3
A Arquitetura NeMaSA

3.1
Introdução

A arquitetura NeMaSa tem como objetivo principal possibilitar a gerência de falhas em redes legadas de telecomunicações de forma escalável. Geralmente tais redes são controladas e administradas por sistemas centralizados de gerência, muitas vezes seguindo padrões tais como o TMN. Tais sistemas concentram informações essenciais para procedimentos de diagnóstico e correção de falhas. Caso tais procedimentos sejam automatizados e executados com muita freqüência, corre-se o risco de que o funcionamento tanto do sistema de gerência como o do sistema de diagnóstico sejam prejudicados devido à inevitável sobrecarga daquele.

Este trabalho traz como contribuição a proposta de um mecanismo para se contornar este problema de sobrecarga através da distribuição pela rede das informações necessárias para os procedimentos de diagnóstico. Tal distribuição é realizada com o auxílio de agentes de software que extraem as informações do sistema centralizado de forma mais organizada. Desta forma, é possível que os agentes responsáveis por executar os procedimentos de diagnóstico e correção de falhas desempenhem suas atividades sem a necessidade de uma comunicação direta com o sistema centralizado, o que alivia a carga sobre o mesmo.

3.2
Motivação

Uma das principais motivações para a realização do presente trabalho foi proveniente da realidade observada em uma rede de dados de uma das maiores empresas prestadoras de serviços de telecomunicações no Brasil. O núcleo da rede estudada é baseado no protocolo ATM [I.321]. Há, entretanto, outras tecnologias com uma significativa presença na rede, como o Frame Relay [I.233] e o TDM (circuitos de dados baseados em multiplexação determinística). Estas tecnologias enquadram-se no nível dois do modelo OSI e são todas baseadas na técnica de circuito virtual, em que as rotas entre dois nós são definidas preliminarmente e mantêm-se inalteradas até o fim da transmissão de dados. Este modelo diverge significativamente do modelo de datagramas (adotado pelo protocolo IP, por

exemplo), em que as rotas são definidas dinamicamente em cada nó na medida em que os pacotes percorrem a rede.

No caso particular da rede estudada, é muito comum o uso de circuitos virtuais permanentes, em que recursos da rede são reservados de forma permanente para clientes, criando conexões dedicadas entre duas de suas instalações, como, por exemplo, duas filiais de uma empresa em cidades diferentes. A criação e a manutenção destes circuitos são efetuadas através de um sistema centralizado de gerência que abrange a rede em sua quase totalidade. Também cabe a este sistema legado a tarefa de monitorar a rede, detectando falhas em equipamentos e determinando novas rotas para os circuitos afetados. Devido às dimensões desta rede, que abrange boa parte do território brasileiro, este sistema encontra-se sobrecarregado. De fato, alguns tipos de monitoramento menos críticos tiveram de ser desativados para permitir que o sistema pudesse continuar exercendo tarefas mais importantes de gerência de elementos e circuitos.

Um dos principais objetivos da equipe que administra esta rede é o de reduzir o tempo para o diagnóstico de falhas em equipamentos e a descoberta de erros de configuração. Entende-se que esta redução do tempo de diagnóstico resulte em uma redução dos custos operacionais necessários para manter a rede em funcionamento. Para suprir esta necessidade foi desenvolvido um sistema de diagnóstico baseado em heurísticas. Este sistema busca na rede (tanto em equipamentos quanto em sistemas de gerência) os sintomas de problemas mais comuns ou mais críticos, identifica falhas e, quando possível, as corrige.

Um detalhe importante, entretanto, é que, apesar de o sistema desenvolvido ter automatizado o processo de diagnóstico de falhas, ele ainda depende da intervenção de um operador de rede (humano) que solicite explicitamente a execução de testes em um determinado circuito ou equipamento. Por esta razão, um próximo passo no sentido de possibilitar uma redução ainda maior nos custos de operação desta rede seria prover um gerenciamento mais autônomo, com uma menor intervenção direta dos operadores de rede. Desta forma seria possível identificar algumas falhas antes mesmo que elas cheguem a afetar significativamente o usuário.

Para que esse objetivo seja alcançado, é importante monitorar equipamentos e circuitos de forma contínua. Em muitos casos, pode também ser necessário executar testes custosos em termos de processamento. Um possível exemplo de teste custoso seria um em que se acesse uma base temporal para identificar desvios em relação a um comportamento historicamente observado. Tais requisitos, associados à grande dimensão da rede, sugerem que se dê especial atenção à questão da escalabilidade quando da formulação de uma solução para este problema. É neste contexto de busca de uma maior automação nos mecanismos de detecção de

falhas, levando-se em consideração questões de escalabilidade, que se enquadra a arquitetura apresentada neste trabalho.

3.3 Descrição Geral

3.3.1 Principais Desafios

O principal objetivo que se pretende alcançar através da arquitetura proposta neste trabalho é a aplicação de testes em equipamentos para o diagnóstico e correção de falhas na rede. Tais testes devem ser realizados de forma que não haja sobrecarga sobre a rede como um todo, nem sobre nenhum de seus elementos em particular. Um ponto importante é que a arquitetura proposta deve se adequar aos requisitos e às limitações da rede de telecomunicações previamente apresentada.

Muitas das fontes de informação necessárias para o diagnóstico de falhas estão dispersas pela rede. O exemplo mais nítido disto são as informações sobre o estado dos equipamentos que são obtidas diretamente dos próprios equipamentos. Como estas fontes de dados estão distribuídas, é bastante razoável o emprego de alguma técnica também distribuída para a coleta e manipulação destas informações. Uma vantagem desta estratégia distribuída é que ela contribui naturalmente para uma melhor escalabilidade da arquitetura.

Há, entretanto, uma fonte de dados fundamental para o funcionamento da solução aqui proposta que se encontra concentrada em um repositório centralizado. Esta fonte de dados é o sistema de gerência legado previamente citado. As informações mais relevantes que ele contém são a topologia e o inventário da rede. De fato, a maioria das informações que o sistema de diagnóstico de falhas citado anteriormente extrai deste repositório para efetuar seus testes são dados de topologia e inventário da rede. A estratégia de acesso a estes dados representa um primeiro desafio a ser superado, já que informações de topologia e o inventário da rede são essenciais para o diagnóstico de algumas falhas. Por outro lado, o acesso a este sistema de gerência representa um gargalo que pode comprometer a escalabilidade da solução proposta. Em um caso mais extremo, pode-se até mesmo prejudicar o funcionamento de tal sistema de gerência (e, por conseqüência, de toda a rede) caso haja uma demanda muito grande de informações de topologia solicitada simultaneamente. Como será visto a seguir, a solução aqui proposta utiliza um mecanismo para acessar este repositório centralizado de uma forma racional, evitando demandas excessivas sobre o mesmo.

Outro importante desafio deste trabalho é permitir a criação de soluções mais autônomas de gerência de falhas baseadas na arquitetura proposta. Em uma rede de

telecomunicações em produção, o tempo economizado no processo de diagnóstico e correção de uma falha é proporcional a uma redução nos custos de operação da rede. Por esta razão é que a autonomia é tão importante neste domínio. Idealmente, as falhas devem ser identificadas e tratadas antes mesmo de se manifestarem, ou pelo menos, assim que se manifestarem.

Há pelo menos duas formas de se imprimir autonomia na gerência de falhas em redes. Uma forma, simples e universalmente utilizada, é coletar estatísticas e alarmes enviados por equipamentos e tomar ações baseadas nas informações recebidas. Outra forma, mais complexa e até por isso menos comum, é a previsão de falhas baseada em dados coletados da rede. Normalmente um perfil histórico do comportamento da rede é traçado, e este é continuamente comparado com o estado presente da rede. Quaisquer desvios ou diferenças significativas entre o perfil histórico e o estado atual podem indicar a ocorrência de falhas. Em alguns casos, técnicas baseadas em inteligência artificial podem ser aplicadas para efetuar estas comparações.

A grande vantagem do método de monitoramento de estatísticas e alarmes é que as evidências ou sintomas das falhas são descobertos pelos próprios equipamentos, cabendo ao sistema de gerência de falhas apenas a tarefa de filtrá-las, interpretá-las e tomar as medidas corretivas cabíveis. A principal desvantagem deste método é que em muitos casos, os problemas decorrentes das falhas já se manifestaram e afetaram o funcionamento da rede.

A principal vantagem do método de previsão de falhas é que ele torna possível identificar indícios de eventuais falhas antes mesmo que elas venham a ocorrer. Desta forma, medidas corretivas podem ser tomadas antes que o funcionamento da rede venha a ser afetado significativamente. Por outro lado, sua grande desvantagem é a relativa complexidade de implementação dos mecanismos de predição. Tais mecanismos normalmente exigem a criação e manutenção de bases com dados históricos sobre o funcionamento da rede, e aplicam técnicas estatísticas ou mesmo de inteligência artificial.

Um terceiro desafio na concepção da arquitetura proposta é mantê-la suficientemente flexível para que possa ser aproveitada sem maiores alterações em outras redes de telecomunicações semelhantes. Esta não é uma questão fundamental, uma vez que o objetivo principal deste trabalho é atender os problemas da rede previamente apresentada. Entretanto, sempre que possível, é interessante adotar mecanismos que permitam uma fácil adaptação da arquitetura proposta a uma nova rede, com novos tipos de falhas. Como será visto adiante, o mecanismo de definição e execução dos testes permite que diferentes tipos de testes sejam desempenhados na rede através da arquitetura proposta.

