4
O *Framework* para o Sub-Sistema de Julgamento

O objetivo desta seção é apresentar o *framework* que fornecerá o suporte necessário ao processo de julgamento apresentado ao longo do capítulo 3. Este *framework* representa o *Judgment System* apresentado na figura 2.

Um *framework* é uma aplicação semi-completa e reutilizável que deve ser estendida e especializada para construir aplicações completas e específicas [21]. As aplicações construídas a partir de um *framework* devem pertencer ao mesmo domínio (família de problemas) ao qual o *framework* se destina. O *framework* deve capturar os conceitos gerais de um domínio, possuir partes que são passíveis de extensão (pontos de flexibilização) chamados *Hot Spots* e partes que não são passíveis de extensão, chamados *Frozen Spots*. Os *hot spots* expressam aspectos do domínio que podem variar para cada aplicação instanciada. Já os *frozen spots* expressam aspectos do domínio que são invariáveis para qualquer aplicação deste domínio. Para se produzir uma aplicação completa a partir de um *framework* é preciso instanciar o *framework*, implementando o código específico da aplicação em cada um dos *hot spots*. Os principais benefícios obtidos com a utilização de *frameworks* são: (i) a redução de custos na produção de novos softwares, dado o seu alto grau de reutilização, e (ii) o aumento da qualidade desses novos softwares produzidos, fato relacionado e condicionado à depuração do *framework* advinda de sua utilização em aplicações anteriores.

O *framework* para o sub-sistema de julgamento visa auxiliar o desenvolvimento de sistemas multi-agentes que utilizem o mecanismo de governança baseado em testemunhos apresentado neste trabalho. Este *framework* foi desenvolvido utilizando o *framework* ASF (*Agent Society Framework*) [22] como base. É importante lembrar que o mecanismo de governança e a arquitetura propostos neste trabalho podem ser implementados em qualquer plataforma, *framework* ou arquitetura de agentes que implementem os conceitos de sociedade de agentes, planos e ações.

PUC-Rio - Certificação Digital Nº 0511021/CA

4.1
O *Framework* ASF

O ASF é um *framework* orientado a objetos desenvolvido em Java que define um conjunto classes abstratas que devem ser estendidas para a implementação de aplicações multi-agentes. O objetivo do ASF é fornecer suporte a implementação dos principais conceitos de uma sociedade de agentes, que são: Agentes, organizações, papéis de agentes, ambiente e interações entre os agentes. Agentes são entidades ativas e autônomas, que desempenham papéis em organizações. Organizações são também entidades ativas e autônomas que agrupam um conjunto de agentes por um determinado conjunto de características. Empresas, departamentos e comunidades são exemplos de organizações. As organizações definem os papéis que os agentes podem desempenhar. Os papéis definem o comportamento dos agentes através da descrição de um conjunto de objetivos, planos, ações e crenças. Os ambientes representam o habitat das organizações e dos agentes.

O ASF é baseado em um modelo conceitual chamado TAO [23], que define um conjunto de abstrações que caracterizam sociedades de agentes. O *framework* também se baseia no modelo *Belief-Desire-Intention* (BDI) e, portanto, oferece suporte à implementação de objetivos, planos e crenças que os agentes podem possuir. No modelo BDI os agentes são entidades orientadas a objetivos. Seu comportamento é caracterizado pelos objetivos que possuem e pelos planos que executam para atingir esses objetivos. No ASF, agentes podem possuir diversos objetivos e planos. Eles podem definir diferentes estratégias para selecionar a ordem em que os objetivos serão atingidos e diferentes estratégias para a seleção dos planos que serão executados para atingirem os objetivos.

O ASF também oferece suporte a interação entre os agentes. Essa interação é realizada através da troca de mensagens realizada entre os agentes. As mensagens trocadas no ASF estão em conformidade com a especificação ACL (*Agent Comunication Language*) proposta pela FIPA [50].

O ASF é composto por vários módulos relacionados e cada módulo representa a abstração de uma entidade da sociedade de agentes. Os módulos mapeiam as entidades através de um conjunto de classes e de relacionamentos entre essas classes. Assim, cada módulo possui uma classe abstrata que representa uma entidade da sociedade de agentes e um conjunto de classes abstratas ou concretas que representam as propriedades dessa entidade.

PUC-Rio - Certificação Digital Nº 0511021/CA

- ***Agent Module*:** Este módulo é composto por um conjunto de classes e relacionamentos que representam os agentes e suas propriedades. Um agente é representado pela classe abstrata *Agent* que estende a classe *Thread* do Java. Várias *threads* podem estar associadas a classe *Agent* para representarem os vários papéis que o agente pode estar desempenhando. Existem classes abstratas para representarem os objetivos, as crenças, os planos e as ações, que são propriedades dos agentes. As classes *Goal*, *Belief*, *Plan* e *Action* representam essas abstrações e estão relacionadas com a classe *Agent* através de atributos que a classe *Agent* possui. Um plano é composto por um conjunto ações e está associado ao objetivo que ele atinge. As ações representam as tarefas que devem ser executadas pelo agente para atingir seu objetivo. A comunicação entre os agentes é realizada utilizando-se a classe *Message*, que se relaciona com a classe *Agent* através de dois atributos responsáveis por armazenar as mensagens recebidas e as mensagens enviadas de um agente. Os agentes da aplicação instanciada, seus objetivos, crenças, planos e ações, bem como as mensagens que eles enviam e recebem, devem ser representados através da implementação de classes concretas que estendem as classes definidas neste módulo.
- ***Agent Role Module:*** Este módulo representa os papéis que os agentes podem desempenhar. Um papel de agente é representado pela classe abstrata *AgentRole*. Um papel de agente pode definir objetivos, crenças, deveres, direitos e protocolos para o agente que o desempenha. Essas características são representadas através de atributos da classe *AgentRole* que estão associados às classes *Goal*, *Belief, Duty*, *Right* e *Protocol*. Os papéis de agentes definidos na aplicação instanciada devem ser representados por classes concretas que estendem a classe *AgentRole*.
- ***Organization Module*:** Este módulo é composto por um conjunto de classes e relacionamentos que representam as organizações e suas propriedades. Uma organização pode ser representada pelas classes abstratas *MainOrganization* ou *Organization*, que estende *MainOrganization*. A classe *MainOrganization* estende a classe *Thread* de Java. Organizações também possuem objetivos, crenças, planos e ações e, portanto, as classes *Goal*, *Belief*, *Plan* e *Action* também estão definidas como atributos da classe *MainOrganization*. As organizações definem os papéis que os agentes devem desempenhar e podem possuir sub-

PUC-Rio - Certificação Digital Nº 0511021/CA

organizações. Sub-organizações agem como agentes dentro de uma organização principal e, portanto também desempenham papéis definidos pela organização principal, representados por classes concretas que estendem a classe AgentRole. A relação entre organização e sub-organização é representada pelo relacionamento entre as classes *MainOrganizatio*n e *Organization*. Organizações que não desempenham papéis devem estender a classe *MainOrganization*, enquanto organizações que desempenham papéis, sendo sub organizações de uma organização principal, devem estender a classe *Organization*.

