Device Monitoring Tool using Intelligent Software Agents

Calebe P. Bianchini
Anhembi Morumbi University
São Paulo/SP – Brazil – Zip 04546-041
calebepb@netsite.com.br

and

Eduardo S. de Almeida, Antonio F. do Prado
Federal University of São Carlos – Computing Department
São Carlos/SP – Brazil – Zip 13.565-905
{ealmeida, prado}@dc.ufscar.br

Abstract

The increasing decentralization of computational resources and the need for distributed and heterogeneous systems management have motivated many researchers to build tools to assist network administrators to perform a large part of their tasks by means of automatic device monitoring. Pervasive computing involves the control of devices distributed throughout computing networks, including the vast network of the Internet. Software Engineering, on the other hand, seeks to improve the quality of software processes and products by reducing their development efforts and costs. Among the techniques employed to achieve these goals are those that employ software agents. The tasks marked by events, often repetitive, from the different application domains, including those of distributed applications, are assigned to software agents programmed in a knowledge base. Combining the ideas of pervasive computing and software agents, this article presents a tool to monitor devices distributed in a network. This tool is component-based and uses intelligent software agents as its main monitoring mechanism.

Keywords: computer network, network management, pervasive computing, software agents, software engineering.

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. A computação pervasiva preocupa-se com o controle dos dispositivos distribuídos pelas redes de computadores, incluindo a vasta rede *Internet*. Por outro lado, a Engenharia de *Software* procura melhorar a qualidade dos processos e produtos de *software*, com a redução dos esforços e custos do seu desenvolvimento. 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. Combinando as idéias de computação pervasiva e agentes de *software*, este artigo apresenta uma ferramenta para monitoramento de dispositivos, distribuídos em uma rede de computadores. A ferramenta foi desenvolvida baseada em componentes e utiliza agentes de *software* inteligentes como o principal mecanismo para realizar o monitoramento.

Palavras claves: redes de computadores, gerenciamento de redes, computação pervasiva, agentes de software, engenharia de software.

1 Introduction

The evolution and expansion of computer networks have opened up possibilities for the development of new applications in this computational area, giving rise to a high demand for improvements in network management to facilitate and accelerate managerial tasks. To meet these new demands, the quality of services requires improvements, particularly services involving network management.

Several lines of research are being developed, involving network management and using mainly the SNMP (Simple Network Management Protocol) [1] management protocol. This protocol emerged in the 1980s and was aimed at standardizing network and equipment management. In 1989, the SNMP was adopted as the network management standard based on TCP/IP [2]. This standardization led to the interconnection of the environments of several manufacturers and machines. SNMP is a set of standards for network management, which includes the protocol and specification of a tree whose leaf nodes contain the monitorable objects of each device.

The spread of the Internet was driven by the development of Web applications, accessed by devices with different software platforms. The emergence of pervasive computing [3, 4] also allowed for access to the Internet through mobile devices such as digital cellular phones, PDAs and pagers, among others. This new resource further extended the Internet's boundaries, allowing for the increasing expansion of Web applications. One of the means of pervasive computer communication is through the mobile communication protocol, WAP (*Wireless Application Protocol*) [5], which is a standard defined by a consortium of four companies for the convergence of the Internet with cellular telephony. This protocol supplies specifications for the development of applications operating in wireless networks. Another form of communication is through the development platform J2ME (*Java 2 Micro Edition*) [6], which meets the needs and peculiarities of the Pervasive Computing devices, in which small and flexible Java applications can be implemented.

Another technique used to speed up network management tasks is the software agent. A software agent [7] is a computational system that inhabits an environment, which may be the physical world, a user – through a graphic interface, a collection of other agents, the Internet, or all these collectively, acting autonomously to achieve the objectives and accomplish the tasks for which it was designed [8, 9]. Some researches in this area use neural networks [10], diffuse logic [11], or simply filters [12] as the agent's intelligence. Another way to define the agent's behavior is through clauses of a logical language, such as Prolog. There is a language called KB, for example, built by Lumina Corporate Solutions [13]. This language, which incorporates the Object-Oriented [14, 15] and Logic [16, 17] paradigm principles underpinning the use of software agents that can communicate with real world objects, supports Object-Oriented programming expressed in clauses in the knowledge base. Its fundamental operation is based on the unification of terms, with two terms being unifiable if a replacement of variables exists that renders them equal.