3.3.2
Soluções Propostas

Levando-se em consideração a razoabilidade do uso de técnicas distribuídas e os requisitos de autonomia e flexibilidade, o uso de agentes de software neste problema é bastante apropriado, conforme pode ser depreendido da discussão das seções anteriores sobre o uso de agentes de software no domínio de gerência de redes.

Por essa razão, foi concebido um modelo baseado em agentes de software distribuídos pela rede, que executam testes em equipamentos através de protocolos comuns de gerência de redes, agindo diretamente sobre os nós ou sistemas de gerência. Esta solução permite uma ação mais intensa sobre os equipamentos do que seria possível com um modelo centralizado. Além disto, esta solução também permite distribuir a carga de processamento de testes mais custosos em diversas máquinas, uma vez que os agentes de software podem agir de forma independente entre si.

Outra vantagem no uso de agentes de software é que eles representam um meio bastante apropriado para a criação de sistemas autônomos e pró-ativos. Apesar de o aspecto de pró-atividade na arquitetura proposta não ser profundamente abordado neste trabalho, há perspectivas de futuros desenvolvimentos nesta área (vide o Capítulo 6).

A principal desvantagem do uso de agentes de software é que a arquitetura proposta se torna moderadamente complexa. Há diversas entidades (agentes) agindo de forma relativamente independente, há uma maior complexidade nas interações entre estas entidades, o que resulta em mais possíveis formas de o sistema falhar e não atingir seus objetivos. Contudo, considera-se que as vantagens deste modelo baseado em agentes de software sejam fortes o bastante para justificar tal proposta.

A solução proposta para se contornar a questão do acesso ao repositório centralizado foi distribuir pela rede a informação de topologia contida no repositório centralizado. Para isto são utilizados agentes de software projetados especificamente com o propósito de obter e difundir estes dados. Para viabilizar o emprego desta estratégia, a rede pode ser dividida em partições ou sub-redes, que por sua vez também podem ser subdivididas em outras sub-redes, tantas vezes quantas necessárias. A manutenção da informação de topologia de cada uma dessas sub-redes fica a cargo de um agente de software que centraliza nele as solicitações de informações de topologia da sub-rede pela qual é responsável. A Figura 3.1 representa tal organização, ilustrando a decomposição da rede nacional em três sub-redes, e a decomposição da sub-rede do Rio de Janeiro em outras três sub-redes. Esta estratégia de divisão da rede em níveis hierárquicos permite aliviar a carga que inevitavelmente surgiria sobre o sistema de gerência centralizado.

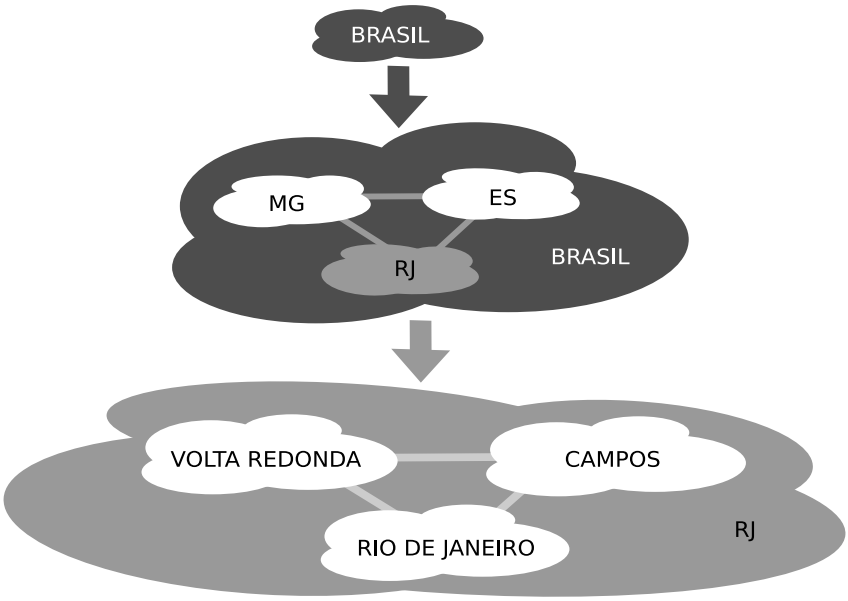

Figura 3.1: Hierarquia de redes

Com esta organização em mente, este trabalho propõe a modelagem do sistema com o uso de quatro tipos de agentes de software, listados a seguir:

Agente de gerência. Serve de ligação entre o sistema de gerência centralizado e os demais agentes. Tem a responsabilidade de controlar os acessos aos recursos do sistema de gerência centralizado, evitando que este seja eventualmente sobrecarregado por requisições dos demais agentes;

Agente de topologia. Tem como principal responsabilidade fornecer informações de topologia de sua sub-rede aos demais agentes (exceto o de gerência). Ele deve implementar mecanismos para manter-se atualizado em relação à topologia real mantida na base do sistema de gerência. Para tanto, ele pode comunicar-se diretamente com o agente de gerência ou através de outros agentes de topologia, valendo-se de consultas diretas às bases de topologia mantidas por estes agentes ou registrando-se para o recebimento de notificações de mudanças de topologia;

Agente de teste. Monitora e age diretamente sobre equipamentos da rede, verificando o estado destes e coletando estatísticas de seu funcionamento. Este comportamento é realizado através de *testes* solicitados por outros agentes;

Agente de usuário. Serve como interface entre o sistema e seus usuários, interpretando os parâmetros de entrada dos usuários, disparando as ações que forem necessárias e retornando aos usuários o resultado destas ações.

A Figura 3.2 ilustra a distribuição desses agentes em uma rede fictícia.

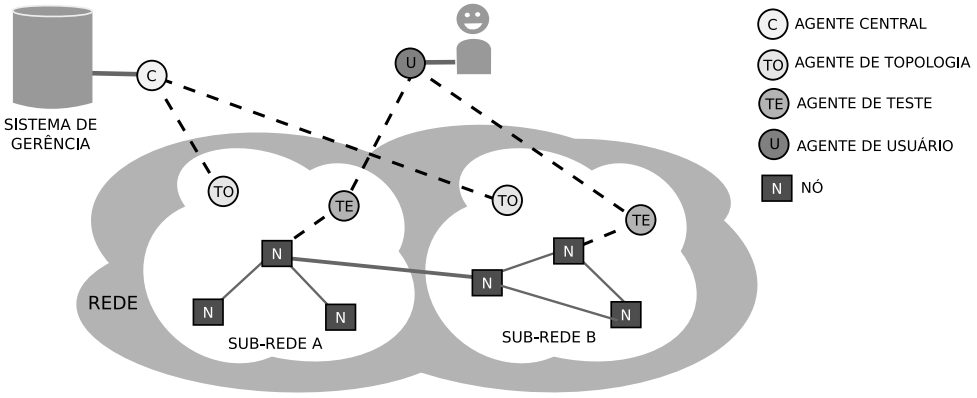

Figura 3.2: Distribuição dos agentes pela rede

Um outro aspecto central da solução proposta diz respeito aos testes que serão feitos nos equipamentos e circuitos da rede. No contexto deste trabalho, um *teste* pode ser entendido como um fluxo de tarefas automatizadas necessárias para a obtenção de um determinado diagnóstico ou correção de falha. Estes testes usualmente coletam dados de equipamentos, processam os mesmos, e fornecem um laudo que identifica uma eventual falha. Naturalmente, os testes que necessitarem de informações de topologia ou inventário da rede para identificar alguma falha, devem consultar o agente de topologia correspondente e extrair dele os dados desejados. Também é possível que um teste aja sobre um equipamento de rede a fim de efetuar uma correção e não apenas uma coleta de dados.

Em alguns casos, pode ser interessante ou até mesmo necessário que um teste consulte um sistema de gerência para buscar alguma informação não relacionada à topologia ou inventário da rede. Apesar de esta comunicação direta entre um agente de teste e o sistema de gerência ser possível, tal estratégia de comunicação direta é desaconselhada, pois pode fragilizar a solução em termos de sua escalabilidade. Contornar este problema no caso de informações de topologia foi, na verdade, uma das maiores motivações para o presente trabalho. Uma melhor solução seria, se possível, adaptar os agentes de topologia para trabalhar com este novo tipo de dado.

Um exemplo simples de teste seria verificar se uma determinada porta está com uma taxa de perda de pacotes acima de um limite aceitável. Neste caso, o teste deve conter tarefas para buscar no equipamento o número de pacotes perdidos na porta e para calcular a taxa em questão. Os parâmetros de entrada deste teste seriam a porta a ser verificada e o limite de tolerância de pacotes perdidos. O resultado do teste seria uma simples mensagem indicando se há um excesso de pacotes perdidos.

Um exemplo mais complexo de teste, que faz uso de informações de topologia, seria verificar um por um os recursos de rede que um determinado

circuito de um usuário da rede utiliza. Este cenário é comum em redes ATM, Frame Relay e TDM em que os circuitos utilizados por cada usuário podem ser determinados em um momento inicial e podem permanacer inalterados indefinidamente. Nestes casos, um teste interessante é percorrer cada recurso que compõe este circuito verificando a existência de anormalidades que porventura possam prejudicar a qualidade do serviço prestado ao usuário em questão. Outro exemplo interessante de teste em que informações de topologia de rede se fazem necessárias é o teste de fonte de sincronismo de nós. Este teste é discutido em detalhes no Capítulo 4.