- ***Environment Module:*** Este módulo é composto por apenas uma classe abstrata que representa o ambiente de uma aplicação. Como as propriedades do ambiente dependem da aplicação instanciada, elas devem ser definidas na implementação da classe concreta a ser estendida da classe *Enviroment* definida neste módulo. Ambientes representam o habitat dos agentes, organizações e outros objetos, portanto a classe *Environment* define atributos para armazenar os agentes, organizações e outros tipos de objetos Java que habitam o ambiente.

O *framework* ASF oferece uma infra-estrutura necessária para a implementação de aplicações que utilizam os conceitos relacionados à sociedade de agentes. Ao desenvolver uma aplicação multi-agente utilizando o ASF, o desenvolvedor deverá se preocupar apenas com detalhes específicos do domínio da aplicação a ser implementada. Detalhes relacionados à infra-estrutura de uma sociedade de agente já foram implementados pelo *framework* e podem ser reutilizados apenas com a extensão de suas classes.

Um *framework* para sistema multi-agentes deve oferecer suporte aos conceitos essenciais da abordagem de agentes. Alguns autores em [24] apontam agentes, organizações, ambientes e interações como conceitos centrais desta abordagem. Em [25], os autores identificam papéis e grupos também como características importantes em sistemas multi-agentes. Por fim, os autores de [26][27] também definem que sociedades de agentes são compostas por agentes que desempenham papéis em organizações e que habitam ambientes. A utilização do *framework* ASF foi considerada para ser a base da implementação do *framework* de julgamento por modelar e oferecer suporte a todos esses conceitos considerados por [24][25][26][27], além de utilizar ACL como base para a camada de comunicação entre os agentes e as organizações. Embora outros *frameworks*, plataformas ou arquiteturas tenham sido propostos

PUC-Rio - Certificação Digital Nº 0511021/CA

na literatura [28][29][30][31], nenhum deles oferece, ao mesmo tempo, suporte a definição de ambientes, organizações e papéis de agentes [22].

4.2 O *Framework* de Julgamento

As funcionalidades do *framework* de julgamento são disponibilizadas através da implementação de um conjunto de classes que estendem as classes do *framework* ASF e que representam os agentes, seus planos, suas ações e os recursos do sistema. Na figura 4 ilustramos o *framework* de julgamento, juntamente com algumas classes do *framework* ASF. As classes que aparecem tracejadas na figura são as classes do *framework* ASF que foram estendidas pelo *framework* de julgamento.

O *framework* de julgamento apresenta um conjunto de pontos de extensões que devem ser implementados de acordo com as especificações e necessidades de cada aplicação instanciada. As classes que representam os pontos de extensão do *framework* estão ilustradas na figura pelo estereotipo <<hot-spot>> e são: o recurso *Testimony*, utilizado para representar os testemunhos enviados pelos agentes, o plano *JudgingTestimony* que efetua o julgamento dos testemunhos e o agente *Police*, cujo objetivo é verificar violações de algumas normas.

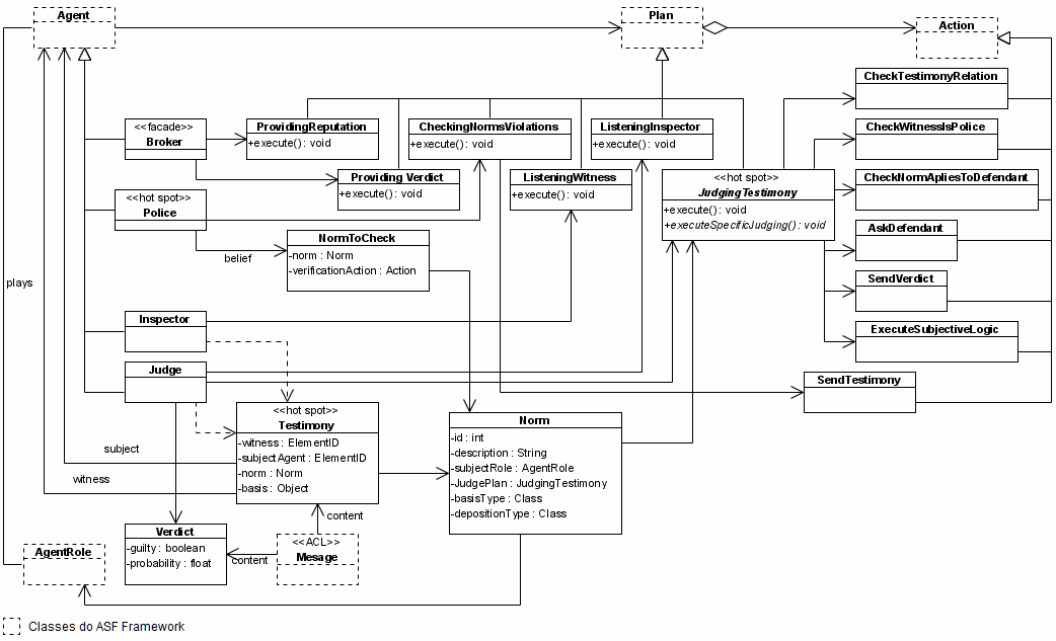

Figura 4: Arquitetura do sub-sistema de julgamento. Diagrama de classes do *framework* de julgamento

Cada uma das classes apresentada na figura 4 será detalhada a seguir:

PUC-Rio - Certificação Digital Nº 0511021/CA

- ***Os agentes Inspector:*** A responsabilidade destes agentes é receber os testemunhos enviados pelos agentes da aplicação. Este objetivo é alcançado através da execução de um único plano chamado *ListeningWitness*. Após receber o testemunho, o agente *Inspector* deve criar um agente *Judge* e enviar o testemunho recebido para este agente *Judge* criado.
- **O recurso *Testimony*:** Cada testemunho enviado pelos agentes da aplicação deve informar a norma que foi violada, o contexto que caracteriza a violação da norma (estado do sistema), o agente que violou a norma e o agente que enviou o testemunho. É importante notar que o contexto que caracteriza a violação depende de cada norma especificada na aplicação. Ou seja, se uma norma está associada a um protocolo de interação, o contexto pode ser, por exemplo, a mensagem que foi envida, responsável pela quebra do protocolo. Se uma norma está associada a políticas de acesso a recursos do sistema, o contexto pode ser, por exemplo, o recurso que foi indevidamente acessado. Sendo assim, o recurso *Testimony* foi implementado através de uma classe abstrata, caracterizando um *hot spot* do *framework*. Este deve ser estendido para representar contextos específicos de cada aplicação instanciada.
- **Os agentes *Broker*:** Agentes do tipo *Broker* têm como objetivo possibilitar a interação entre os sub-sistemas de julgamento, reputação e sanção. Os sub-sistemas de julgamento e reputação devem interagir quando o sub-sistema de julgamento necessita da reputação de algum agente da aplicação. O agente *Broker* possibilita essa interação através da execução do plano *ProvidingReputation*. Os sub-sistemas de julgamento, reputação e sanção devem interagir quando o sub-sistema de julgamento necessita enviar um veredicto aos outros dois sub-sistemas. O agente *Broker* possibilita essa interação através da execução do plano *ProvidingVerdict*.
- **O recurso *Verdict*:** Cada veredicto deve informar o testemunho que foi julgado e o resultado do julgamento (culpando ou inocentando o acusado) associado a sua probabilidade, conforme descrito na seção 3.2.1.1.
- **O recurso *Norm*:** Normas são descrições de como os agentes devem se comportar para que estejam de acordo com procedimentos previamente estabelecidos pela a aplicação. As normas definem as obrigações, as permissões e as proibições para os agentes de uma aplicação. Para julgar as violações, cada norma deve conter uma descrição, deve indicar a qual ou a quais papeis de agentes ela se aplica e um plano de ação que deverá

PUC-Rio - Certificação Digital Nº 0511021/CA

ser executado para julgar um testemunho enviado sobre a sua violação. Cada norma definida na aplicação pode apresentar características muito específicas, fazendo com que a estratégia de verificação de um testemunho seja igualmente específica. Por isso a necessidade da associação de cada norma com um plano de julgamento. Esta associação permite que o sub-sistema de julgamento seja flexível suficiente para tratar normas das mais diferentes naturezas.

- **Os agentes *Judge*:** São responsáveis por executar os planos que realizam o processo de julgamento. Quando um agente da aplicação percebe a violação de uma norma, ele deve enviar seu testemunho a um agente *Inspector*. Como dito anteriormente, o agente *Inspector* deve criar um agente *Judge* e repassar o testemunho recebido a este agente *Judge*. Todo testemunho deve informar a norma que foi violada e toda norma da aplicação deve estar associada a um plano de julgamento para ser executado quando da sua violação. Assim sendo, o agente *Judge* irá executar este plano associado a norma violada contida no testemunho.
- **O plano *ListeningWitness*:** Este plano é constantemente executado pelo agente *Inspector* e é responsável por receber os testemunhos enviados pelos agentes da aplicação. Os testemunhos são enviados através de mensagens ACL, cujos destinatários devem ser agentes do tipo *Inspector*. O plano *ListeningWitness* verifica a fila de mensagens do agente *Inspector* buscando por uma mensagem cuja performativa é *INF_TESTIMONY*. Para cada mensagem *INF_TESTIMONY* recebida, o agente *Inspector* verifica com cada agente *Judge* em execução (através da troca de mensagens) se o testemunho em questão está relacionado com o processo de julgamento em execução. Caso algum dos agentes *Judge* confirme a relação do novo testemunho com o seu processo de julgamento em execução, o agente *Inspector* simplesmente repassa esse testemunho para este agente *Judge*, para que ele possa ser utilizado no processo de julgamento em andamento. Caso todos os agentes *Judge* em execução não tenham confirmado a relação do novo testemunho com seu processo de julgamento, o agente *Inspector* deverá criar um agente *Judge* e enviar o testemunho recebido para este agente criado, iniciando um novo processo de julgamento.
- **O plano *ListeningInspector*:** Este plano é constantemente executado pelo agente *Judge* e tem como objetivo verificar se um novo testemunho está relacionado com o seu processo de julgamento em execução. Este

PUC-Rio - Certificação Digital Nº 0511021/CA

plano é responsável por receber as mensagens enviadas pelo agente *Inspector* (durante a execução do plano *ListeningWitness*), sobre o questionamento da relação entre um novo testemunho e o seu processo de julgamento em execução. Ao receber o questionamento do agente *Inspector*, a verificação da relação entre um novo testemunho com o processo de julgamento em execução é realizada através da execução da ação *CheckTestimonyRelation*. Caso a relação seja confirmada, esse novo testemunho é adicionado na lista de depoimentos que podem ser utilizados durante a execução do passo VI e VII do processo de julgamento. Além disso, uma resposta é enviada ao agente *Inspector*, confirmado o relacionamento do novo testemunho com o processo de julgamento. Caso contrário, uma resposta é enviada ao agente *Inspector,* alertando que processo de julgamento não está relacionado com o novo testemunho.

- **O plano *ProvidingReputation*:** O plano *ProvidingReputation* é executado por agentes do tipo *Broker* e tem a finalidade de fornecer a reputação de um agente da aplicação ao sub-sistema de julgamento, quando solicitada. Quando um agente *Judge* necessita da reputação de um agente da aplicação, ele deve solicitar essa reputação a um agente *Broker*. O plano *ProvidingReputation* é responsável por efetuar a comunicação com o sub-sistema de reputação, adquirir a reputação solicitada e enviá-la ao sub-sistema de julgamento.
- **O plano *ProvidingVerdict*:** O plano *ProvidingVerdic*, que é executado pelos agentes *Broker*, é responsável por enviar as decisões (veredictos) do sub-sistema de julgamento para os sub-sistemas de reputação e de sanção. Ao concluir um veredicto, o agente *Judge* deve solicitar ao agente *Broker* o envio desse veredicto aos sub-sistemas de reputação e de sanção, para que possam tomar as devidas providências em relação ao acusado e à testemunha.
- **O plano *JudgingTestimony*:** Este plano representa a estratégia base para qualquer processo de julgamento. O objetivo dessa estratégia é determinar a veracidade de um testemunho e concluir se o acusado realmente violou uma norma. Como foi descrito na seção 3.2.1.1, existe um conjunto de passos (ações) que são comuns ao processo de julgamento de qualquer norma. Esses passos consistem na execução do plano que se dá com as operações realizadas pelas ações *CheckTestimonyRelation* (passo I), *CheckWitnessIsPolice* (passo II), *CheckNormAppliesToDefendant* (passo III), *AskDefendant* (passo IV),

PUC-Rio - Certificação Digital Nº 0511021/CA

ExecuteSubjectiveLogic (passo VII) e *SendVerdict* (passo VIII). Os passos V e VI são específicos para cada norma e para cada aplicação instanciada, portanto, devem ser implementados através da extensão da classe *Action*. Uma vez implementadas essas as ações, é necessário estender a classe *JudgingTestimony* para fazer com que elas façam parte da estratégia de julgamento de cada uma das normas. Isso faz do plano *JudgingTestimony* um ponto de extensão do *framework* que deve ser obrigatoriamente estendido. Planos implementados a partir da extensão de *JudgingTestimony* executarão os passos I, II, III, IV, VII e VIII, que são independente da aplicação, além dos passo V e VI, que são dependentes da aplicação.

