Um Padrão para Gerenciamento de Redes

Calebe de Paula Bianchini [1], Eduardo Santana de Almeida [2], Diogo Sobral Fontes [3], Rossana Maria de Castro Andrade [4]

[1] Departamento de Computação – Universidade Anhembi Morumbi (UAM)
R. Quatá, 203 – São Paulo – SP – Brasil – CEP 04.546-041
Fone/Fax: +55 11 3847.3159

[2] Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Av. Prof. Luis Freire, s/n – Recife – PE – Brasil – CEP 50.740-540 – C.P. 5871

[3] Departamento de Computação – Universidade Paulista (UNIP) – Compus de Assis
R. Myrtes S. Conceição, 301 – C. Nelson Marcondes – Assis – SP – Brasil – CEP 19815-050

[4] Departamento de Computação – Universidade Federal do Ceará (UFC)
Campos do Pici, Bloco 910 – Fortaleza – CE – Brasil – CEP 60455-760

[1] calebe@netsite.com.br, [2] esa2@cin.ufpe.br,
[3] sobral.terra@terra.com.br, [4] rossana@ufc.br

Resumo

A crescente descentralização dos recursos computacionais e a necessidade de gerenciamento de sistemas distribuídos e heterogêneos têm motivado os pesquisadores na construção de ferramentas que auxiliam os administradores de redes, em grande parte das suas tarefas, através do monitoramento automático dos dispositivos. Dentre as técnicas para atingir estes objetivos, destacam-se as que utilizam agentes de software. As tarefas, sinalizadas por eventos e muitas vezes repetitivas, dos diferentes domínios de aplicações, incluindo o das aplicações distribuídas, são delegadas aos agentes de software programados em uma base de conhecimento. Assim, esse trabalho apresenta um padrão de arquitetura para o desenvolvimento de agentes de software inteligentes, em um sistema multi-agentes, que auxiliam o gerenciamento de redes. Esse padrão é denominado GeR (Gerenciamento de Redes).

Abstract

The increasing decentralization of computational resources and the need of distributed and heterogeneous systems management are the motivation for researches to construct tools that assist network managers, in great part of their tasks through automatic monitoring devices. Among the techniques to achieve these goals, the ones that use software agents stand out. The tasks, which are signalized by events and many times repetitive, of different application domains, including distributed application, are assigned to the software agents programmed in a knowledge base. Thus, this work presents an architecture pattern called GeR for developing intelligent software agents that assist in the network monitoring.

1 Contexto

A evolução e a expansão das redes de computadores abriram caminho para o desenvolvimento de novas aplicações nesta área da computação.

Além disso, os avanços na tecnologia aplicada a redes de computadores têm enfatizado a investigação de agentes de *software* inteligentes como um paradigma promissor no desenvolvimento de *softwares* complexos e distribuídos. De um modo geral, agentes de *software* são vistos como um conjunto de objetos complexos, com capacidade de tomar decisões diante uma situação. De fato, objetos e agentes apresentam pontos de similaridade, porém o desenvolvimento de *software* orientado a agentes propõe outros desafios para a Engenharia de *Software* a partir do momento em que agentes necessitam de um nível maior e mais complexo de abstração [1]. Já a inteligência de um agente pode variar de acordo com a capacidade de tomar decisões, as formas de representação dessa inteligência (linguagens simbólicas, etc..), a interatividade com outros agentes, as decisões e ações visando um objetivo, entre outras características que definem o grau de inteligência de um agente.

Esses avanços, juntamente com a expansão das redes de computadores, têm provocado grande necessidade de melhorias no gerenciamento de redes para facilitar e agilizar as tarefas dos seus administradores. Para garantir essas novas exigências, é necessário melhorar a qualidade dos serviços, principalmente os relacionados com gerenciamento de redes [2, 3]. A utilização de agentes de *software* inteligentes facilita a construção de modelos dinâmicos, personalizados para melhorar a qualidade dos serviços no monitoramento de redes, prevendo eventos ou aumentando o tempo de resposta aos eventos ocorridos.

2 Motivação

A crescente descentralização dos recursos computacionais e a necessidade de gerenciamento de sistemas distribuídos e heterogêneos têm motivado os pesquisadores na construção de sistemas que auxiliam os administradores de redes, em grande parte das suas tarefas, através do monitoramento automático dos dispositivos.

Essa melhoria tem sido alcançada com o uso de agentes de *software* inteligentes auxiliando a análise dos dados coletados de cada dispositivo. Além dos recursos que o próprio protocolo de gerenciamento oferece (*traps* [3], *Event Mibs* [4], *Expression Mibs* [5], etc), vários trabalhos vêm propondo técnicas e estratégias para o gerenciamento de redes utilizando agentes [6,7,8,9,10].

3 Problema

Apesar das recentes e constantes pesquisas na área de gerenciamento de redes, ainda há carência de técnicas e ferramentas que suportem tanto o desenvolvimento quanto a utilização de agentes de *software* inteligentes em sistemas de gerenciamento.