As tarefas que compõem um teste e que se comunicam com equipamentos podem ser classificadas como *sensores* ou *comandos*. Um sensor é uma tarefa que busca de um equipamento alguma informação relativa ao estado do mesmo. Um comando é uma tarefa que executa alguma ação sobre um equipamento, possivelmente alterando suas configurações. Tanto sensores quanto comandos atuam como mecanismos de comunicação entre os agentes de teste e os equipamentos. A diferença entre ambos é que um comando é, por definição, intrusivo e pode mudar o comportamento de um equipamento. Um sensor, por sua vez, apenas lê dados do equipamento sem maiores efeitos colaterais. Tal distinção é importante porque permite identificar os testes intrusivos (que são potencialmente mais perigosos que os testes que apenas possuem sensores). Outro aspecto relacionado é que um teste que realiza uma correção possui necessariamente pelo menos um comando.

Sensores e comandos devem realizar a comunicação com os equipamentos usando protocolos de gerência de rede tais como SNMP [RFC3411], TL1 [GR-831], Netconf [RFC4741] e JUNOScript [Juniper]. Uma outra forma de acesso comum é a CLI (*Command Line Interface*), normalmente conduzida através dos protocolos Telnet [RFC854] ou, preferencialmente, SSH [RFC4251].

Para garantir uma maior flexibilidade à arquitetura proposta, buscou-se manter um baixo acoplamento entre o agente de teste e os tipos de teste que podem ser aplicados à rede. A implicação prática disto é que o agente de teste carrega dinamicamente os testes a partir de uma base de testes. Uma vantagem desta estratégia é que ela facilita a criação de novos tipos de testes, uma vez que o restante do sistema (em particular o agente de teste) não precisa ser alterado. Outra conseqüência é que também é facilitada a adaptação desta arquitetura para funcionamento em uma outra rede de telecomunicações, uma vez que um novo conjunto de testes direcionados a outra rede pode ser adaptado à arquitetura aqui proposta.

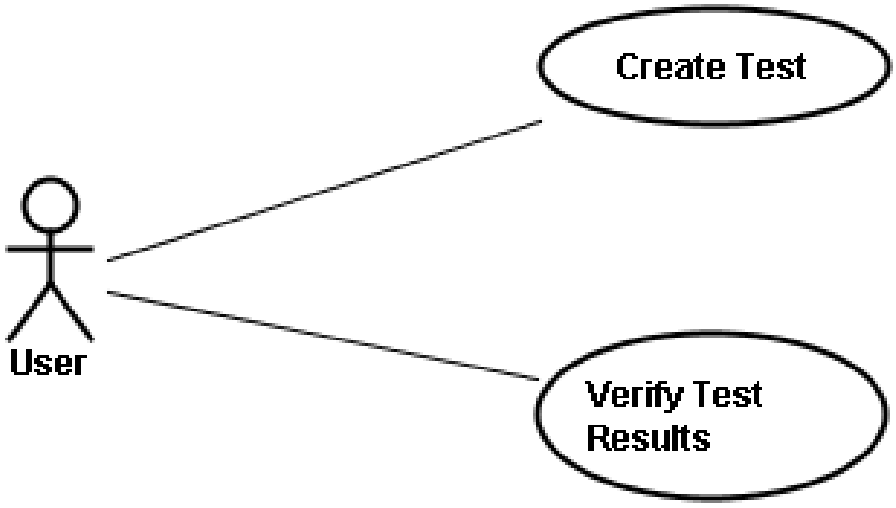

Figura 3.3: Diagrama de casos de uso

3.4 Arquitetura

3.4.1 Requisitos Funcionais

Sob a ótica do usuário final, geralmente um operador de rede, a funcionalidade do sistema proposto é bastante simples. O que ele se propõe a fazer na prática é possibilitar a execução de testes na rede e, depois de concluídos, exibir os resultados dos mesmos. Nos casos em que se deseje um monitoramento ao longo de um intervalo de tempo, o usuário deve fornecer valores para os parâmetros que indicam a freqüência e periodicidade desejadas para o teste em questão. De fato, a simplicidade desta interface pode ser encarada como um ponto positivo já que a interação do usuário com o sistema é facilitada. O que este sistema tem de especial é a forma como estes testes são executados, que não fica evidente para o usuário.

O diagrama de casos de uso da Figura 3.3 ilustra estas duas funcionalidades básicas.

3.4.2 Modelagem do Sistema

O sistema é dividido em quatro módulos principais correspondentes aos quatro tipos de agente. Além disto, há dois módulos adicionais com as classes que representam as entidades do sistema (objetos de negócio) e as entidades de testes. Há também o módulo para serialização de algumas destas entidades para o formato XML. Estes módulos são apresentados a seguir.

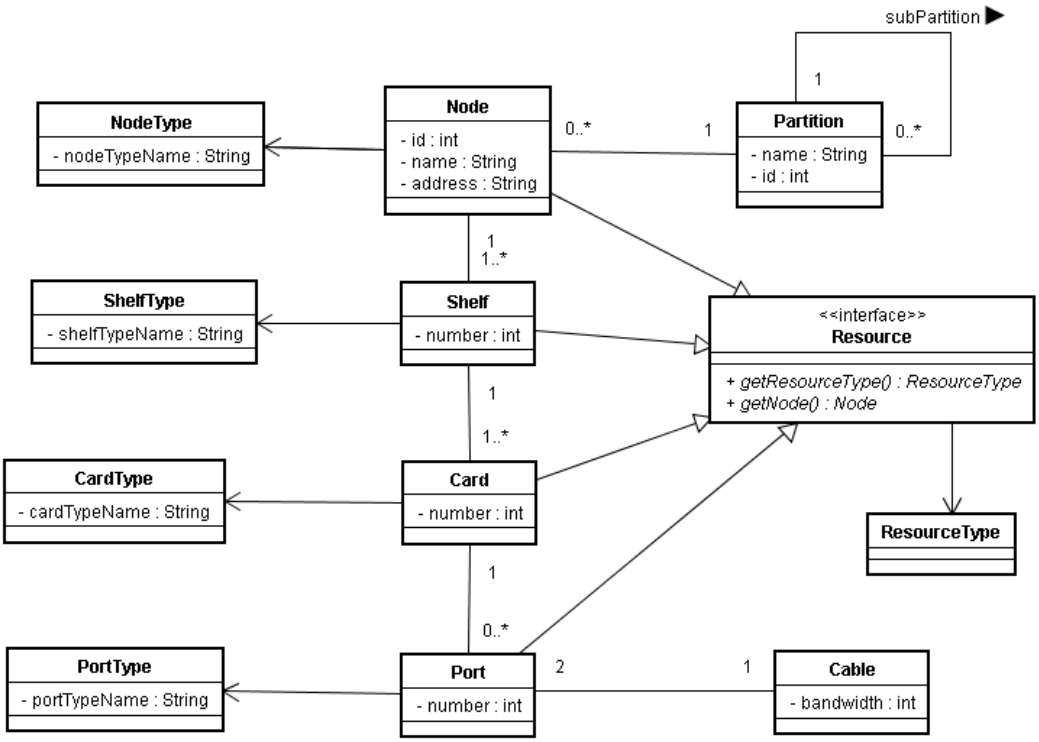

Figura 3.4: Diagrama de classes (simplificado) das entidades básicas do sistema

Módulo de Entidades

O módulo de entidades contém basicamente as classes utilizadas para representar recursos de rede sobre os quais o sistema irá executar os diversos testes. Os recursos de rede modelados são nós, *shelves*, cartões, portas e cabos. Também há classes para modelar os tipos de nós, *shelves*, cartões e portas existentes na rede. Há ainda uma classe para modelar as partições da rede.

As classes básicas podem ser visualizadas na Figura 3.4.

Módulo de Conversão de Entidades

O módulo de conversão de entidades é fortemente relacionado com o módulo de entidades. Nele encontram-se as classes responsáveis pelo mapeamento entre as instâncias de objetos que modelam recursos de rede e documentos XML equivalentes. A serialização de objetos que modelam recursos de rede é necessária para possibilitar a troca de mensagens entre os diversos agentes de software que compõem o sistema.

A classe `AbstractConverter` contém os mecanismos de conversão de objetos Java (gerados pela API JAXB [JAXB]) em XML e vice-versa. Cabe às demais classes implementar as conversões entre estas classes geradas automaticamente e as classes de entidades exibidas na Figura 3.4.

A Figura 3.5 apresenta as classes do módulo de conversão de entidades.