- **A ação *CheckTestimonyRelation*:** Esta ação é responsável pela execução do passo I do processo de julgamento, além de ser executada também pelo plano *ListeningInspector*. Existem dois aspectos importantes relacionados ao recebimento e julgamento de um testemunho: (i) tratar o recebimento de mais de um testemunho sobre a mesma violação (testemunhos de várias origens contendo o mesmo acusado e a mesma norma violada) durante a execução de um processo de julgamento e (ii) tratar o recebimento de mais de um testemunho sobre a mesma violação, após o processo de julgamento ter concluído o veredicto onde o acusado tenha sido considerado culpado.

  Sendo assim, o objetivo desta ação é verificar se o testemunho recebido está relacionado a algum outro processo de julgamento em execução ou já executado. Desta forma, vários testemunhos sobre uma mesma violação de norma não são tratados como violações diferentes, evitando que o agente acusado seja punido várias vezes pela mesma infração cometida, caso ele seja considerado culpado.

  A verificação da relação entre um novo testemunho com um processo de julgamento é feita através de comparações entre os atributos do novo testemunho com os atributos do testemunho que originou o processo de julgamento.
- **A ação *CheckWitnessIsPolice*:** Esta é a ação responsável pela execução do passo II do processo de julgamento. O objetivo desta ação é verificar se o agente que enviou o testemunho é um agente do tipo *Police*. Testemunhos enviados por agentes do tipo *Police* são considerados sempre verdadeiros e o acusado é sempre considerado culpado pela violação.

PUC-Rio - Certificação Digital Nº 0511021/CA

- **A ação *CheckNormAppliesToDefendant*:** A execução do passo III do processo de julgamento é realizada por esta ação. Durante a execução da ação *CheckNormsApliesToDefendant* o papel contido na descrição da norma é comparado com o papel desempenhado pelo acusado contido no testemunho. Papéis coincidentes indicam que a norma se aplica ao acusado. Papéis diferentes indicam que a norma não se aplica ao acusado. Neste último caso o testemunho é considerado falso.
- **A ação *AskDefendant*:** Esta ação é responsável pela execução do passo IV do processo de julgamento e é através dela que o agente *Judge* questiona o acusado a sua opinião sobre o testemunho. Caso o acusado responda que é inocente, o processo de julgamento prossegue com a execução do passo V, caso contrário, o processo de julgamento é finalizado e o acusado é considerado culpado.
- **A ação *ExecuteSubjectiveLogic*:** Esta ação é responsável pela execução do passo VII do processo de julgamento. Caso o processo de julgamento não tenha sido concluído após a execução do passo V, será necessário obter um consenso entre os depoimentos colhidos durante a execução do passo VI. Este consenso é obtido através da utilização de *subjective logic*, como foi descrito na seção 3.3.4, e todas as operações necessárias a obtenção desse consenso estão implementadas nesta ação.
- **A ação *SendVerdict*:** Esta ação é responsável pela execução do último passo do processo de julgamento. Esta ação é responsável por enviar o veredicto do processo de julgamento a um agente *Broker. O agente Broker* deve enviar este veredicto aos sub-sistemas de reputação e de sanção.
- **Os agentes *Police***: Como mencionado na hipótese IV, o mecanismo proposto não obriga que agentes enviem testemunhos quando observam violações de normas e nem garante que todas as violações sejam observadas pelos agentes da aplicação. Para minimizar os efeitos dessas duas situações é possível criar agentes especiais que devem ser extensões do agente *Police* (hipótese VI). O objetivo desses agentes especiais é identificar violações de algumas normas da aplicação. É importante identificar para quais normas o envio de testemunhos sobre sua violação não é de interesse de nenhum agente da aplicação e quais normas suas violações podem não ser observadas pelos próprios agentes da aplicação. Os agentes *Police* devem verificar violações relacionadas apenas as essas normas, devendo enviar seu testemunho no caso em que seja percebida alguma violação. Uma vez identificadas quais normas

PUC-Rio - Certificação Digital Nº 0511021/CA

devem ser verificadas, o *belief normsToCheck*, que agentes do tipo *Police* possuem, deve ser preenchido de forma a conter uma lista de objetos da classe *NormToCheck*. Essa classe deve armazenar a norma, cuja violação deve ser verificada, e a ação responsável por executar essa verificação. Essa ação deve ser implementada pelos desenvolvedores da aplicação e devem estender a classe *Action*. O testemunho desses agentes são sempre considerados verdadeiros, o que significa que o acusado será sempre considerado culpado.

- **O Plano *CheckingNormsViolations*:** Este plano é constantemente executado por agentes do tipo *Police* e é responsável por verificar violações de normas específicas, contidas no *belief normsToCheck* deste tipo de agente. Durante a execução deste plano, a lista de normas a serem verificadas é percorrida, e para cada item da lista, a ação associada, responsável por verificar a violação de norma, é executada. As ações de verificação devem inspecionar o ambiente em que os agentes estão inseridos tentando identificar mudanças que implicam na violação da sua norma associada. Caso seja identificada uma violação de norma, um testemunho contendo o contexto da violação (atributo *basis* da classe *Testimony*) deve ser criado e armazenado no *belief testimony* do agente *Police*. A última ação executada pelo plano é a ação *SendTestimony*, que é ativada apenas em casos onde a violação foi detectada e o testemunho foi criado.
- **A ação *SendTestimony*:** Esta ação é executada durante a execução do plano *CheckingNormsViolations* e é responsável por enviar um testemunho sobre a violação de uma norma para algum agente *Inspector*. Esta ação verifica a existência de um testemunho através do *belief testimony* e, em caso positivo, esse testemunho é enviado.