These and other techniques have been used by researchers who seek solutions for network management problems, owing to the variety of hardware and software platforms utilized, involving different connected devices and requiring ever more dedication and attention on the part of managers for suitable management. Much concern, therefore, focuses on addressing management services, seeking to automate a large portion of administrative tasks.

Based on these ideas, this article presents a tool to monitor devices using pervasive computing and intelligent agents [18]. The main characteristic of this tool is the use of software agents that help the administrator in his decision-making, speeding up the monitoring and control of the monitored device. The exchange of information between administrator and agents can take place directly in the tool or through mobile devices connected to the Internet, allowing for the queries and answers that guide the monitoring to be carried out at any point. Other contributions of this design come from the use of different technologies, highlighting the construction of intelligent software agents to support the semi-automatic management of a network with devices from different platforms. Another aspect worth highlighting is that the tool was designed based on current software engineering techniques, including the use of software standards and components, which facilitate its maintenance to keep up with the constant evolution of computer technology.

This article is divided as follows: section 2 presents the monitoring tool, section 3 analyzes case studies in which the tool was used and, finally, section 4 discusses our conclusions concerning this research.

2 Device Monitoring Tool using Intelligent Software Agents

The Device Monitoring Tool, called MoDPAI, meets the specifications described by the SNMP protocol. The tool, which inhabits a machine, monitors a network's remotely distributed devices to collect information. This information is analyzed by the software agents that are programmed in a knowledge base to manage the monitored devices. According to the degree of intelligence, which is predefined by means of clauses written in the KB language [13], the agents make decisions that change the devices' configurations. Alternatively, the network

administrator may interact directly with the tool through its interface and menus, thus intervening in the monitoring activities.

Figure 1 shows a typical network architecture monitored by the MoDPAI tool. This tool is installed in a computer and offers monitoring resources, including the collection and analysis of information and network management decision-making. Each computer has its own software for communication with the tool through the SNMP protocol. The tool is designed to support the addition of new devices, the configuration of software agents, and other operational requirements of monitoring. In addition to direct interaction through a graphic interface (GUI), a digital cellular phone can be used via WAP interaction (WUI).

The MoDPAI tool can communicate directly with cellular phones or through the Web server to receive queries and send answers. The database stores information about the monitored devices and is consulted during the monitoring activities. Another resource is the knowledge base, in which the agents that assist the network's device monitoring activities are programmed. To reduce the tool's complexity, facilitate its maintenance, and increase its reuse, the tool was designed based on the Component-Based Development (CBD) approach. Reusable software components are clearly identifiable self-contained artifacts that describe or carry out a specific function and have clear interfaces, according to a given software architecture model, appropriate documentation and a defined degree of reusability. The MoDPAI tool was designed with UML (Unified Modeling Language) [15] techniques, following the Catalysis method [19] of Component-Based Development (CBD). Part of the tool was implemented in JAVA and the other part in the logical language, KB, which supports the construction of software agents.