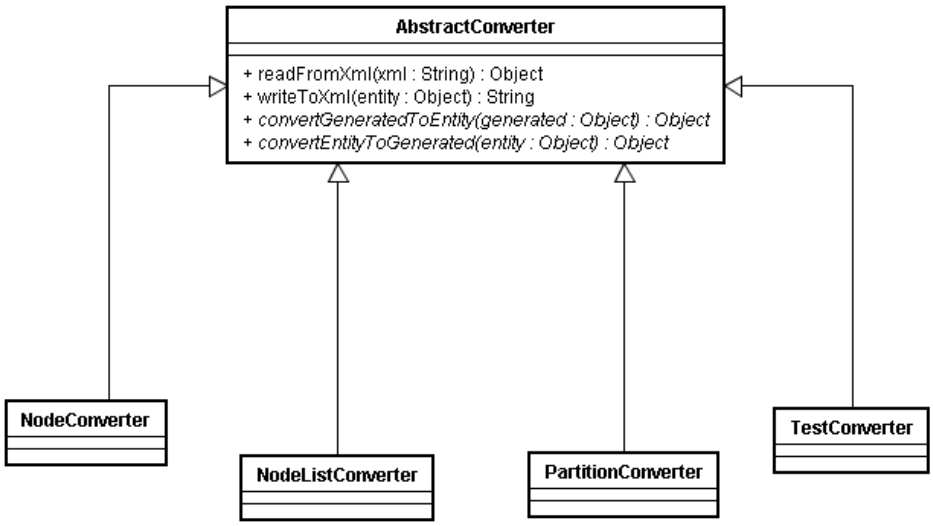

Figura 3.5: Diagrama de classes (simplificado) do módulo de conversão de entidades

Módulo de Testes

O módulo de teste reúne as classes relacionadas às abstrações de testes. A principal classe é uma classe abstrata (`Test`) da qual todos os testes devem herdar. Cada teste possui uma lista de parâmetros associada a si. Os parâmetros são os argumentos de entrada que o teste necessita para poder ser executado. Um parâmetro que virtualmente todos os testes devem ter, por exemplo, é o recurso de rede a que ele se aplica. Esta lista pode ter nenhum, um ou vários parâmetros dependendo do tipo do teste. Cada parâmetro tem um nome, uma descrição e, opcionalmente, um valor padrão.

Cada teste também possui um `TestCommand` associado, que é a classe em que de fato se encontra o método com a lógica para a execução do teste. Esta estratégia é baseada no padrão de projeto Comando [Gamma95], e provê um maior desacoplamento entre os diversos tipos de teste e o agente de teste, uma vez que o agente só precisa conhecer a interface e não as suas diversas implementações. Esta classe deve fornecer um resultado com uma mensagem e um nível de gravidade. Há três níveis de gravidade possíveis: em ordem (ou OK), alerta ou erro.

Na Figura 3.6 podem-se visualizar as classes do módulo de testes. Há também nesta figura duas classes que representam exemplos de testes.

Módulo do Agente Central

Este módulo contém as principais classes relacionadas ao agente central. Neste módulo destacam-se a classe `CentralAgent` que implementa o agente, a classe `ProvideTopologyBehaviour` que implementa o comportamento de

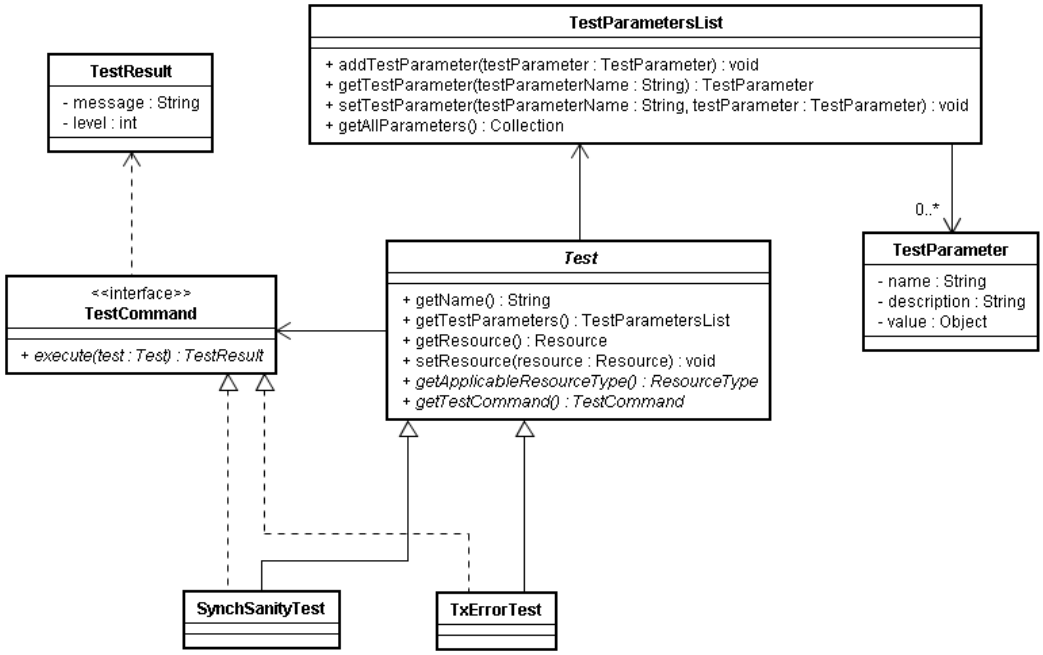

Figura 3.6: Diagrama de classes (simplificado) do módulo de testes

provimento de informações de topologia e as classes de acesso a dados (DAOs [Alur03]) que fazem a interface com o sistema centralizado de gerência.

A Figura 3.7 ilustra o módulo do agente central.

A interface `CentralDao` possui métodos para a extração de dados de topologia do sistema de gerência centralizado. Em última análise, toda e qualquer informação de topologia contida neste sistema de gerência e necessária para a execução de um teste será (indiretamente) obtida através de uma chamada a esta interface.

Módulo do Agente de Topologia

Este módulo contém as principais classes relacionadas ao agente de topologia. Neste módulo destacam-se se a classe `TopologyAgent` que implementa o agente de topologia e a classe `ProvideTopologyBehaviour` que implementa o comportamento de provimento de informações de topologia sobre os nós que gerencia. Este comportamento utiliza uma classe de acesso a dados (DAO) para recuperar as informações de topologia que precisa prover. Há ainda outros dois comportamentos: o `GetNodesFromPartitionBehaviour` que busca do agente central uma lista com os nós sobre os quais este agente é responsável, e o `CreateTestAgentBehaviour` que recebe solicitações de agentes de usuário para a criação de agentes de teste.

A Figura 3.8 ilustra o módulo do agente de topologia.