Em um lado, diversas técnicas têm sido utilizadas pelos pesquisadores na busca de soluções para os problemas do gerenciamento de redes [11, 3], dada a variedade de plataformas de *hardware* e *software* utilizada, envolvendo diferentes dispositivos conectados, e requerendo cada vez mais dedicação e atenção dos administradores para um bom gerenciamento [2,3]. Assim, existe uma grande preocupação em atingir os serviços de gerenciamento [11,3], procurando automatizar grande parte das tarefas dos administradores.

Já no outro lado, pesquisas envolvendo o estado da arte de agentes de *software* seguem duas diferentes linhas: engenharia de *software* orientado a agentes [12], e engenharia de *software* orientado a objetos para sistemas multi-agentes [13]. Na primeira linha, pesquisadores afirmam veementemente que sistemas multi-agentes são muito mais complexos que os orientados a objetos, sendo que o desenvolvimento tradicional não consegue capturar a complexidade de sistemas multi-agentes. Por outro lado, na segunda linha, pesquisadores

propõem a integração de agentes na orientação a objetos, além de defenderem que agentes e objetos são abstrações complementares.

4 Forças

- É necessário minimizar esforços no desenvolvimento dos sistemas de gerenciamento e dos agentes de *software* inteligentes, concentrando-se nas soluções dos problemas de gerenciamento de redes usando agentes.
- Permitir a integração de diversas formas de representação de inteligência e de definição de comportamento dos agentes em um único sistema de gerenciamento.

5 Solução

O objetivo do GeR é propor um conjunto de componentes para a construção de sistemas de gerenciamento de rede. Desse modo, o Padrão para Gerenciamento de Redes – Padrão GeR – permite:

- integrar diferentes tecnologias, como UML [14,15,16], SNMP [2,3], CORBA [17,18] e Java [19], para a construção de sistemas de gerenciamento de rede [2,3];
- cobrir o ciclo de vida do desenvolvimento de agentes de *software* inteligentes, para serem aplicados em sistemas inteligentes de gerenciamentos, desde a fase de especificação até a fase de implementação, culminando na sua utilização e definição de seu comportamento; e
- utilizar diversas estratégias para conduzir e definir o comportamento do agente, como, por exemplo, linguagem lógica, redes neurais, lógica difusa, filtros de verificação, entre outras técnicas de construção e definição de sua inteligência.

Com esse propósito, GeR oferece dois componentes básicos:

- **device**, que implementa as especificações do protocolo SNMP; e
- **agent**, que analisa as configurações coletadas dos dispositivos.

O componente **device** se comunica com o dispositivo remoto, coletando as configurações descritas pela Mib. O **agent** se conecta ao componente **device** através de suas interfaces, recuperando os valores das configurações. A análise é baseada nesses valores, e seu comportamento é descrito na base de conhecimento através de cláusulas lógicas.

GeR sugere que os gerenciamentos de redes e agentes de *software* sejam definidos e construídos separadamente, mantendo sua independência, mas com interfaces bem definidas para sua relação. Essa modularidade permite a reutilização dos componentes em aplicações de outros domínios, como por exemplo, *e-commerce*, agentes de busca, educação a distância, bioinformática, entre outros, e a utilização de diversas técnicas de construção e definição de sua inteligência. Além disso, também permite que apenas partes do padrão sejam reutilizadas, tanto no desenvolvimento de sistemas inteligentes de gerenciamento de redes, como também no desenvolvimento e definição de agentes de *software* inteligentes.

Na solução do GeR, utilizamos o método Catalysis para o desenvolvimento dos componentes, cobrindo suas fases, desde o levantamento de requisitos até a implementação. Para a utilização dos agentes, utilizou-se a plataforma DSAP, que fornece toda uma infraestrutura para o desenvolvimento, criação, execução e mobilidade desses agentes. Dessa forma, grande parte do esforço que seria gasto na construção do agente é direcionada para a definição de seu comportamento e inteligência.

6 Estrutura

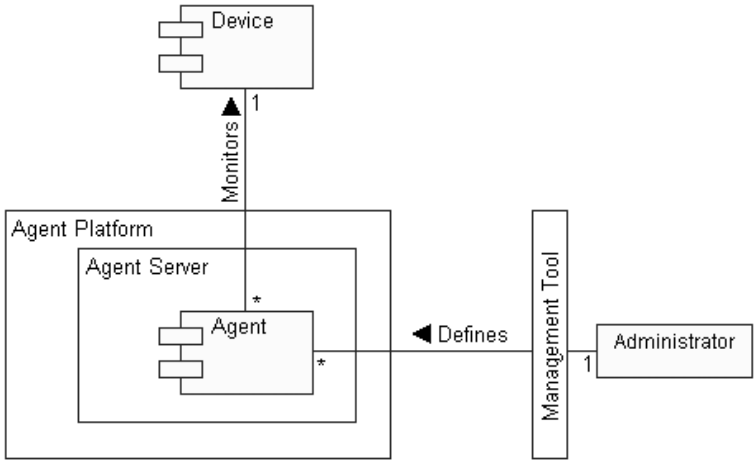

Figura 1. Estrutura do Padrão

7 Participantes

Administrator
- interage com a ferramenta definindo o gerenciamento dos dispositivos, criando agentes de *software* e descrevendo seu comportamento na base de conhecimento.

Management Tool
- oferece recursos para a interação entre o administrador e o sistema de gerenciamento e com os agentes de *software* [6].