4.3 Utilizando o *Framework* de Julgamento

O *framework* de julgamento proposto deve ser utilizado junto com o *framework* ASF para construir uma aplicação multi-agentes que utilize o mecanismo de governança baseado em testemunhos, proposto no capítulo 3. A construção da aplicação está dividida em quatro etapas: (i) definição das normas, (ii) instanciação do mecanismo de governança através da extensão do *framework* de julgamento, (iii) instanciação da aplicação através da extensão do *framework* ASF e (iv) especialização dos agentes da aplicação para atenderem

PUC-Rio - Certificação Digital Nº 0511021/CA

às necessidades do mecanismo de governança, ou seja, implementação dos agentes de forma que eles sejam capazes de detectar violação das normas que são de seu interesse (aquelas que, quando violadas, afetam a sua execução) e enviar testemunhos e depoimentos quando necessário. Esta última etapa não é obrigatória, pois como ela trata da implementação dos agentes da aplicação e como o mecanismo proposto pode ser utilizado em sistemas abertos, não é possível garantir que os agentes participantes do sistema sejam implementados dessa forma. Porém agentes que desejem participar de um sistema regulado por este mecanismo devem ser especializados conforme será descrito a seguir para que possam se beneficiar do ambiente normativo.

Durante a etapa (i), os desenvolvedores deverão se preocupar em definir quais normas deverão ser respeitadas pelos agentes da aplicação ao longo de sua execução. Essas normas deverão determinar as permissões, obrigações e proibições de ações, de qualquer natureza, que são executadas pelos agentes da aplicação. Uma vez definidas conceitualmente as normas elas deverão ser representadas no sistema através da instanciação de objetos da classe Norm.

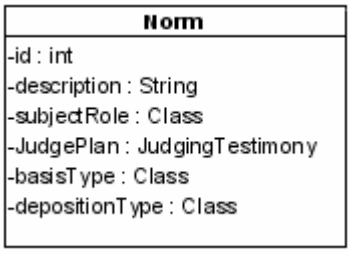

Figura 5: Classe *Norm*

O atributo *subjectRole* determina a qual tipo de papel de agente a norma se aplica e o atributo *judgePlan* determina o plano que será executado pelo agente *Judge* para julgar testemunhos sobre a norma associada. O atributo *basisType* determina o tipo do objeto que deverá servir de evidência para o testemunho e o atributo *depositionType* determina o tipo do objeto que deverá servir como depoimento, quando este for solicitado.

Na etapa (ii) os desenvolvedores deverão definir a estratégia do processo de julgamento de cada uma das normas definidas para a aplicação. As estratégias são implementadas através da extensão da classe *JudgingTestimony* e da implementação do método abstrato *executeSpecificJudging.* O método *executeSpecificJudging* deve conter apenas chamadas para as ações que devem tratar detalhes específicos da aplicação instanciada e da norma violada, tais como verificar um determinado recurso da aplicação ou solicitar um depoimento de algum agente da aplicação. A implementação deste método deve

PUC-Rio - Certificação Digital Nº 0511021/CA

representar a execução dos passos V e VI do processo de julgamento descrito na seção 3.2.1.1, já que no momento de sua execução, os passos I, II, III e IV, que foram implementados pelas classes *CheckTestimonyRelation*, *CheckWitnessIsPolice*, *CheckNormAppliesToDefendant*, *AskDefendant* definidas no *framework* respectivamente, já terão sido executados.

Quando um testemunho é enviado ao agente *Inspector*, este cria uma instância do agente *Judge*, repassando-lhe o testemunho. O agente *Judge*, verifica através da associação da norma violada (informada no testemunho) com o plano de julgamento, definido pelo atributo *judgePlan* qual plano deve ser executado para realizar o julgamento. O atributo *judgePlan* deve estar associado a uma classe que foi estendida de *JudgingTestimony*. Ao saber qual plano deve ser executado, o agente *Judge* chama o método *execute*, cuja implementação já é fornecida pela classe *JudgingTestimony.* Por sua vez, este método chama o método *executeSpecificJudging*, que deve ser implementado na classe estendida. A figura 8 ilustra o pseudo-código do método execute da classe *JudgingTestimony*.

```
/* Passo I: Verificar se o testemunho já foi julgado */
actionCheckTestimonyRelation = new CheckTestimonyRelation();
actionCheckTestimonyRelation.execute();
if (!beliefJudgmentProcessCanceled)
{
  /* Passo II: Verificar quem é a testemunha */
  actionCheckWitnessIsPolice = new CheckWitnessIsPolice();
  actionCheckWitnessIsPolice.execute();
  if (! beliefReachVerdict )
  {
    /* Passo III: Verificar se a norma se aplica ao acusado. */
    actionCheckNormAppliesToDefendant = new
                                        CheckNormAppliesToDefendant();
    actionCheckNormAppliesToDefendant.execute();
    if (! beliefReachVerdict )
    {
      /* Passo IV: Questionar o acusado se ele é realmente culpado. */
      actionAskDefendant = new AskDefendant();
      actionAskDefendant.execute();
      if (! beliefReachVerdict )
      {
         /* Passo V: Julgar o testemunho de acordo com a norma (passo
          *           dependente da aplicação).
          * Passo VI: Procurar depoimentos de outros agentes (passo
          *          dependente da aplicação). */
```

PUC-Rio - Certificação Digital Nº 0511021/CA

```
          this.executeSpecificJudging();
          if (! beliefReachVerdict )
          {
             /* Passo VII: Buscar um consenso entre os depoimentos. */
             actionExecuteSubjectiveLogic = new ExecuteSubjectiveLogic();
             actionExecuteSubjectiveLogic.execute();
           }
         }
       }
     }
   }
   /* Passo VIII: Estabelecer um veredicto. */
   actionSetVerdict = new SetVerdict();
   actionSetVerdict.execute();
}
```

Figura 6: Pseudo-código descrevendo o método *execute()* do plano *JudgingTestimony*.

Durante a etapa (iii), os desenvolvedores deverão instanciar agentes, definir e implementar objetivos, planos e ações específicos do domínio em questão. Esses itens devem ser implementados através da extensão de classes do *framework* ASF. Maiores detalhes sobre essa etapa estão fora do escopo deste trabalho e poderão ser encontrados em [22].

Por fim, na etapa (iv) os desenvolvedores devem adicionar características aos agentes da aplicação para que eles possam interagir com o mecanismo de governança. Interagir com o mecanismo de governança significa (i) enviar testemunho quando da percepção de uma violação de norma, e (ii) responder as mensagens enviadas pelo agente *Judge*. Para enviar um testemunho, um agente da aplicação precisa instanciar um objeto da classe *Testimony* ou de uma classe que tenha sido estendida de *Testimony* (caso tenha ocorrido alguma necessidade de especialização).