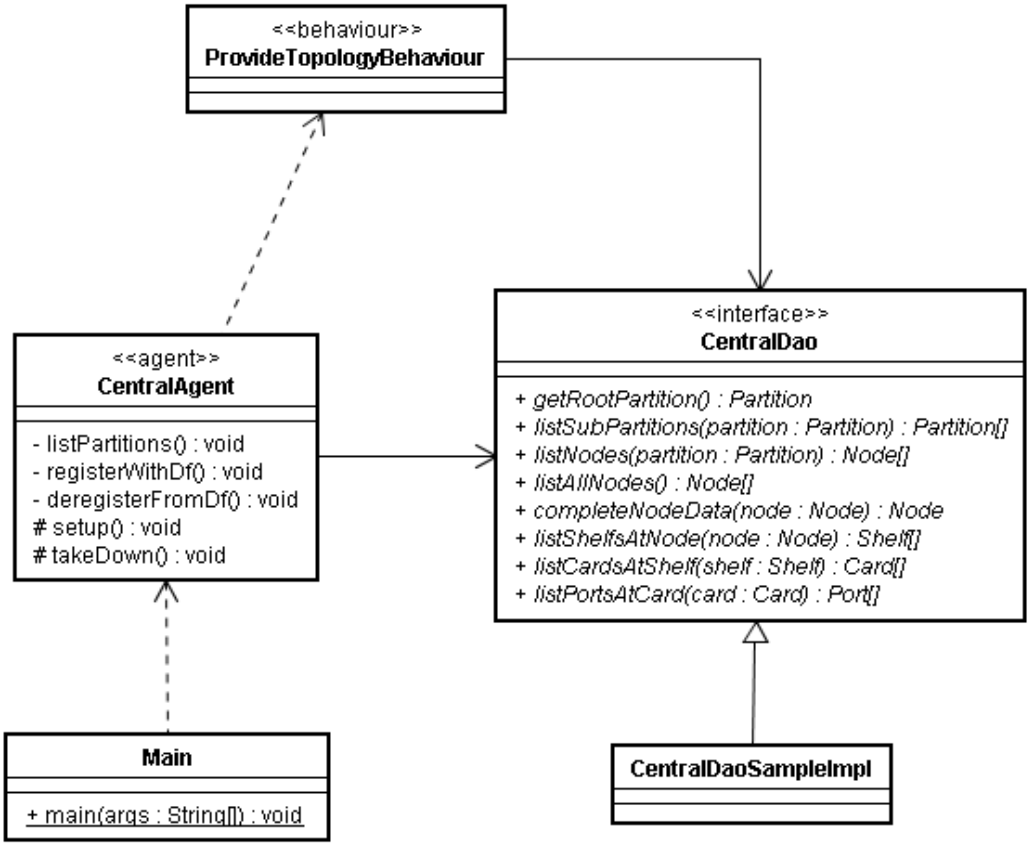

Figura 3.7: Diagrama de classes (simplificado) do módulo do agente central

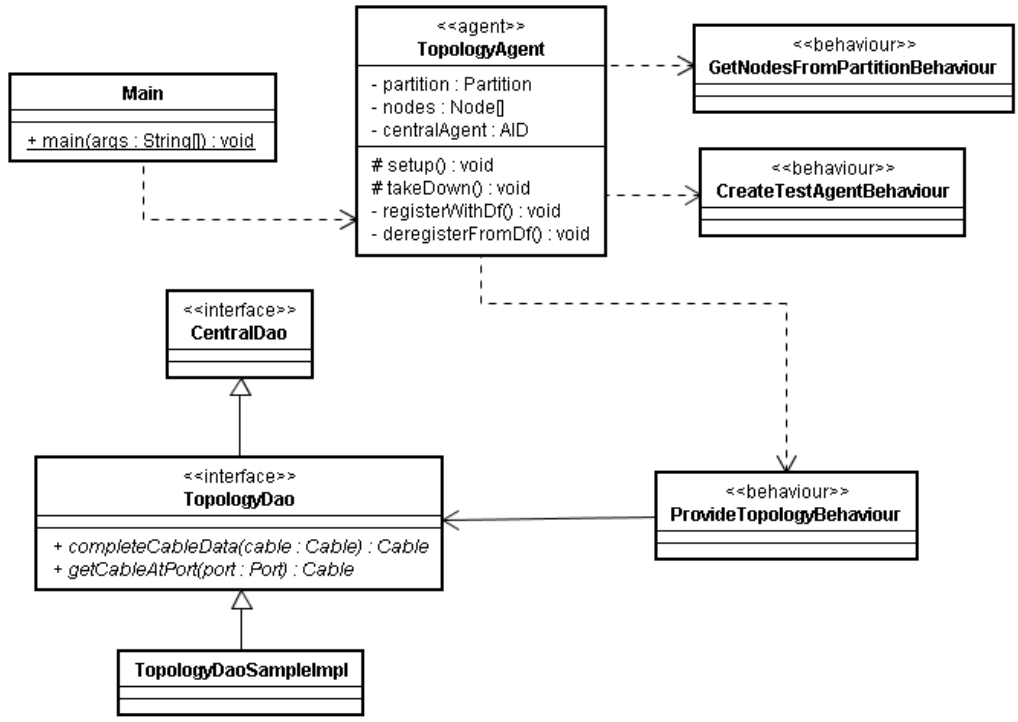

Figura 3.8: Diagrama de classes (simplificado) do módulo do agente de topologia

A interface `TopologyDao` possui métodos para a extração de dados de topologia mantidos pelo agente de topologia em uma base local que contém as informações que este obteve do agente central.

Há muitas semelhanças entre a interface `TopologyDao` exposta pelos agentes de topologia e a interface `CentralDao` exposta pelo agente central, uma vez que ambas são utilizadas para a obtenção de dados de topologia. A principal diferença entre as duas é que elas foram projetadas para atender diferentes clientes. A interface `CentralDao` deve ser utilizada primordialmente por agentes de topologia e a interface `TopologyDao` deve ser utilizada principalmente por agentes de teste. Em virtude disto, a interface `CentralDao` possui serviços mais simples para a recuperação de dados de topologia e inventário. Há duas razões principais para isto. A primeira razão é que não é aconselhável delegar tarefas complexas ao agente central uma vez que ele é o ponto mais crítico da arquitetura. Por isso as tarefas que ele desempenha devem ser tão simples quanto possíveis. A segunda razão é que é interessante fornecer aos agentes de testes serviços que facilitem a implementação dos mesmos no que diz respeito à disponibilização de informações de topologia e inventário de rede. Tais interfaces devem fornecer serviços não triviais como, por exemplo, identificar a porta de um nó remoto conectada (através de um cabo) a uma porta qualquer em questão. Esta relação entre as duas interfaces de topologia é evidenciada pelo fato de que a interface `TopologyDao` herda da interface `CentralDao`.

Módulo do Agente de Teste

Este módulo contém as principais classes relacionadas ao agente de teste. A principal classe deste módulo é a `TestAgent` que implementa o agente de teste. Há três comportamentos desempenhados por este agente: `ReceiveTestRequestBehaviour`, que recebe dos agentes de usuário as solicitações para a execução de testes, `ExecuteTestBehaviour`, que é quem de fato executa os testes, e `AskForTopologyBehaviour`, que é o comportamento responsável por solicitar informações de topologia ao agente de topologia correspondente.

Há ainda neste módulo a classe `TopologyDaoStubImpl`. Esta classe implementa a interface `TopologyDao` e é usada por testes que necessitem de informações de topologia para a sua execução. Esta classe transforma requisições de topologia em um documento (XML) que é utilizado pelo agente de teste para comunicar-se com um agente de topologia e solicitar do mesmo a informação de topologia desejada.

A Figura 3.9 ilustra as classes do módulo do agente de teste.

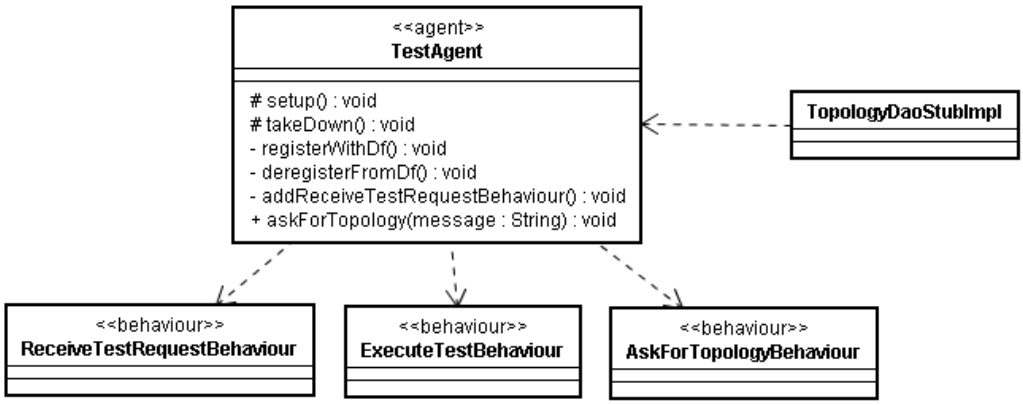

Figura 3.9: Diagrama de classes (simplificado) do módulo do agente de teste

Módulo do Agente de Usuário

Este módulo contém as principais classes relacionadas ao agente de usuário. A classe mais importante neste módulo é a classe `UserAgent` que implementa o agente de usuário. Ela utiliza as classes de comportamento `GetAllNodesBehaviour`, que busca uma lista com todos os nós da rede de um agente de topologia, `AskToCreateTestAgentBehaviour`, que solicita a um agente de topologia a criação de um agente de teste, e `AskToPerformTestBehaviour`, que solicita a um agente de teste a execução de um teste em um determinado recurso da rede. Este módulo ainda conta com algumas classes que representam as janelas da interface gráfica com o usuário.

A Figura 3.10 ilustra as classes do módulo do agente de usuário.

3.4.3 Comportamentos e Interações dos Agentes

Um aspecto interessante acerca da arquitetura do sistema diz respeito à dinâmica do mesmo, em particular aos comportamentos dos agentes de software e à interação entre diversos agentes. No contexto desta discussão, um *comportamento* de um agente representa uma tarefa que este deve desempenhar. Um comportamento pode ser uma atividade perene que deva ser executada indefinidamente (enquanto o agente permanecer ativo), como também pode ser uma atividade efêmera que deva ser executada e concluída rapidamente. Um agente ativo deve possuir pelo menos um comportamento, sendo também possível que ele possua mais de um simultaneamente[1].

A seguir são apresentados os principais comportamentos dos agentes do sistema proposto.

[1]Este conceito de comportamento é mapeado diretamente na classe Behaviour da plataforma JADE [JADE] usada neste projeto.

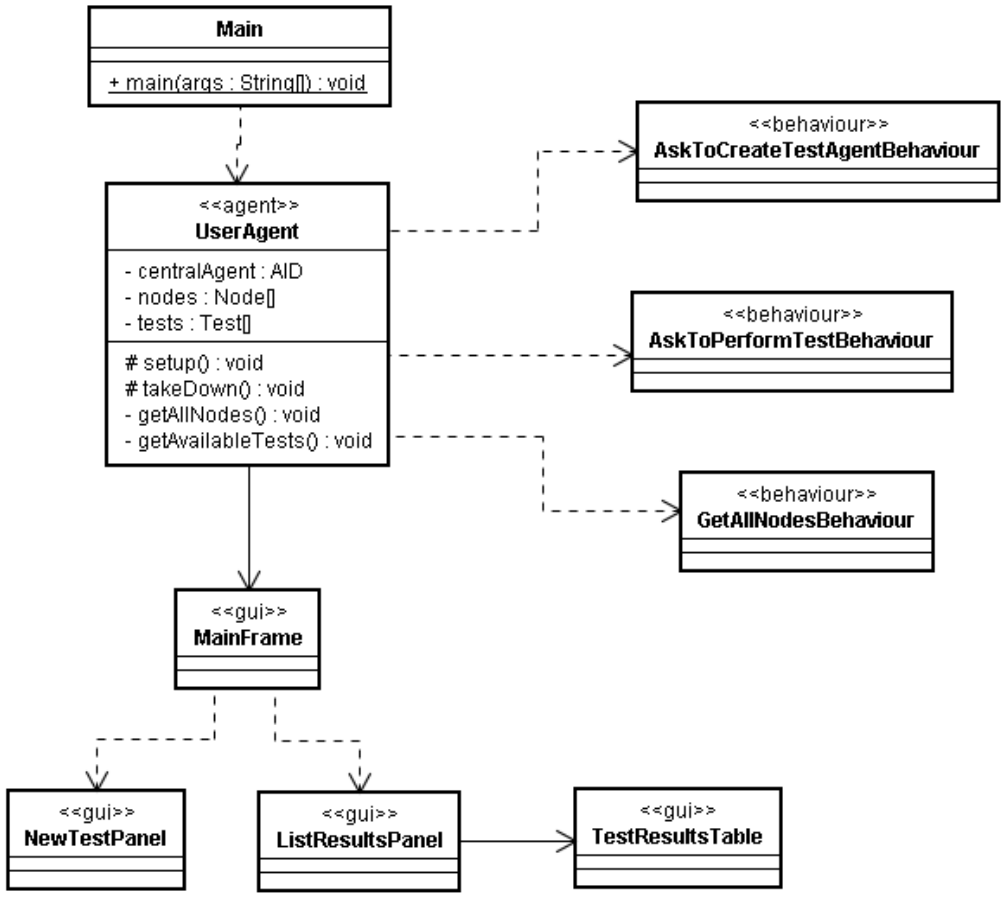

Figura 3.10: Diagrama de classes (simplificado) do módulo do agente de usuário

Agente Central

A principal atribuição do agente central é prover informações de topologia da rede de telecomunicações aos agentes de topologia. Logo, quando este é criado, ele cria e configura o comportamento que provê estas informações aos demais agentes. Em seguida, ele registra seus serviços no agente DF (*Directory Facilitator*), que provê um serviço de "páginas amarelas". O agente DF é definido no padrão [FIPA00023] e permite que um agente descubra outros agentes que ofereçam serviços necessários para que ele alcance algum de seus objetivos.