Device
- são os dispositivos gerenciáveis, descritos através de suas MIBs, definidos pelo administrador e gerenciados pelos agentes.

Agent
- implementa um conjunto de classes e os seus relacionamentos [20], para a instanciação de agentes inteligentes.

Agent Server
- fornece infra-estrutura e serviços necessários às aplicações distribuídas e multiplataformas em sistemas multi-agentes [21,22]

Agent Platform
- define um ambiente para desenvolvimento de aplicações de diversos domínios e um conjunto escalável de servidores [21,22].

8 Conseqüências

O GeR oferece os seguintes benefícios:
- **Modularidade**: o Padrão permite separar os aspectos de desenvolvimento de agentes e de gerenciamento de redes; e

- **Reutilização**: através dos aspectos de modularidade oferecidos, os desenvolvedores podem reutilizar os componentes, diminuindo a redundância de código.

Mesmo com as vantagens listadas acima, as seguintes desvantagens podem ser apresentadas:

- Conhecimento de outros paradigmas [23,24]: o comportamento dos agentes utilizado no monitoramento é descrito em uma linguagem lógica [25,26,27], obrigando o administrador a conhecer esse paradigma.

9 Implementação

Para a implementação do GeR, utilizou-se as seguintes abordagens:

9.1 Catalysis

Catalysis [14] é um método de desenvolvimento de *software* Orientado a Objetos que utiliza Componentes Distribuídos, Padrões de Projetos e *Frameworks* para projetar e construir Sistemas de Negócio, que devem suportar tecnologias orientadas a objetos, utilizando Java, CORBA ou DCOM (*Distributed Component Object Model*) [28]. Sua notação é baseada na UML [15] e fundamenta-se em três princípios: abstração, precisão e componentes "*plug-in*". O princípio **abstração** orienta o desenvolvedor na busca dos aspectos essenciais do sistema, dispensando detalhes que não são relevantes para o contexto do sistema. O princípio **precisão** tem como objetivo descobrir erros e inconsistências na modelagem e o princípio **componentes "*plug-in*"** usa o reuso de componentes para construir outros componentes [14,16].

Catalysis apresenta modelos precisos e um processo de desenvolvimento completo e sistemático, permitindo aos desenvolvedores partirem da análise e especificação do domínio da aplicação e chegarem ao código, dividindo o sistema em componentes e identificando ao longo do processo os elementos de reutilização [14].

O processo de desenvolvimento de *software* em Catalysis é divido em três níveis: Domínio do Problema, Especificação dos Componentes e Projeto Interno dos Componentes.

O componente *agent* foi desenvolvido utilizando os níveis Catalysis, como se segue:

Domínio do Problema.

Neste nível é dado ênfase no entendimento do problema, isto é, especifica-se "o que" o sistema deve atender para solucionar o problema. Nesta fase, utilizam-se técnicas de entrevista coletiva e informal com o usuário, que são documentados com *Storyboards* [16] e *MindMaps* [16]. Essas representações identificam a ligação entre os termos obtidos na entrevista, definindo assim, uma terminologia do domínio do problema.

Uma vez definida a terminologia do domínio do problema, pode-se criar Modelos de Colaboração [16] e de Casos de Uso [16], que representam os atores e suas interações com o sistema.

A Figura 2 apresenta o Domínio do Problema. Os requisitos identificados foram especificados em Modelos de Colaboração, representando a coleção de ações e os objetos participantes. Em seguida, estes Modelos foram refinados em Modelos de Casos de Uso.

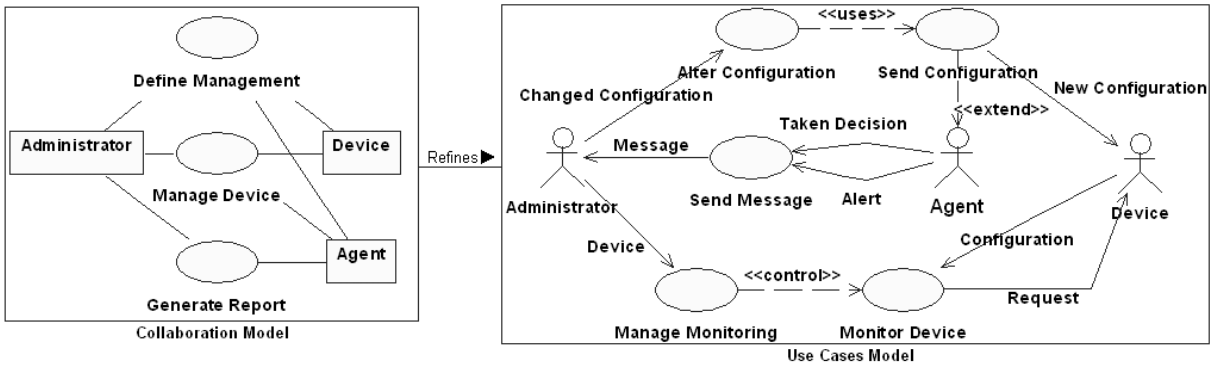

Figura 2. Domínio do Problema

Especificação de Componentes.