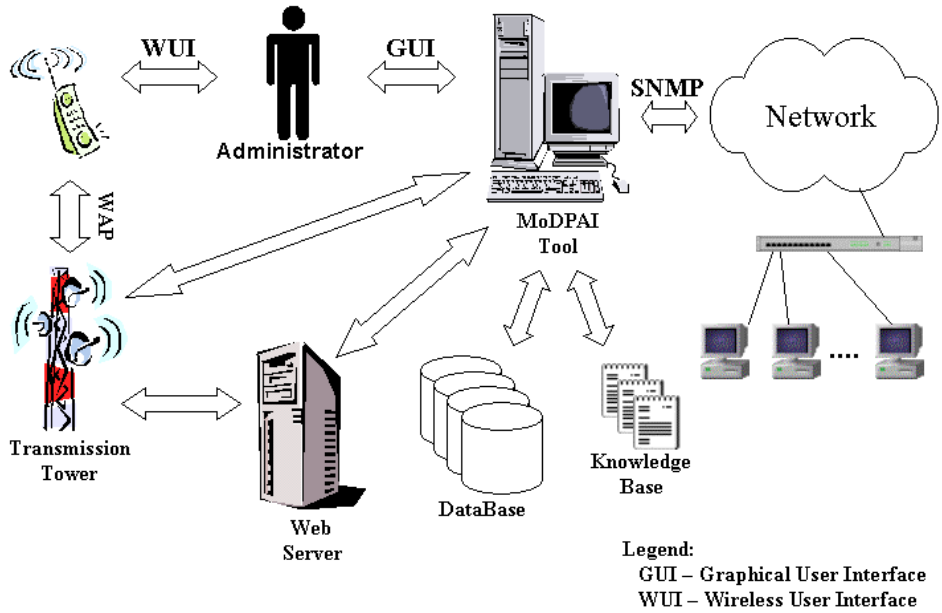

Figure 1. Network monitoring architecture for the MoDPAI tool

2.1 Problem Domain

The first step involved the identification of the tool's main requirements in the domain of the network monitoring application. This phase was carried out with the help of experienced professionals who perform network management tasks, as well as observation techniques, meetings, and available documentation concerning network management problems and the adopted solutions.

Three main actors, whose interaction with the tool is important, were identified during the system's conception, i.e., the Administrator, the Device and the Agent:

- Administrator – identifies and registers the devices to be monitored, creates software agents, defines the agents' knowledge base, requests reports, and receives messages about the tool's monitoring and other administrative and operational activities;
- Device – determines the type of access, authenticates requests for access, receives new configurations, and provides information about current activities and others relating to its monitoring;
- Agent – analyzes the devices' configurations, solves the problems identified by this analysis, sends warning messages about problems to the Administrator, and makes records of the Agent's actions, among other activities involved in the automatic management defined in the knowledge base.

The requisites identified were specified in the applications' Collaboration Models [19], representing the set of actions and the participating objects. These Models were then refined into Use Cases Models [20, 21].

The use cases were used to capture the tool's required behavior and to determine its context and architecture. In addition to the use cases that capture the behavior of objects, other models such as the Sequence Diagram, which details each normal or alternative route of the tool's use cases, were specified at this level in the Problem Domain.

2.2 Component Specification

The MoDPAI tool's components were specified based on Problem Domain modeling [14, 15, 19, 20, 21].

The example in Figure 2 illustrates the components specified in the development phase, which are responsible for monitoring the devices. The dashed arrows indicate the components' interdependence. Thus, the **Device** component that supports the registration of monitored devices, for instance, depends on the **Agent** and **Mib** components.

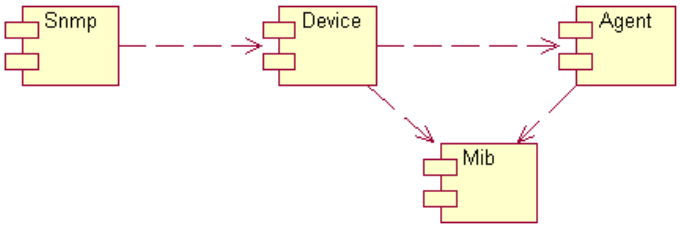

Figure 2. Monitoring component diagram

The **Agent** component [7] defines the behavior of the intelligent software agents that assist the monitoring activities. An agent's behavior is defined by clauses in the knowledge base. The agents can analyze the devices' configurations defined in a Mib and make device management decisions during monitoring. The **Mib** component describes the devices' configurations. To describe a device, the **Mib** component uses a language whose grammar is based on the Asn1 [22] standard. An analyzer that checks the devices' description syntax and uses semantic actions to generate its configuration in the database was developed for this grammar.

The **Snmp** component starts or suspends the monitoring services. This component is based on the SNMP protocol standards and is responsible for the exchange of messages between the application and the devices. The tool's other components were specified similarly, and were grouped into packages according to the 3-tier architecture: User Services, Business Services and Data Services, depicted in Figure 3. This architecture facilitates the reusing and futures modifications and extensions of the tool.