A Figura 3.11 ilustra o processo de criação do agente central.

Uma vez criado, o agente central se encontra pronto para responder a solicitações de informações de topologia. A principal função do comportamento `ProvideTopologyBehaviour` é prover acesso à interface `CentralDao`. Para que isto ocorra este comportamento aceita mensagens provenientes de agentes de topologia. Estas mensagens devem conter uma indicação de que tipo de informação desejam e do recurso de rede em questão (se for o caso). Estas indicações e os respectivos recursos são diretamente mapeados para métodos e

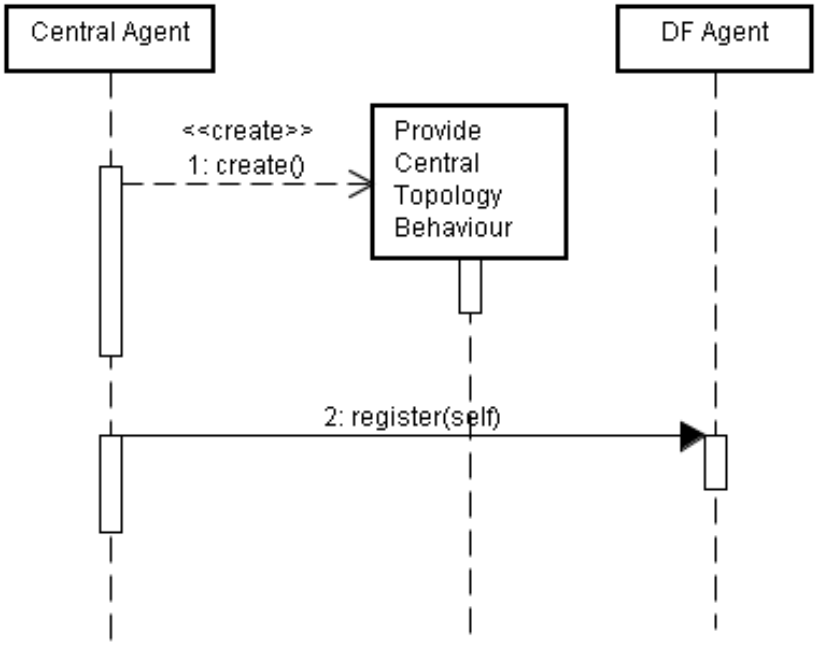

Figura 3.11: Diagrama de seqüência da criação do agente central

parâmetros da interface `CentralDao`. Caso não haja nenhum problema, o agente central enviará uma mensagem de resposta ao agente de topologia contendo as informações solicitadas.

Agente de Topologia

A principal atribuição do agente de topologia é prover informações de topologia aos demais agentes do sistema. Ele deve executar mecanismos para se manter atualizado com a topologia obtida do agente central. Além disso, o agente de topologia é o responsável pela criação de agentes de teste vinculados a equipamentos localizados na partição pela qual é responsável.

A primeira ação do agente de topologia quando de sua criação é buscar no agente DF a identificação do agente central. De posse desta informação o agente de topologia requisita ao agente central uma lista com os nós em sua partição da rede. Em seguida, o agente cria e configura o comportamento de prover informações de topologia para outros agentes do sistema e também o de criar agentes de teste quando a execução de um teste em sua partição for solicitada. Por fim o agente de topologia registra seus serviços no DF para que ele possa ser posteriormente contatado.

O processo de criação do agente de topologia é ilustrado na Figura 3.12.

Uma vez criado, o agente de topologia se encontra pronto para responder a solicitações de informações de topologia de nós pertencentes à partição sobre a qual é responsável. A principal função do comportamento

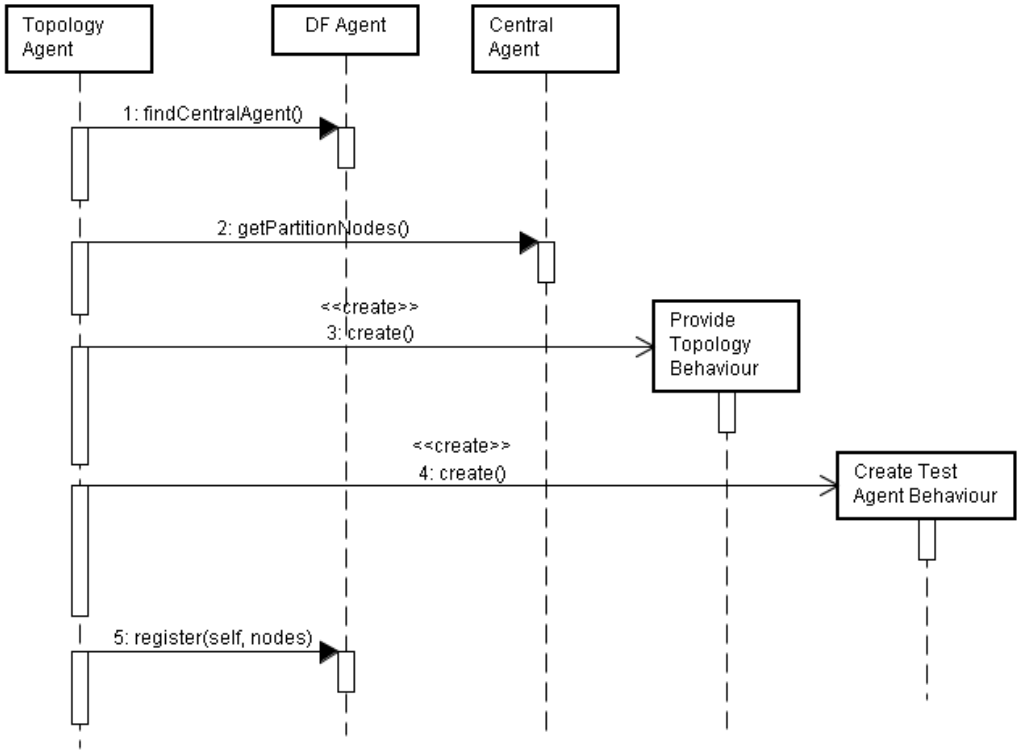

Figura 3.12: Diagrama de seqüência da criação de um agente de topologia

`ProvideTopologyBehaviour` é prover acesso à interface `TopologyDao`. Este comportamento é bastante semelhante ao comportamento que permite a troca de informações entre o agente central e agentes de topologia. Para que ocorra a troca de informações entre agentes de topologia e agentes de teste, este comportamento aceita mensagens provenientes destes agentes. Estas mensagens devem conter uma indicação de que tipo de informação desejam e do recurso de rede em questão (se for o caso). Estas indicações e os respectivos recursos são diretamente mapeados para métodos e parâmetros da interface `TopologyDao`. Caso não haja nenhum problema, o agente de topologia enviará uma mensagem de resposta ao agente de teste contendo as informações solicitadas. Para fornecer a resposta, o agente de topologia pode consultar a base que mantém localmente ou consultar outro agente de topologia ou mesmo o agente central.

Outro importante comportamento do agente de topologia é o que cria agentes de testes. Este comportamento espera por requisições de outros agentes, em particular os agentes de usuário, para a criação de um agente de teste. Este novo agente de teste a ser criado será necessariamente responsável por um nó que pertença à partição controlada pelo agente de topologia em questão.