Este nível descreve o comportamento externo do sistema de uma forma não ambígua, tendo início com o mapeamento dos diagramas obtidos no Domínio do Problema para o Modelo de Tipos [16] que especifica o comportamento dos objetos. Esse modelo mostra os atributos e as operações dos tipos de objetos, sem se preocupar com a implementação. A partir do Modelo de Tipos constroem-se Diagramas de Seqüência [15] que têm por objetivo mostrar os cenários de execução das operações ao longo do tempo. Outras técnicas, como Diagramas de Estados [15], podem ser utilizadas para a especificação dos componentes. A Figura 3 apresenta o Diagrama de Estados para o componente *agent*.

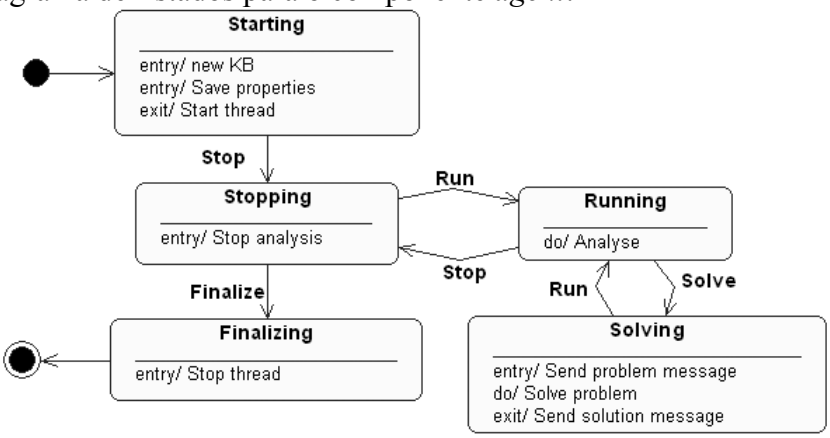

Figura 3. Diagrama de Estados

Projeto Interno de Componentes.

Neste nível define-se "como" serão implementados os requisitos especificados para os componentes, preocupando-se com sua distribuição. O Projeto Interno dos Componentes começa com a definição do Modelo de Classes [15] do Sistema, mostrando as classes com seus atributos, operações e relacionamentos. Esse modelo é derivado do Modelo de Tipos, obtido no nível anterior. Utiliza-se o Modelo de Interação [15] para mostrar a interação entre os objetos. Este modelo é derivado do Diagrama de Seqüência obtido na Especificação dos Componentes. Técnicas para representar a Arquitetura das plataformas Física e Lógica também são usadas neste nível [14]. A Figura 4 mostra as principais classes do componente *agent*.

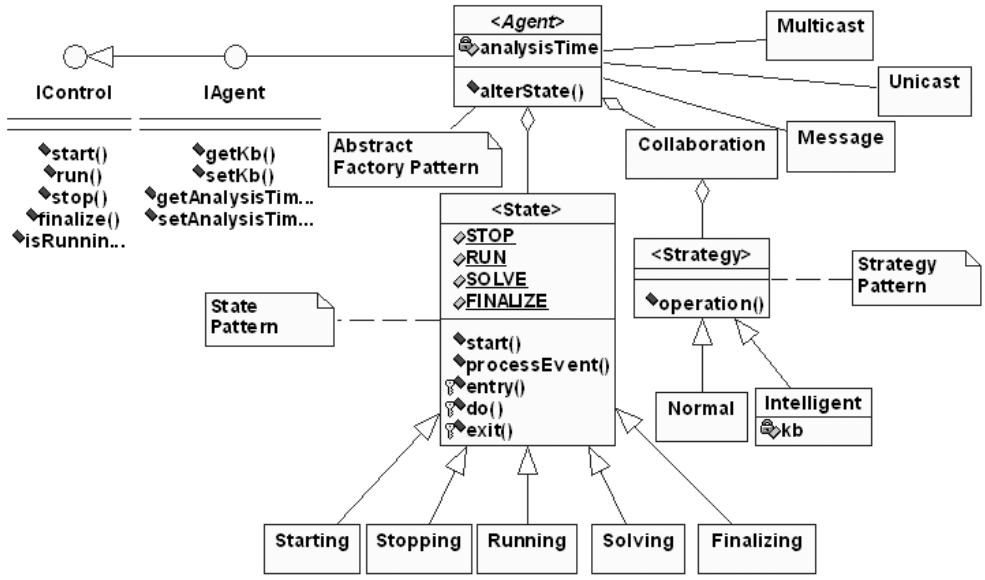

Figura 4. Modelo de Classes

9.2 DSAP – *Distributed Software Agent Platform*

A plataforma DSAP (*Distributed Software Agent Platform*) [21,22] é um ambiente para desenvolvimento de aplicações que utilizam a tecnologia de agentes de *software* em ambiente distribuído, implementada em Java.

Uma das evoluções da plataforma é a utilização de CORBA [17,18], que é um padrão já estabelecido pela OMG (*Object Management Group*) para suportar a distribuição de objetos. CORBA apresenta interfaces bem definidas e independentes de aplicações, através da IDL (*Interface Definition Language*) [17,18] que se encaixa perfeitamente no contexto de Desenvolvimento Baseado em Componentes. Outros aspectos que motivaram o uso de CORBA foram: independência de linguagem de programação, devido a possibilidade do mapeamento da IDL para diversas linguagens; a portabilidade entre ambientes computacionais; e os serviços de Segurança, Nomeação e Notificação, oferecidos pela especificação.