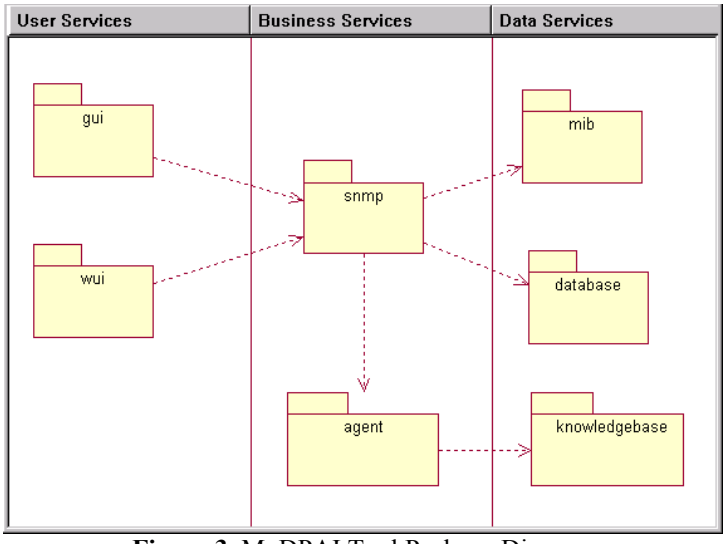

Figure 3. MoDPAI Tool Package Diagram

The **mib** package is responsible for creating and manipulating the object tree of a monitored device. The **database** package contains components for connection and access to the database. The **knowledgebase** package controls the knowledge base, which stores facts and rules for the monitoring of each device. The **snmp** package contains the components that implement the SNMP protocol and is responsible for sending and receiving the tool's requests, including the handling of traps. Traps usually occur when an internal condition of the device is met.

The **agent** package contains components that create, execute and destroy agents, whose behaviors are defined through facts and rules of the knowledge base. The **wui** package contains components for the tool's communication with mobile devices via the WAP protocol. The **gui** package, in turn, contains the tool's graphic interface components.

2.3 Components Inner Project

The Components Inner Project was based on their specifications. This step is obtained by refining the classes of the components specified in the previous step, considering implementation constraints and non-functional requirements. The details of implementation are considered at this level, with particular attention focusing on security, persistence, distributed architecture and implementation language. New classes and interfaces were added to deal with aspects involving component implementation, considering the hardware and software platform defined for its execution.

Based on the specifications of the previous step, the types and their actions, detailed in the use cases and Sequence Diagram, were refined in a Classes Diagram and their relationships. The design of classes sought to utilize standards that facilitate reuse and maintenance, making the services available through interfaces.

The Components Inner Project of the tool's other packages proceeded similarly, modeling their classes, attributes, operations and relationships. The first prototype of the tool was obtained with all the components, allowing the proposed ideas to be tested and validated.

3 Case Studies

Different case studies were developed to validate the MoDPAI tool, which combines the use of pervasive computing and software agents in the monitoring of distributed devices.

3.1 TCP Syn Flooding Attack

In this case study, an analysis is made of a local network server to investigate an attempted attack of the type known as TCP Syn Flooding. If such an attack is detected, a message is sent to the network's Administrator warning him of this invasion.

The TCP Syn Flooding attack [23, 24] explores a failure in the TCP protocol's connection-establishing mechanism. The connection is established through a three-way handshake [25]: first, the client sends a TCP segment with an activated Syn flag to the server; the server then sends back a segment containing the activated Syn flags and Ack to the client's address; and finally, the client sends back a segment containing the activated Ack flag to the server.

This attack consists of overloading the server with TCP segments containing activated Syn flags, adulterating the originating address. When the server receives this segment, it sends back a second segment to the nonexistent address. The segment transmitted by the server will remain unanswered until the limit time is reached, thereby preventing the TCP from completing the three-way handshake.

The intention of such attacks is to send several Syn requests to one or more of the server's TCP portals so as to overload the request lines, causing the unavailability of the services executed by the attacked portals.

To detect probable attacks of this type, the management object **tcp.tcpAttemptFails** [26, 1, 25] was used, which computes the number of times TCP connections pass from the SYN-SENT or SYN-RCVD state to the CLOSED state, added to the number of times these connections pass from the SYN-RCVD to the LISTEN state.

An operating server awaiting connections remains in the LISTEN state until it receives a Syn request. Upon receiving such a request, the server goes to the SYN-RCVD state, in which it awaits confirmation of the Syn-Ack segment that was transmitted. Because this confirmation does not materialize, the server remains in the SYN-RCVD state until the limit time is reached, when it passes on to the CLOSED state, causing an increment of **tcp.tcpAttemptFails**.

Tests carried out at the server demonstrated that, at constant 5-second intervals, the **tcp.tcpAttemptFails** delta remained constant at 0, with the server in a normal operating condition. Even when available services were accessed (www, ssh, samba), this delta remained stable. After the attack, the delta increased, reaching a mean value of 300 units over that period.

