chamada *MonitorTas*k que estende de *Thread* (Java). Esta classe possui uma referência para o monitor e, enquanto o mesmo existir, ela estará ativa. Sempre que ela recebe uma mensagem via socket com um alerta de alteração de criticalidade, a mesma pede para o monitor executar o método *computeCriticalityFromXmlaw* e passa o valor e a operação a ser realizada.

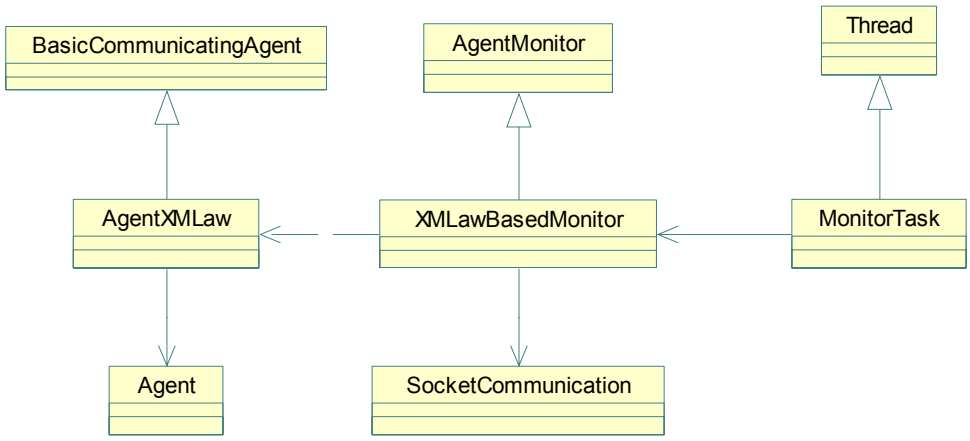

Figura 25 - Extensões no DimaX

O valor recebido foi calculado pelo XMLaw *a priori* e nada mais é que a multiplicação do peso do tipo do evento (a_e) com o valor especificado para o evento (v_e). Logo, seja w_i (t) a criticalidade do agente i em um determinado instante de tempo t, o valor da criticalidade irá variar segundo o seguinte critério:

- Se tiver sido gerado por um elemento Increase:

$$w_i(t) = w_i(t) + a_e * v_e$$

- Se tiver sido gerado por um elemento Decrease:

$$w_i(t) = w_i(t) - a_e * v_e$$

Desta forma, o cálculo do número de réplicas continuará sendo o mesmo utilizado pelo DimaX. Isto é, seja w_i a criticalidade do agente, W a soma de todas as criticalidades dos agentes, rm o número mínimo de réplicas que devem ser criadas independente do valor da criticalidade, e Rm um valor estimado para os recursos disponíveis que define todas as possíveis réplicas simultâneas. O número de réplicas nb_i do agente *i* é calculado como a seguir:

$$\boldsymbol{nbi(t)} = \text{rounded}(rm + wi(t)\ \ Rm/W)$$

4.2.4. Instanciando a arquitetura

Para implementar um agente que se beneficie do monitoramento da criticalidade através dos elementos de leis, basta estender da classe *AgentXMLaw* existente no módulo de monitoramento do DimaX. Esta classe estende de *BasicCommunicatingAgent* e possui toda a infra-estrutura e suporte de desenvolvimento de sistemas multiagentes provido pelo DimaX. Além disso, essa classe possui uma referência para a classe *Agent* do M-Law. Esta classe, por sua vez, implementa de forma transparente todo o mecanismo de regulação de interação de agentes através de leis. Logo, o desenvolvedor não precisa se preocupar com isto. Basta implementar os agentes em conformidade com as leis.

4.3. Discussões

Como visto neste capítulo, é necessário que o desenvolvedor das leis especifique a análise de criticalidade a ser implementada pelo mecanismo de monitoramento para que ele seja ativado. Entretanto, vale ressaltar que agentes com criticalidade elevada podem não ter a sua criticalidade diminuída caso a especificação não tenha sido feita da forma adequada.

Como exemplo, se tiver sido especificado que a criticalidade de um agente em uma determinada cena deve aumentar e não tiver sido especificado em momento algum quando a criticalidade desse mesmo agente deve diminuir, então haverá um elevado número de réplicas consumindo recursos do sistema que não mais deveriam existir. Portanto, de forma geral, para cada evento que possa ocorrer durante a execução de um protocolo que aumente a criticalidade de um agente, deveria existir um outro evento posterior que diminua a criticalidade do mesmo.

Logo, é de extrema importância que a fase de análise e projeto da especificação da criticalidade seja baseada em um levantamento de requisitos bem detalhado e consistente para garantir que problemas como este, de consumo de recursos inadequado, não ocorram.