Nessa plataforma multi-agente, é utilizada a engenharia de *software* orientada a objetos, já que as tecnologias de agentes de *software* e orientação a objetos possuem abstrações complementares [13,29]. Assim, pode-se estender técnicas existentes na orientação a objetos, como padrões [30], *frameworks* e componentes [14], e linguagens de modelagem [15,16] para o desenvolvimento de sistemas multi-agentes.

Esta plataforma consiste de um conjunto escalável de servidores utilizados por aplicações de diversos domínios, como por exemplo: *e-commerce*, agentes de busca e educação a distância, criadas a partir da reutilização do *framework* disponível na plataforma. A Figura 5 apresenta a visão geral da plataforma.

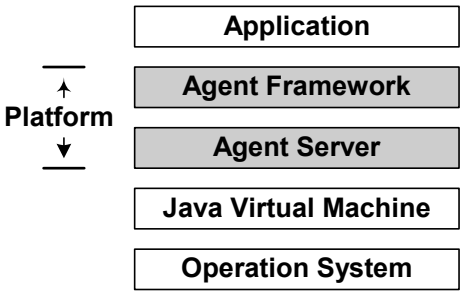

Figura 5. Plataforma DSAP – *Distributed Software Agent Platform*

A camada *Agent Framework* é descrita por um conjunto de classes abstratas e concretas relacionadas, que podem ser estendidas ou instanciadas, necessárias para criação dos agentes, e que irão realizar suas atividades em um ambiente de sistemas distribuídos. A Figura 6 mostra as principais classes do *framework*.

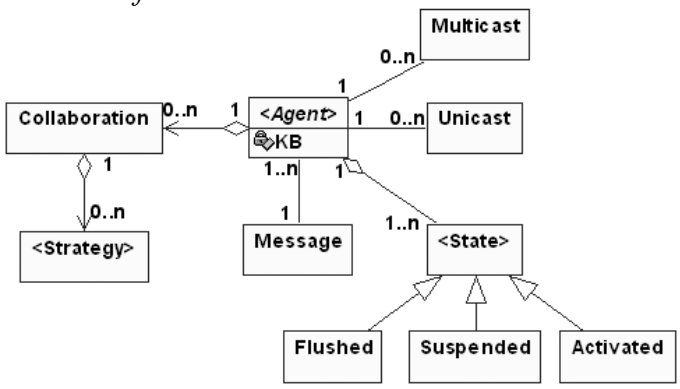

Figura 6. *Agent Framework*

A camada *Agent Server* fornece a infra-estrutura e os serviços necessários às aplicações distribuídas e multiplataformas em sistemas multi-agentes. É responsável por tratar as requisições das aplicações e de outros *Agent Servers*, fornecendo o ambiente de execução para os agentes móveis ou estacionários, conforme mostra a Figura 7.

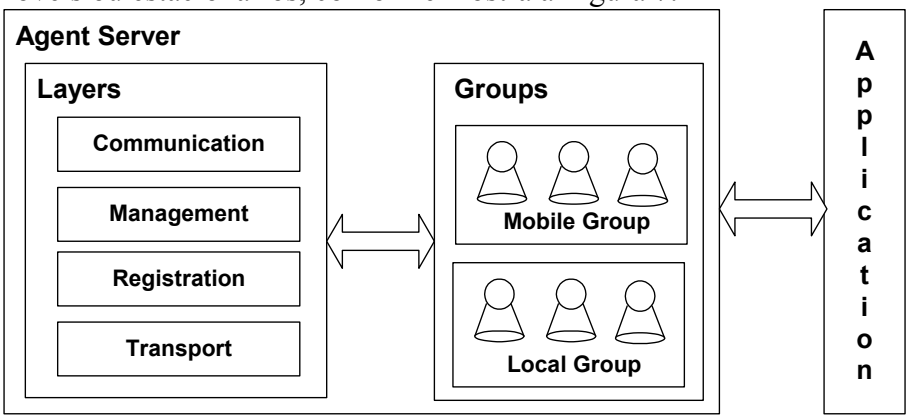

Figura 7. *Agent Server*

O *Agent Server* contém informações dos servidores disponíveis no momento e permite que as aplicações ou os agentes móveis encontrem os servidores distribuídos na rede. O *Agent Server* contém as seguintes camadas:

- *Communication*: responsável pela comunicação remota entre servidores, fornecendo serviços básicos para as aplicações, tais como: localização de servidores, localização de agentes e transporte de agentes;
- *Management*: responsável pela criação e controle dos agentes, fornecendo às aplicações as seguintes funcionalidades: criação, remoção, suspensão e ativação dos agentes. A suspensão e ativação são utilizadas pelos agentes móveis para migrar de um servidor para outro;
- *Register*: responsável por registrar o agente no servidor, tornando-o visível para as aplicações; e
- *Transport*: responsável pelo armazenamento do estado de execução do agente, preparando-o para ser enviado a um outro servidor. Realiza também a recuperação do agente.