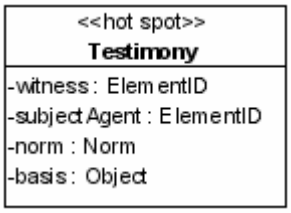

Figura 7: Classe *Testimony*

O atributo *witness* deve identificar o agente que envia o testemunho, já o atributo *subjectAgent* deve identificar o agente que cometeu a infração. O atributo *norm* deve identificar a norma que foi violada, enquanto que o atributo *basis* deve conter o fato, no qual o testemunho é baseado, e deve ser do tipo

PUC-Rio - Certificação Digital Nº 0511021/CA

determinado na definição da norma através do atributo *basisType*. Uma mensagem recebida ou enviada, um recurso do sistema, um conhecimento previamente adquirido pelo agente, como um contrato ou um recibo são exemplos de fatos que podem estar contidos no atributo *basis*. O agente deverá, então enviar uma mensagem para um agente que desempenha o papel de *Inspector* cuja performativa é *INF_TESTIMONY* e cujo conteúdo é este objeto instanciado. Exemplos de testemunhos serão apresentados no capítulo 5.

O agente *Judge* pode enviar dois tipos de mensagens aos agentes da aplicação. O primeiro tipo de mensagem é enviado ao agente acusado da violação, questionando se ele realmente violou a norma. O segundo tipo de mensagem é a solicitação de depoimentos, que pode ser realizada durante a execução do passo V, caso um veredicto ainda não tenha sido concluído. Os agentes da aplicação devem estar preparados para receberem mensagens dessas naturezas e enviar respostas ao agente *Judge*. Por tanto, os agentes da aplicação devem possuir objetivos e planos que recebam essas mensagens e enviem as respectivas respostas. No tipo de mensagem questionando a opinião do acusando em relação a sua culpa, o agente *Judge* envia uma mensagem cuja performativa é *JUD_ASK_DEFENDANT* e cujo conteúdo é a norma que foi violada. O acusado deve responder a mensagem com a performativa *JUD_ANSWER_DEFENDANT* e com o conteúdo booleano. Caso o acusado não responda a mensagem do agente *Judge*, ele será julgado à revelia. Mensagens de solicitação de depoimentos devem ser enviadas para agentes que supostamente possuem interesse na não violação da norma. A performativa dessas mensagens deve ser *JUD_ASK_DEPOSITION* e o conteúdo deve ser a norma violada. O agente ao qual foi solicitado o depoimento deve responder a mensagem com a performativa JUD_ANSWER_DEPOSITION e com o conteúdo solicitado. O conteúdo deverá ser um objeto do tipo definido na norma, através do atributo *depositionType*.

4.4 Um Mecanismo de Detecção de Violações de Normas

Como já mencionado anteriormente, para que o mecanismo de governança proposto possa regular o comportamento dos agentes da aplicação é preciso que esses agentes enviem testemunhos relatando fatos que estão relacionados à violações de normas. Para que esses agentes possam enviar os testemunhos, eles devem ser capazes de não só perceberem fatos ou eventos do ambiente, mas também relacionar esses fatos e eventos a possíveis

PUC-Rio - Certificação Digital Nº 0511021/CA

violações de normas. Portanto, percebe-se que ao implementar um agente da aplicação, o desenvolvedor desse agente deve não só se preocupar com aspectos do domínio da aplicação em questão e com as normas que esse agente deve seguir, como também com a detecção de violações de normas que se aplicam a ele próprio e também com normas que se aplicam a outros agentes da aplicação, cujas violações afetam sua execução. Isso gera uma complexidade extra no desenvolvimento desse agente.

Para minimizar o impacto da detecção das violações sobre desenvolvimento dos agentes da aplicação, o *framework* de julgamento dispõe do agente *ApplicationAgent* e de um conjunto de *actions* e outras classes capazes de realizar essa tarefa, conforme será ilustrado na Figura 9. A classe *ApplicationAgent* é responsável por executar esse conjunto de ações que realizam a tarefa de detecção de violações. Ao estender *ApplicationAgent*, um agente da aplicação adquire, por herança, a capacidade de detectar a violação de normas.

A detecção das violações realizada com suporte fornecido pelo *framework* se baseia em três aspectos: (i) no conjunto de *beliefs* do agente e qualquer alteração realizada sobre esses *beliefs*, (ii) no conjunto de mensagens que o agente recebe ou envia e (iii) no conjunto de regras de verificação que o desenvolvedor do agente deve especificar e instanciar.

O mecanismo de verificação consiste em realizar comparações entre os *beliefs* modificados e as mensagens enviadas e recebidas do agente com o seu conjunto de regras de verificação, determinado no momento da instanciação do agente. Portanto, é necessário que os *beliefs* e as mensagens enviadas e recebidas sejam monitorados conforme suas alterações, envios e recebimentos respectivamente. Para isso, tanto a estrutura que implementa o conjunto de *beliefs* do agente, quanto a sua fila de mensagens implementados na classe *Agent* do *ASF Framework* foram modificados através da sua extensão, com a criação da classe *ApplicationAgent* para a implementação do padrão de projeto *Observer*. Com o padrão *Observer* é possível implementar uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda de estado, todos os seus dependentes são notificados automaticamente [48]. Assim é possível realizar o monitoramento de todos os *beliefs* e todas as mensagens do agente sem maiores impactos na performance da aplicação.

O conjunto de regras de verificação determina quais normas devem ser verificadas e quais condições determinam a violação de cada norma. Nem todas as normas definidas para a aplicação a ser instanciada precisam ser verificadas

PUC-Rio - Certificação Digital Nº 0511021/CA

por um agente. Um agente precisa verificar somente as normas cujas violações afetem sua execução. A especificação e instanciação desse conjunto de regras são de responsabilidade do desenvolvedor do agente da aplicação e deve ser baseada na especificação das normas da aplicação. Essa tarefa é realizada através da instanciação de objetos das classes *ViolationCheckRule* e *ComparisonRule*, apresentadas na figura 5. Uma vez instanciada a classe *ViolationCheckRule*, a ação *AddToVerificationList* deve ser executada para que a regra seja adicionada na lista de verificações. Apenas as normas adicionadas na lista de verificações são verificadas. Para cada norma a ser verificada, um objeto do tipo *ViolationCheckRule* deve ser instanciado e adicionado na lista *checkNormViolationList*. Essa lista é um *belief* que agentes que estendem *ApplicationAgent* possuem.

Cada vez que um *belief* é modificado ou uma mensagem é enviada ou recebida a ação *NotifyChange* é executada[1]. Essa ação é responsável por executar a ação *CheckNormViolation*. *CheckNormViolation* percorre a lista de regras de verificação (*checkNormViolationList*) e executa a comparação do conteúdo da notificação com o conteúdo de cada regra de verificação. A forma como a comparação é realizada está especificada na regra. O resultado da comparação de uma regra igual a verdadeiro representa a violação da norma. Neste caso, a ação *AlertNormViolation*, que é um *hot spot* do *framework* e deve ser estendida pelo desenvolvedor, é executada. Esta ação deve ser responsável por criar o testemunho e enviá-lo ao sub-sistema de julgamento.