Based on the above results, a software agent was built using the delta having a value of 30 units in the knowledge base, limiting the failures to, at most, five per second. Should an attack be detected, a message is sent to the tool's screen and to the Administrator's e-mail address. The KB code is depicted on Figure 4.

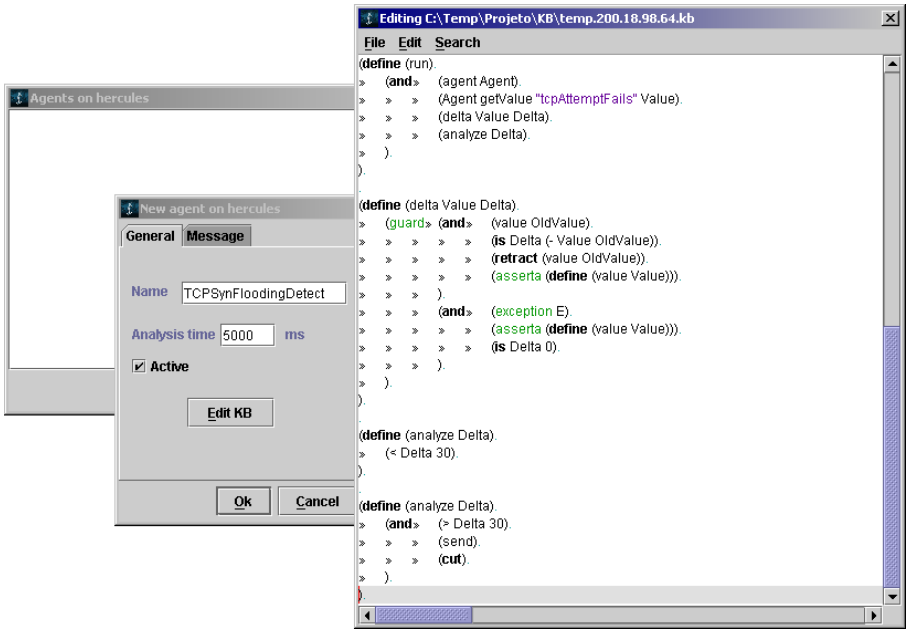

Figure 4. The Software Agent that detect the TCP Syn Flooding attack

The **analyze** rules check the delta's value. The first **analyze** rule verifies if the delta's value is lower than 30 (< **Delta 30**). If this fails, it uses a backtracking mechanism [16, 17], passing on to the second rule, which checks whether the delta is actually greater than 30 (> **Delta 30**), sends a warning message to the Administrator (**send**), and prevents a new backtracking (**cut**).

Adulterated segments were created, using specific programs [23] that sends them to the server. At the onset of the attack, the agent sent the warning message to the tool's screen and to the specified e-mail address, notifying the Administrator about the event. Figure 5 illustrates the screens of the messages sent.