10 Usos Conhecidos

A ferramenta MoDPAI [6,31] é um sistema de gerenciamento de redes que utiliza o protocolo SNMP [2,3]. Ela oferece recursos gráficos para auxiliar o administrador na tarefa de monitoramento, permitindo a criação de agentes inteligentes, programados um uma base de conhecimento, que analisam os dados coletados dos dispositivos.

Os agentes foram desenvolvidos utilizando o padrão GeR, construindo uma comunidade de ferramentas que aperfeiçoam o gerenciamento de redes com a troca de agentes entre si.

O comportamento dos agentes é descritos em uma linguagem lógica, e, ao transportar-se de uma ferramenta para outra, o agente leva consigo o código já instanciado em objetos.

A Figura 8 mostra a tela de gerenciamento da MoDPAI, com alguns dispositivos, servidores e *workstations*, sendo monitorados.

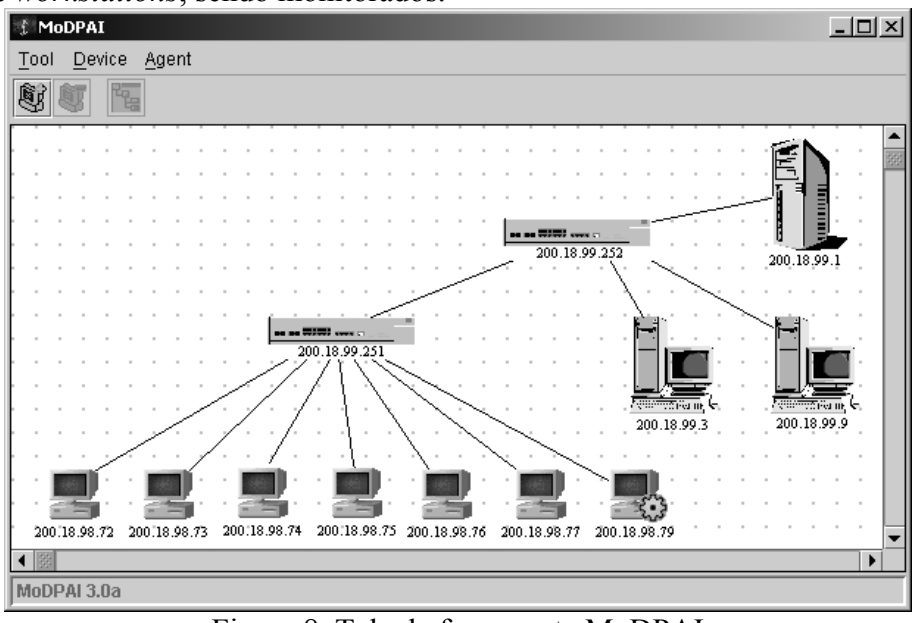

Figura 8. Tela da ferramenta MoDPAI

Para detalhar seu funcionamento em uma rede de computadores, é mostrado um estudo de caso feito no Departamento de Computação da Universidade Federal de São Carlos, onde foi instalada uma comunidade de quatro ferramentas. Foram adicionados todos os servidores (*e-mail*, *webmail*, arquivo, *www*, roteador, DNS) e aproximadamente 200 estações de trabalhos desse departamento. Para cada um desses dispositivos, foi definido um agente de *software* capaz de analisar as configurações coletadas. Em especial, foram analisadas as seguintes perspectivas:

- o número de pacotes TCP [32] e UDP [32] recebidos e enviados da rede interna pelo roteador, viabilizando uma segunda rota de transmissão de dados através de uma nova interface de rede, quando esse número estiver na iminência de sobrecarregar a primeira linha de dados;
- a quantidade de conexões *Web* estabelecidas, relatando seus tempos e verificando o período de maior número de conexões para caracterizá-lo como período crítico;
- o menor e maior período entre duas conexões de *e-mail*, estimando se o serviço de *e-mail*, considerado crítico, não falhou;
- os recursos de memória e disco do servidor de domínio, verificando se este não parou de funcionar; e
- o início e término de uso de cada estação, verificando o volume de informações trocadas pela rede e os recursos de memória e disco disponíveis.

Para o roteador, foram analisadas as entradas da tabela *tcpConnTable*, conforme a Mib mostrada na Figura 9, permitindo alterar as conexões ativas.