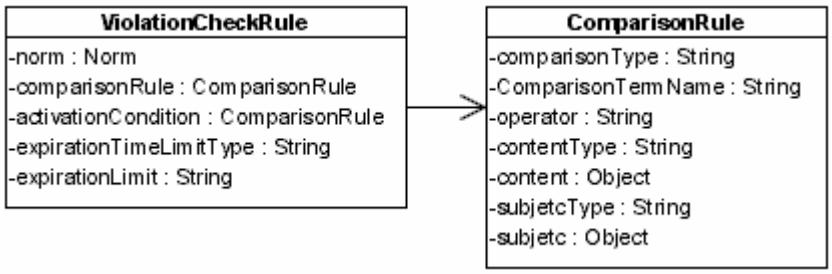

Figura 8: Classes *ViolationCheckRule* e *ComparisonRule* que representam as regras de verificação de violação de normas.

A tabela 2 descreve os atributos da classe *ViolationCheckRule*, enquanto que a tabela 3 descreve os atributos da classe *ComparisonRule*.

Norm	Indica a norma cuja violação deve ser verificada.
comparisonRule	Indica a regra de verificação que determina a violação da norma

[1] Utilização do padrão *Observer* como já mencionado.

PUC-Rio - Certificação Digital Nº 0511021/CA

activationCondition	Uma norma pode ser aplicável segundo alguma condição. Portanto, uma norma pode estar inicialmente inativa. Para normas que estão inicialmente inativas, este atributo indica a regra de verificação que determina a ativação da norma. Se este atributo for nulo a norma é ativada no momento em que a regra de verificação é inserida na lista.
expirationLimitType	Indica o tipo de limite de tempo em que a norma está ativa. Pode ser: literal | expressão em dias | expressão em horas.
expirationLimit	Caso a norma seja do tipo "obrigação", indica o prazo máximo em que a norma deve ser cumprida. Caso a norma seja do tipo "proibição", indica até quando a proibição está ativa. Após esse limite a norma será desativada. Pode ser nulo, indicando que a norma não possui limite de tempo para expirar. Se expirationLimitType = 'literal', então expirationLimit representa uma data. Se expirationLimitType = 'expresão em dias', então expirationLimit representa o número de dias. O prazo final será a data em que a norma foi ativada mais o numero de dias determinado por expirationLimit. Se expirationLimitType = 'expressão em horas', então expirationLimit representa o número de horas. O prazo final será a data em que a norma foi ativada mais o numero de horas determinado por expirationLimit.

Tabela 2. Descrição dos atributos da classe *VerificationCheckRule*

comparisonType	Indica qual o tipo de recurso do agente que será utilizado na verificação. Pode ser: belief | message In | message Out.
comparisonTermName	Se comparisonType = belief, então comparisonTermType indica o nome do belief que será utilizado na verificação. Se comparisonType = message In ou message Out, então comparisonTermType indica a performativa que indica a mensagem que será utilizada na verificação.
Operator	Operador de comparação que determina a ocorrência da violação. Pode ser: =; ≠; <;>; >=; <=
contentType	Pode ser: belief | literal. O conteúdo do atributo representado por comparisonTerm será comparado com o conteúdo do atributo representado por content. O atributo content pode representar um belief ou pode ser fixo.
Content	Valor a ser comparado com o conteúdo do atributo representado por comparisonTerm.

PUC-Rio - Certificação Digital Nº 0511021/CA

subjectType	Indica o tipo do sujeito ao qual a norma se aplica. Pode ser: belief : Indica que o sujeito é o conteúdo de um belief; remetente: Indica que o sujeito é o remetente da mensagem (IN) utilizada na verificação; destinatário: Indica que o sujeito é o destinatário da mensagem (OUT) utilizada na verificação. Literal: Indica que o conteúdo de subject representa o agente ao qual a norma se aplica.
Subject	Agente que está sujeito a aplicação da norma.

Tabela 3.Descrição dos atributos da classe *ComparisonRule*.

A figura 9 ilustra o conjunto de ações que são executadas para a verificação de violação de normas.

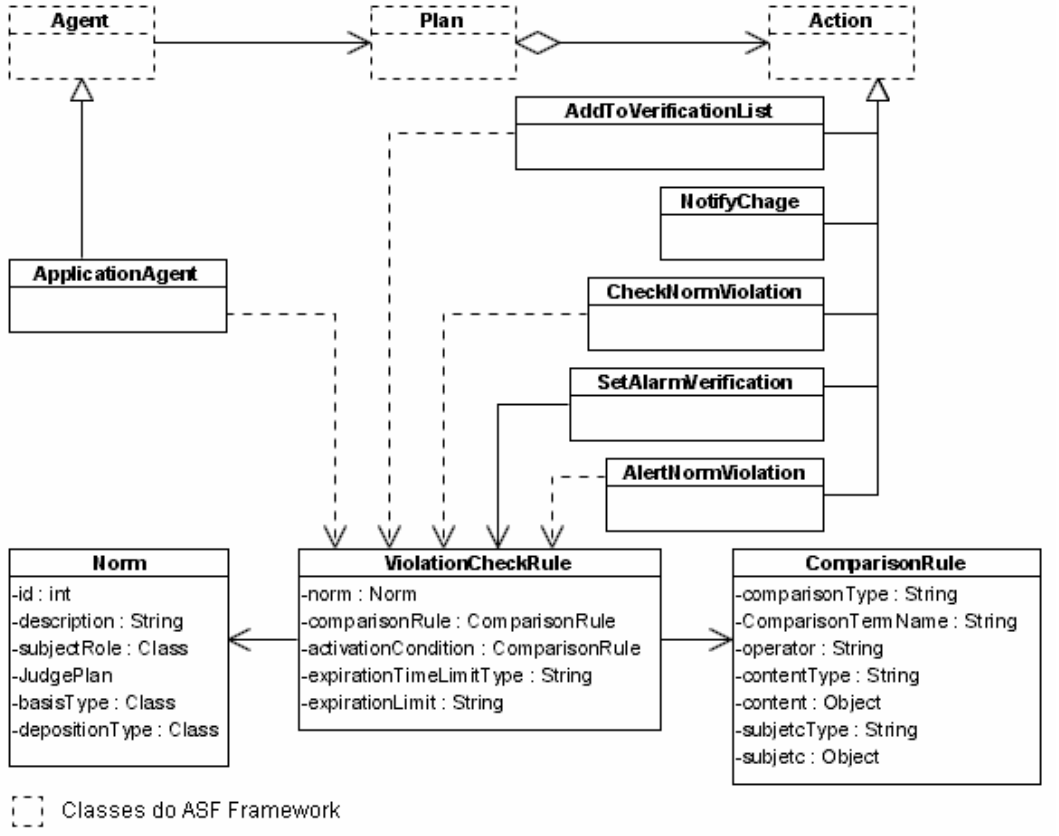

Figura 9: Mecanismo de detecção de violação de normas

A classe *ApplicationAgent*, por tanto, implementa o suporte necessário à percepção de violações de normas para que os agentes da aplicação possam enviar seus testemunhos ao perceberem uma infração. Ao implementar um agente da aplicação que não estende a classe *ApplicationAgent*, os desenvolvedores deverão se preocupar em implementar todas as funcionalidades para a detecção de violações de norma. Porém, ao implementar um agente da aplicação estendendo a classe *ApplicatioAgent* do *framework* de Julgamento os desenvolvedores deverão se preocupar apenas em instanciar as

PUC-Rio - Certificação Digital Nº 0511021/CA

regras de verificação de violação para as normas que se deseja detectar violações.