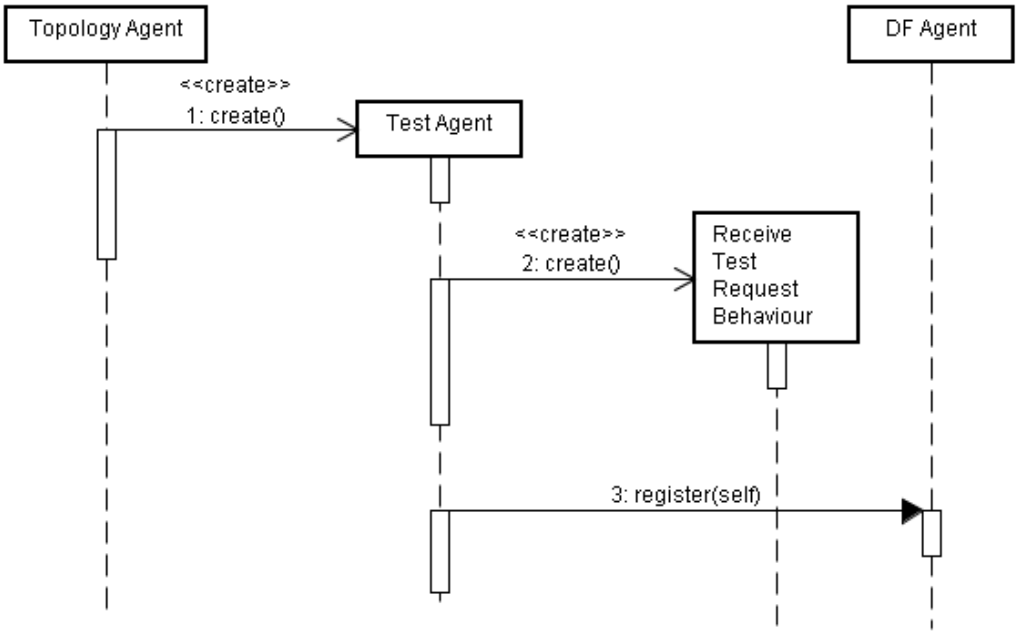

Figura 3.13: Diagrama de seqüência da criação de um agente de teste

Agente de Teste

A principal atribuição do agente de teste é realizar testes nos equipamentos. Um tipo comum de teste é a captura de estatísticas de uso do equipamento e a comparação com algum valor limite ou com uma série histórica. Para disparar a execução do teste ele invoca o método `execute` (da interface `TestCommand`) correspondente ao teste em questão.

Na arquitetura NeMaSA, os agentes de teste são criados por um agente de topologia quando um teste em um equipamento é solicitado e não há nenhum agente de teste responsável pelo mesmo. Para simplificar o projeto foi estabelecido que cada agente de teste fosse responsável por apenas um nó. Isto facilita o estabelecimento de uma hierarquia entre um agente de topologia e os agentes de testes responsáveis por nós pertencentes à partição controlada pelo agente de topologia. Esta hierarquia é interessante, pois permite que o agente de topologia efetue ações sobre agentes de testes em casos de mudanças na topologia da rede.

Quando um agente de topologia cria um agente de teste, este cria e adiciona um comportamento para receber as solicitações de execução de testes e registra seu serviço junto ao agente DF.

Este processo pode ver visualizado na Figura 3.13.

Uma vez criado o agente de teste, este fica responsável pela execução de testes relativos a um nó. Para que a execução de um teste seja solicitada a este agente, o comportamento `ReceiveTestRequestBehaviour` é utilizado. A função básica deste comportamento é receber mensagens com requisições de execução

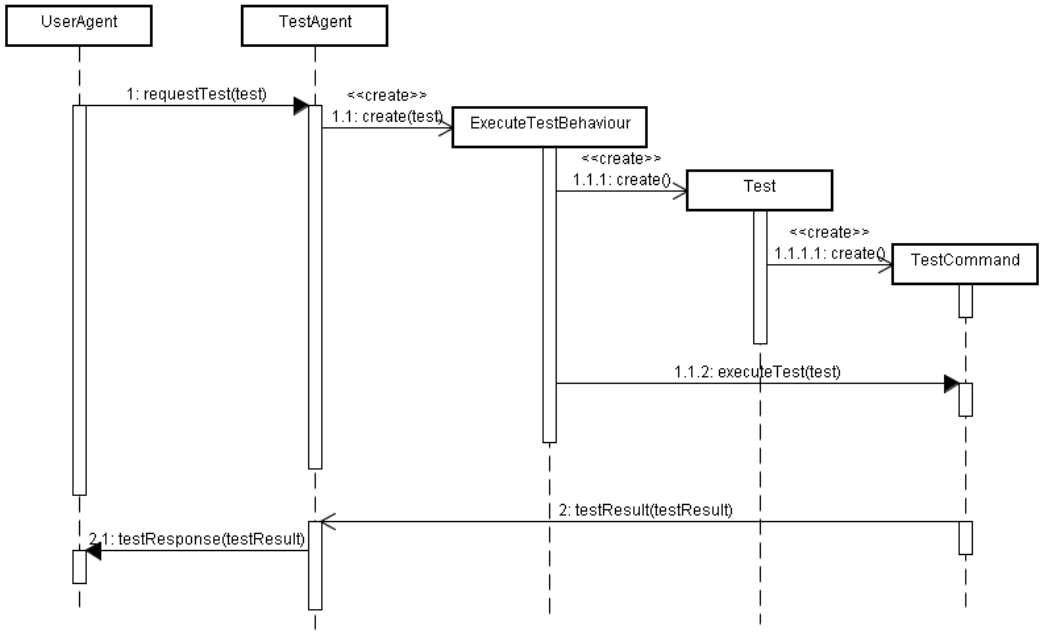

Figura 3.14: Processo de solicitação e execução de um teste

de testes de outros agentes (particularmente de agentes de usuários) e disparar o comportamento de execução do teste. Estas mensagens contêm a indicação de qual teste deve ser executado além dos parâmetros necessários para a execução do mesmo.

Este processo pode ser visualizado na Figura 3.14.

O mais importante comportamento do agente de teste é o comportamento que de fato executa o teste. Apesar de importante, este comportamento é bem simples, uma vez que a complexidade da execução do teste encontra-se encapsulada nas classes de testes. Uma vez encerrado o teste, o agente envia uma mensagem com os resultados do mesmo ao agente que solicitou a execução do teste. Este processo pode ser visualizado na Figura 3.14.

Um aspecto interessante, porém, é que o teste em execução pode eventualmente necessitar de alguma informação de topologia. Para que esta informação seja obtida do agente de topologia correspondente, o teste invoca o método da classe `TopologyDaoStubImpl` correspondente. Esta classe implementa a interface `TopologyDao` e dispara a comunicação entre o agente de teste e um agente de topologia de forma transparente para o teste em execução. Nesta classe a requisição de topologia é convertida em um documento XML que é passado para um método da classe do agente de teste. Neste método, o comportamento `AskForTopologyBehaviour` é criado e executado. Na execução deste comportamento, o agente de teste envia uma mensagem com esta requisição de informação de topologia ao agente de topologia que o criou. O agente de topologia obtém esta informação (de sua base local ou de algum outro agente) e

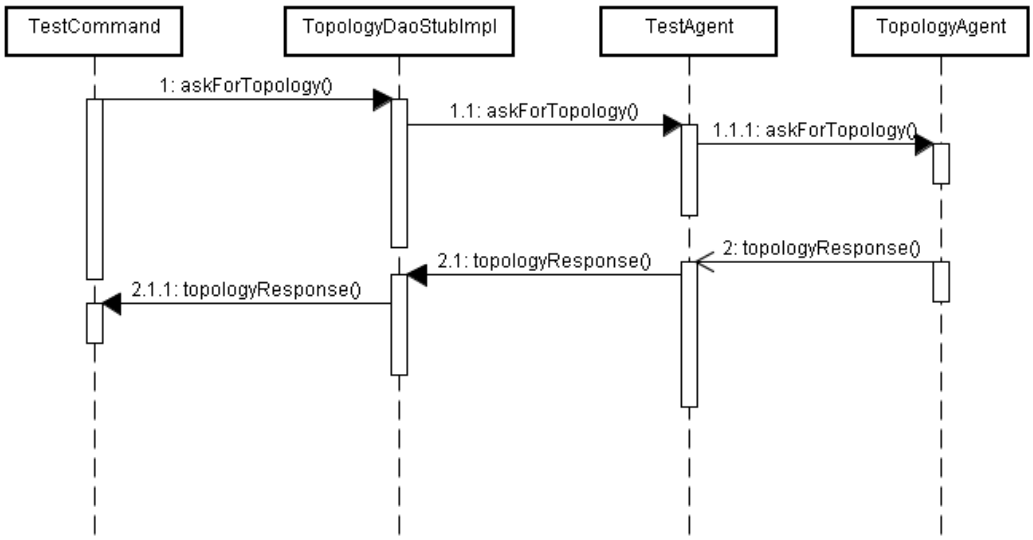

Figura 3.15: Processo de requisição de informações de topologia

envia a resposta ao agente de teste. Por fim, a informação de topologia é passada ao teste que a solicitou inicialmente, e este pode prosseguir.

A Figura 3.15 ilustra este processo.

Agente de Usuário

A principal atribuição do agente de usuário é desempenhar o papel de interface entre um usuário, geralmente um operador de rede, e o sistema. Para tanto, o agente de usuário cria e exibe janelas gráficas (GUI) para que o usuário entre com comandos ou diretivas a serem executadas pelo sistema. Esta interface gráfica também permite que o usuário visualize o resultado das ações do sistema na rede.

Quando um agente de usuário é criado, ele busca um agente de topologia junto ao agente DF para poder obter um inventário simples com todos os equipamentos da rede. Em seguida ele busca uma lista dos testes que o sistema é capaz de executar.

Uma representação deste processo é exibida na Figura 3.16. Estas informações são necessárias para a construção da interface gráfica apresentada na Figura 3.17.

Um dos eventos mais importantes do sistema é a solicitação da execução de um teste por parte do usuário. Quando isto ocorre, o agente de usuário busca junto ao agente DF o agente de teste responsável pelo nó escolhido. Caso não exista um agente de teste que possa atender esta demanda, o agente de topologia criará um novo agente de teste de acordo com o processo representado na Figura 3.13. Com o agente de teste já definido, o agente de usuário solicita a ele a execução do teste, que é conduzida através de um novo comportamento configurado pelo agente de teste.