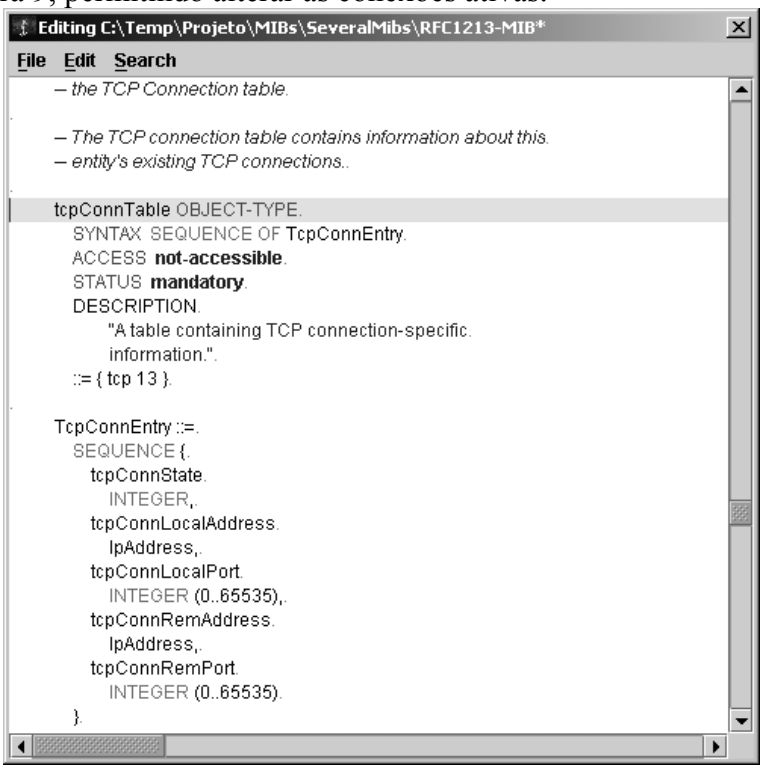

Figura 9. Tabela *tcpConnTable* no arquivo de descrição Mib

A Figura 10 mostra as telas da especificação do um agente de *software* responsável pela verificação dessa tabela.

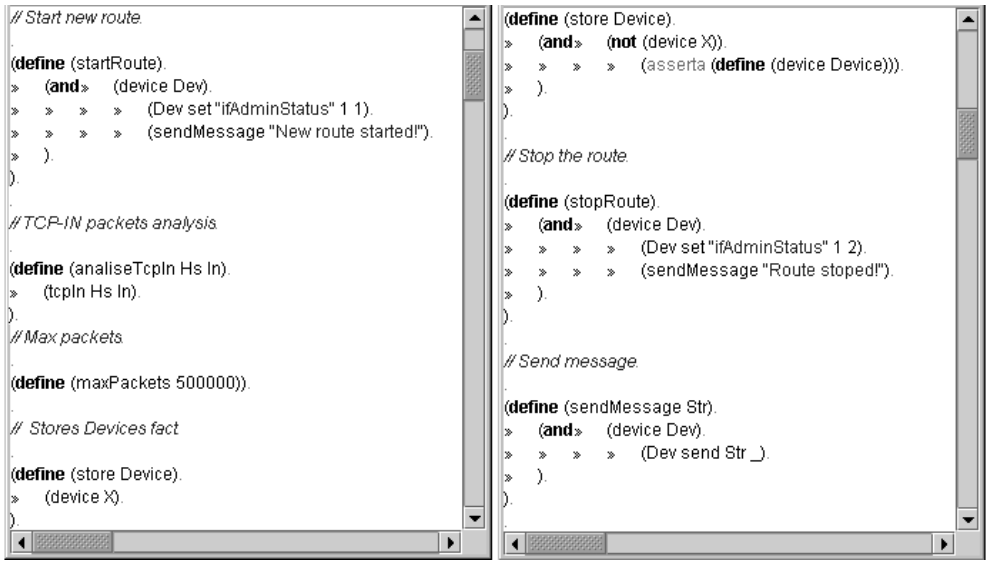

Figura 10. Código KB

Essa figura mostra parte do código KB que define o comportamento do agente, responsável pela análise do número de pacotes que trafegam pelo roteador, habilitando uma nova rota (1), caso haja necessidade. Nesse código, os fatos que indicam o período e a quantidade de pacotes transmitidos são criados ou alterados dinamicamente, mantendo atualizada a base de conhecimento. Com o auxílio de regras, os dados de tráfego são analisados, fazendo busca nos fatos já armazenados (2), e verificando se existe a necessidade de habilitar uma nova rota, conforme o limite máximo de pacotes (3) estipulado para a primeira rota. Além disso, é possível prever o período de maior uso da rede, antecipando a ação de habilitar essa nova rota, assim como o período decrescente de uso da rede, permitindo o desligamento dessa segunda rota (4). Para as decisões que interfiram na rota, uma mensagem é gerada e enviada para uma conta de *e-mail* (5), conforme definido nas propriedades do agente.

Além da MoDPAI, que efetivamente usou o GeR, outros trabalhos [7,8,9,10] envolvendo o monitoramento de redes e agentes de *software* podem implementar esse padrão. Esses trabalhos, além de utilizarem o protocolo SNMP para o gerenciamento, abordam diferentes métodos para a construção do agente. Com o uso do padrão GeR, os esforços na construção desses sistemas e dos agentes são minimizados, havendo uma concentração em solucionar os problemas de redes utilizando formas diferenciadas para a definição do comportamento e representação da inteligência do agente.

11 Padrões Relacionados

- *Abstract Factory* [30,16]: o GeR utiliza esse padrão para fornecer uma interface de criação de família de objetos relacionados ou dependentes sem a especificar sua classe concreta;
- *State* [30,16]: o GeR utiliza esse padrão para permitir que um objeto alterne seu comportamento de acordo com seu estado interno;
- *Strategy* [30,16]: o GeR utiliza esse padrão para encapsular algoritmos relacionados em subclasses de uma superclasse em comum. Isso permite que a seleção de um algoritmo varie não só através de objetos, mas também durante o tempo.