Apesar de permitir a verificação de diferentes tipos de normas para diferentes domínios, este mecanismo de verificação apresenta algumas limitações. Toda verificação de violação de norma é baseada na lista de *beliefs* e na fila de mensagens do agente. Por tanto, para qualquer norma a ser verificada, é preciso que exista algum *belief* ou alguma mensagem relacionada com a norma, que segundo alguma condição, determine a sua violação ou a sua ativação. Por exemplo: (i) A alteração de um *belief x* do agente, segundo alguma condição, determina a violação de uma norma. (ii) O recebimento ou o não recebimento de uma mensagem *x* em um determinado intervalo de tempo determina a violação de uma norma. (iii) O envio de uma mensagem *x* determina a ativação de uma norma. Caso exista uma norma onde não seja possível relacionar algum *belief* ou alguma mensagem com a sua violação, não será possível utilizar este mecanismo para a verificação desta norma. Uma outra limitação imposta por este mecanismo está relacionada com o tipo do conteúdo dos *beliefs* e das mensagens. A comparação só pode ser realizada se os conteúdos de *beliefs* e mensagens forem de tipos primitivos da linguagem Java[2] ou do tipo *String*, *Date*, *Number* ou *Boolean*. *Beliefs* e mensagens com conteúdos do tipo de classe definida pelo desenvolvedor não podem ser utilizadas para este mecanismo de verificação, pois o mecanismo não sabe como comparar esses conteúdos. Para esses casos que não são cobertos pelo mecanismo de verificação proposto, o desenvolvedor do agente terá que implementar sua própria estratégia de detecção de violação. Portanto, o mecanismo proposto pode ser utilizado em conjunto com outras formas de detecção de violação de normas, quando necessário.

4.5 Escalabilidade e Execução Contínua do Mecanismo de Governança

Aplicar uma perspectiva organizacional a sistemas multi-agente é visto como uma forma de quebrar a complexidade dos problemas. Embora alguns sistemas simples possam ser compostos por uma única organização, o crescimento da complexidade dos problemas faz com que princípios como modularização e distribuição sejam aplicados, sugerindo que um sistema multi-agente seja decomposto em várias organizações [49].

PUC-Rio - Certificação Digital Nº 0511021/CA

Sendo assim, sistemas multi-agente de larga escala podem ser compostos por um conjunto de organizações, agrupados em uma estrutura hierárquica, onde agentes desempenham papéis definidos. Nesses sistemas, cada organização pode definir um conjunto de sub-organizações e uma sub-organização deve pertencer apenas a uma única super-organização. Cada organização pode definir um conjunto de normas que deve ser seguido pelos agentes que desempenham papéis na organização. As normas definidas em uma organização também são válidas para as suas sub-organizações e devem ser seguidas pelos agentes que desempenham papéis nessas sub-organizações. Uma norma definida em uma organização não deve estar em conflito com uma norma definida em sua super-organização. Portanto, as normas definidas em uma sub-organização podem apenas ser mais restritivas que as normas definidas em sua super-organização. Além disso, organizações que não pertençam a uma mesma rede hierárquica não devem possuir normas conflitantes entre si, já que agentes que desempenham papéis em organizações de diferentes redes hierárquicas podem interagir uns com os outros. Agentes que desempenham papéis em diferentes organizações, ao interagirem, devem respeitar as normas estabelecidas em ambas organizações.

Questões de performance, escalabilidade e execução contínua são sempre abordadas em sistemas multi-agentes abertos. Para que essas questões sejam tratadas em aplicações que utilizam o mecanismo de governança proposto, cada organização deve possuir (instanciar) seu próprio mecanismo de governança. Portanto, além de possuir agentes que desempenham os papéis definidos, cada organização possuirá também os agentes que implementam cada um dos sub-sistemas definidos na arquitetura do mecanismo de governança proposto na seção 3.2. A distribuição do mecanismo de governança em diversas organizações (que também podem estar distribuídas geograficamente) proporciona um aumento do nível de escalabilidade e de execução contínua, já que todo o processamento necessário à manutenção de um ambiente regulado está distribuído. Os processos de julgamento são executados de forma independente dentro de cada organização bem como as reputações dos agentes, que são mantidas para cada organização.

Os julgamentos são realizados de acordo com os testemunhos enviados para as organizações onde os agentes desempenham os papéis. Assim, um

[2] Limitação pela utilização da linguagem no desenvolvimento do *framework* de julgamento.

PUC-Rio - Certificação Digital Nº 0511021/CA

agente deve enviar um testemunho para o sub-sistema de julgamento da organização a qual ele pertence. Os agentes testemunha e acusado podem desempenhar papéis em diferentes organizações. Dessa forma, o processo de julgamento será realizado pela organização que recebeu o testemunho, caso o testemunho seja relativo a uma norma estabelecida pela organização que o recebeu. Caso o testemunho seja relativo a uma norma estabelecida em uma outra organização, a organização que recebeu o testemunho deverá encaminhá-lo para a organização que estabeleceu a norma. O resultado do processo de julgamento será enviado para o sub-sistema de reputação e para o sub-sistema de sanção da mesma organização onde esse testemunho foi julgado. Caso testemunha e acusado desempenhem papéis em diferentes organizações, o resultado do processo de julgamento será enviado para os sub-sistemas de reputação e sanção de cada uma das organizações onde esses agentes desempenham esses papéis. Cada uma das organizações deverá, independentemente, agir de acordo com o que foi especificado na norma indicada no testemunho. Como dito anteriormente, agentes que desempenham papéis em diferentes organizações, ao interagirem, devem respeitar as normas de ambas as organizações. Sendo assim, não importa em qual organização o processo de julgamento foi realizado, seu resultado será válido para qualquer uma das organizações envolvidas.

PUC-Rio - Certificação Digital Nº 0511021/CA