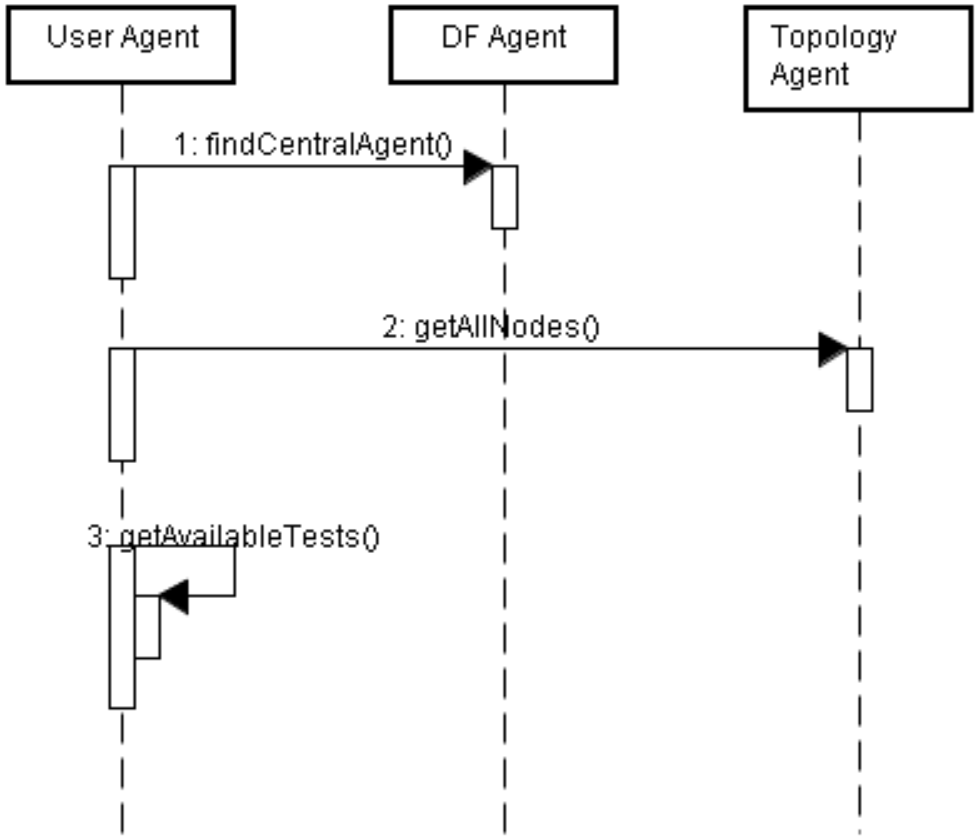

Figura 3.16: Diagrama de seqüência da criação do agente de usuário

Quando a execução do teste se encerra, o agente de teste envia ao agente de usuário o resultado do teste.

Este processo pode ser observado na Figura 3.18.

3.5 Aspectos de Implementação

O sistema é todo escrito em linguagem Java [Gosling05] e usa o *framework* JADE [JADE] como base para a implementação dos agentes de software. A troca de mensagens entre os agentes é feita através da linguagem ACL [FIPA00061]. Sempre que apropriado, as trocas de mensagens entre os agentes seguem os padrões de interação da FIPA, em particular o de consulta [FIPA00027] e o de requisição [FIPA00026].

Em muitas das mensagens trocadas entre os agentes, é necessária a transferência de entidades que representam recursos de rede, testes, resultados de testes, entre outros. No sistema estas entidades são modeladas como objetos Java, como foi discutido anteriormente. Nestes casos, estes objetos são serializados em formato XML, segundo o padrão JAXB [JAXB]. Para tanto é utilizada a biblioteca JaxMe 2 [JaxMe] da *Apache Software Foundation*. Esta estratégia tem a

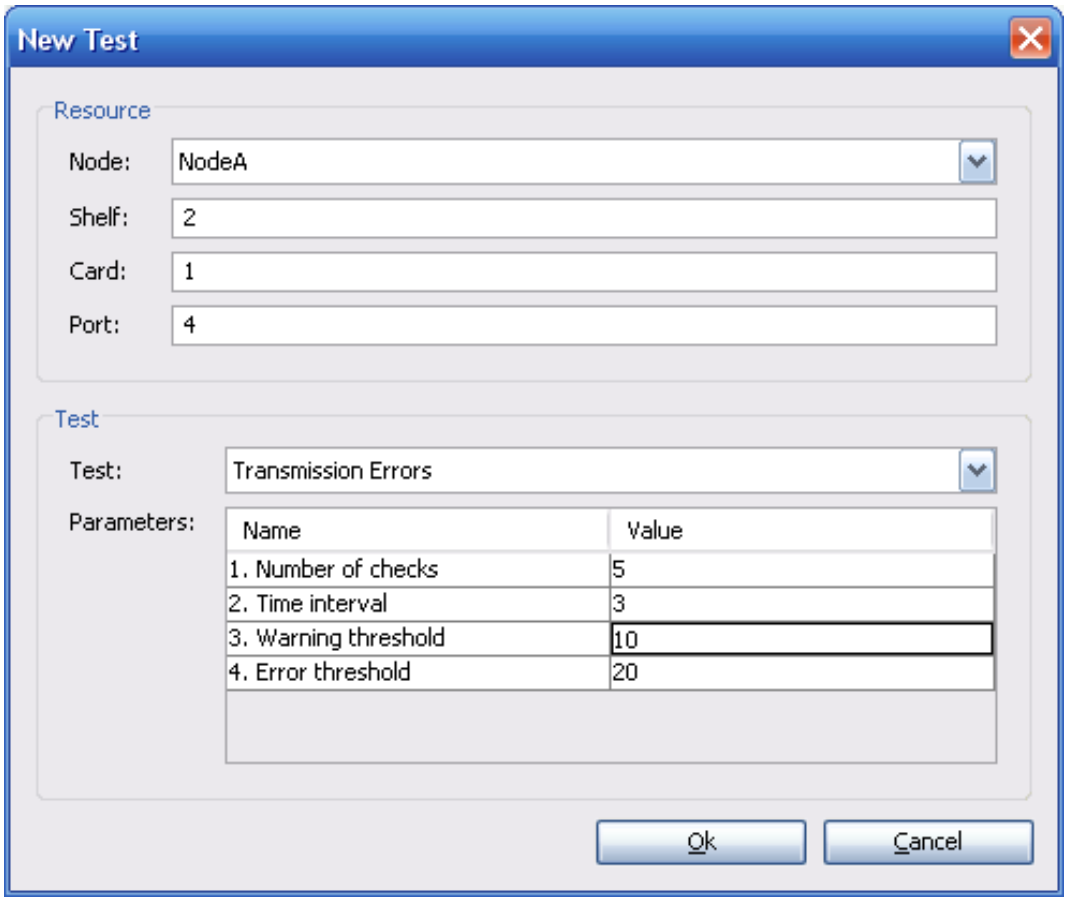

Figura 3.17: Janela para criação de um novo teste

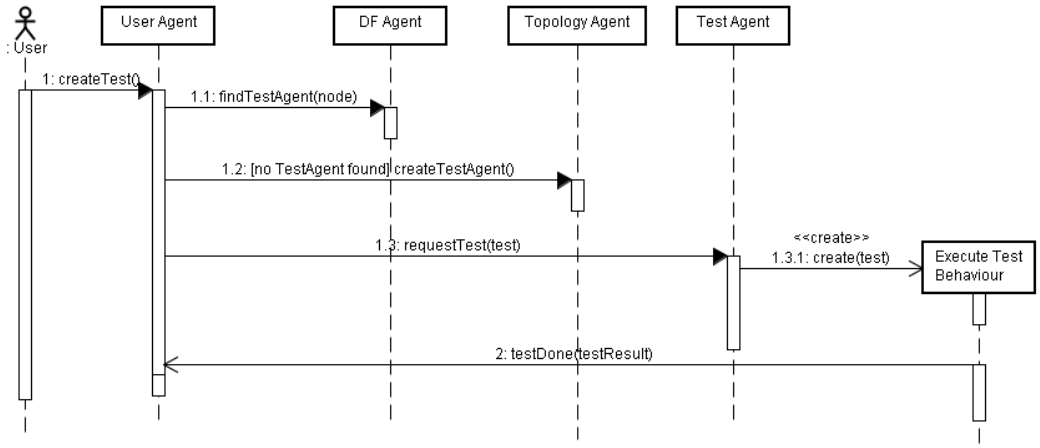

Figura 3.18: Diagrama de seqüência da criação de um teste

vantagem de não utilizar dados binários nas mensagens entre agentes, o que facilita a interoperabilidade com outros eventuais sistemas multi-agentes.

O sistema também utiliza o padrão IoC [Fowler04] para a configuração declarativa (e não programática) de objetos, baseado no *framework* Spring [Spring]. O uso deste padrão traz como vantagem um menor acoplamento entre os objetos do sistema. Este mecanismo é particularmente interessante para a criação do repositório de testes disponíveis aos agentes de testes. Para que um agente obtenha a referência de um teste qualquer basta que este possua o nome do mesmo e solicite a sua instanciação ao objeto que mantém o contexto do Spring. Todavia, um problema experimentado neste projeto e que limitou o uso desta técnica de IoC foi que os objetos que representam os agentes não podem ser configurados diretamente via Spring, pois eles possuem um ciclo de vida especial e devem ser gerenciados diretamente apenas pelo container de agentes do JADE.