12 Agradecimentos

Os autores gostariam de agradecer à *Shepherd* Dra. Rossana Maria de Castro Andrade pelas sugestões recebidas durante o processo de *shepherding*.

13 Referências

1. Lucena, C.; ... [et al]. *Software Engineering for Large-Scale Multi-Agent Systems – SELMAS'2002*. Proceedings of The 24th International Conference on Software Engineering. p653-654. Orlando, Florida, USA. 2002.
2. Stallings, W. *Network Management*. IEEE Computer Society Press. 1993. 354p.
3. Stallings, W. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. 3 ed. Addison-Wesley, 1999. 619p.
4. Kavasseri, R.; Stewart, B. *Distributed Management Expression MIB*. Request for Comments 2982, October, 2000.
5. Kavasseri, R.; Stewart, B. *Event MIB*. Request for Comments 2981, October 2000.
6. Bianchini, C. P.; ... [et al]. *Devices Monitoring Tool using Pervasive Computing and Software Agents*. Proceedings of The 2002 International Conference on Security and Management. Las Vegas, Nevada, USA. 2002.
7. Spolidoro, F.; Rodriguez, N. *Distributed Environment for Web-based Network Management*. Proceedings of the 26th IEEE Conference on Local Computer Networks, 2001, p. 41-48.
8. Goldszmidt, G.; Yemini, Y. *Delegated Agents for Network Management*. IEEE Communications Magazine, March 1998.
9. Bieszcazad, A.; Pagurek, B. *Mobile Agents for Network Management*. IEEE Communication Surveys, Fourth Quarter 1998, Vol.1 No.1.
10. Fernandes, H. D. H.; Duarte Jr., E. P.; Musicante, M. A. *ANEMONA: Uma Linguagem de Configuração para Aplicações Práticas de Gerência Distribuída*. Anais do 20o. Simpósio Brasileiro de Redes de Computadores. Búzios, Brasil. 2002.
11. International Standard ISO/TEC 7498-4. *Information processing systems – Open System Interconnection – Basic Reference Model – Part 4: Management framework*. 1989. 13p.
12. Petrie, C. *Agent-Based Software Engineering*. Lecture Notes in AI, Springer-Verlag. 2000.
13. Lange, D. B.; Oshima, M. *Programming and Deploying Java™ Mobile Agents with Aglets™*. Addison-Wesley, 1998. 225p.
14. D'Souza, D. F.; Wills, A. C. *Objects, Components, and Framework with UML: The Catalysis Approach*. Addison-Wesley, 1998. 785p.
15. Booch, G.; Rumbaugh, J.; Jacobson, I. *UML, guia de usuário*. Editora Campus, 2000. 472p.
16. Larman, C. *Utilizando UML e Padrões*. Bookman, 2001. 494p.
17. Orfali, R., Harkey, D. *Client/Server Programming with Java and CORBA*. John Wiley & Sons, Second Edition, 1998.
18. CORBA. *The Common Object Request Broker: Architecture and Specification*. URL: http://www.omg.org. 04/2001.
19. J2SE 1.3. *Java 2 Platform SE v 1.3*. URL: http://java.sun.com/j2se/1.3/docs/api/index.html 01/2001.
20. Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998. 432p.
21. Bianchini, C. P.; Fontes, D. S.; Prado, A. F. *Distributed Software Agents Platform Framework*. First International Workshop on Software Engineering for Large-Scale Multi-Agent Systems in conjunction with 24th International Conference on Software Engineering. Orlando, Florida, USA. 2002.
22. Bianchini, C. P.; ... [et al]. *Distributed Software Agents Platform*. Proceedings of ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications. p 174-179. Foz de Iguaçu, Paraná, Brazil. 2002.
23. Clocksin, W.F.; Mellish, C.S. *Programming in prolog*. 2ed. Springer-Verlag, 1984. 297p.
24. Sterling, L., Shapiro, E. *The Art of Prolog*. 2ed. The MIT Press, 1994. 509p.

25. Lumina Corporate Solution. *Lumina Corporate Solution – Development*. URL: http://www.luminacorp.com. Acessado em Janeiro de 2001.
26. Lumina Corporate Solution; Moura, L. M. *Implementação da KB*. Relatório Técnico Interno. 2000. 88p.
27. Sant'Anna, M. H. B. *Circuitos Transformacionais*. Rio de Janeiro: PUC-RJ, 1999. (Tese: Doutorado em Informática)
28. Microsoft Corporation. *Distributed Component Object Model (DCOM) - Downloads, Specifications, Samples, Papers, and Resources for Microsoft DCOM*. URL: http://www.microsoft.com/com/tech/dcom.asp. May, 2002.
29. OMG Document. *Agent Technology*. Green Paper. 2000. URL: http://www.objs.com/agent/agents_Green_Paper_v100.doc. Acessado em Dezembro de 2001. 67p.
30. Gamma, E.; ... [et al]. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1994. 395p.
31. Bianchini, C. P. *Ferramenta para Monitoramento utilizando Computação Pervasiva e Agentes de Software Inteligentes*. São Carlos: UFSCar, 2002. (Dissertação: Mestrado em Ciência da Computação).
32. Stevens, W. R. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1999. 575p.