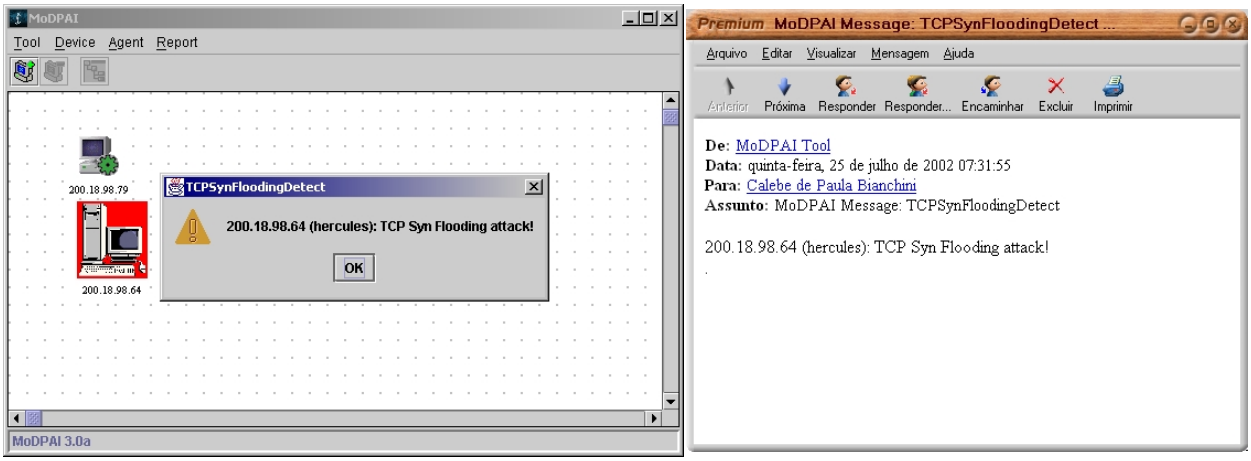

Figure 5. Messages send to the Administrator about the TCP Syn Flooding attack

3.2 CDNet

Among the case studies that have been carried out, some worth mentioning are the monitoring of a computer network in Department of Computer Sciences at UFSCar (Federal University of São Carlos), called CDNet, which consists of 1 Gateway (FreeBSD), 1 Web Server (SunOS), 1 E-mail Server (SunOS), and 6 workstations (Windows NT Workstation).

Figure 6 shows the main interface of the MoDPAI tool. The administrator uses this interface to directly access the operations of device registration and network management.

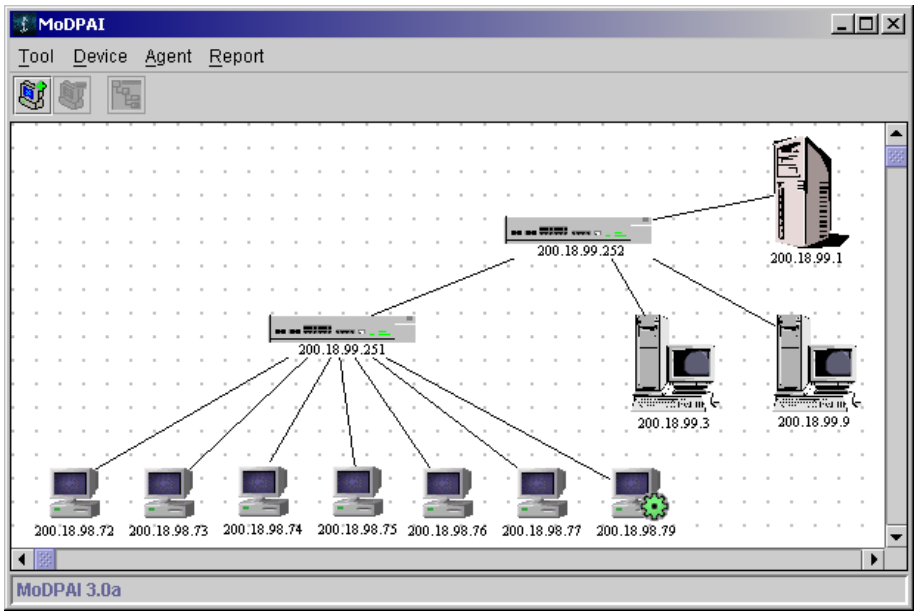

Figure 6. Main screen of the MoDPAI Tool

The process begins with the registration of the devices to be monitored, using the *Device* menu. The devices are identified preferentially by their IP (Internet Protocol) addresses. In the case of the CDNet, there is a Gateway with IP 200.18.99.1, a *Web* Server with IP 200.18.99.3, an E-mail Server with IP 200.18.99.9, and workstations with IPs ranging from 200.18.98.72 to 200.18.98.77. Figure 6 shows the distribution of these CDNet registered devices. Figure 7 depicts the device registration screens to record the IP number, the socket-type communication ports [2, 25], the type of services and their Mib. For the latter, there are some predefined services listed in the *Mib Type* combo-box, which can be edited by using the *New Mib* and *Edit Mib* keys. Part of the mib for Windows 2000 is illustrated in Figure 7, with part of the definition of the monitorable objects.

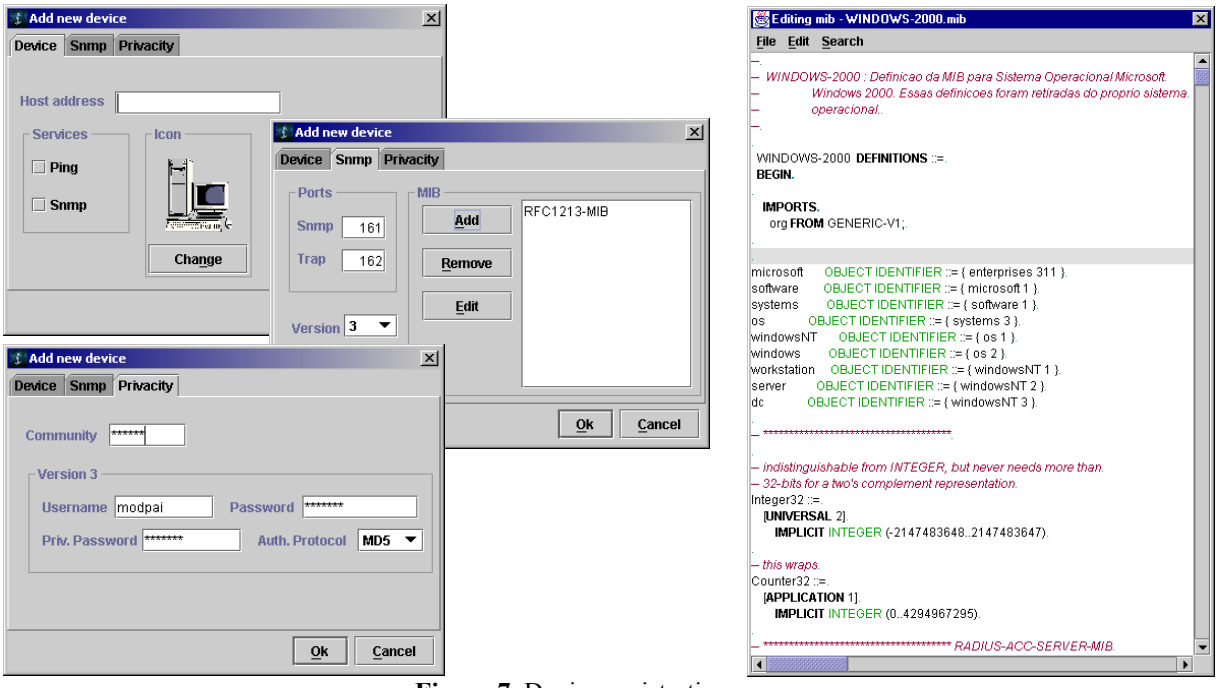

Figure 7. Device registration screen

After collecting the device information, the Administrator analyzes it to define the behavior of the software agent that will be used for monitoring. Figure 8 portrays a screen with information collected from a device. The Administrator obtains information from this interface, as in the case of the Memory object, which supplies all the data concerning the device's memory, both free and in use.

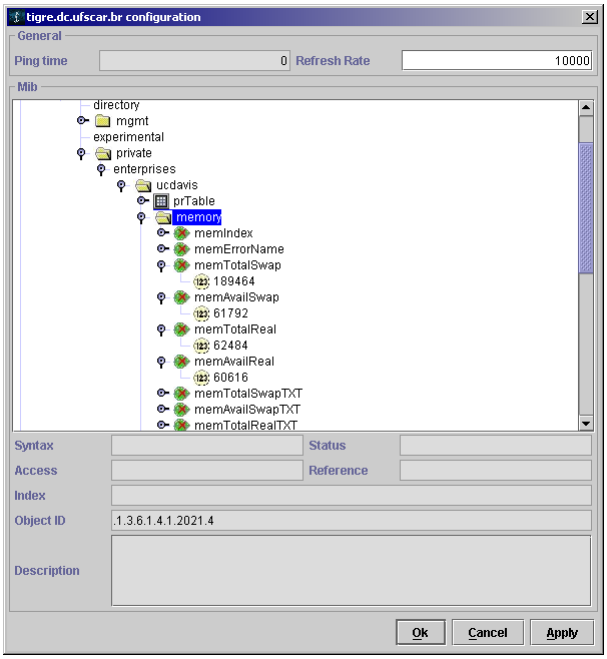

Figure 8. Data collected from a device

This analysis, which is based on the information collected over a given time period, is used to characterize the behavior of the software agents that will monitor the devices.

In this case, the administrator analyzes the following:

- The number of TCP and UDP packages received and sent from the intranet through the gateway, opening up a second route of data transmission through a new interface when this number is about to cause an overload of the first data transmission route;
- The number of Web connections established, relating their times and verifying the period in which the greatest number of connections are made in order to characterize it as a critical period;
- The shortest and longest period between two e-mail connections, considering whether or not the e-mail service considered critical failed; and
- The beginning and ending of the use of each station, verifying the volume of information exchanged by the network and the memory and disk resources available.

After this data has been analyzed, the software agents can be specified, and their behaviors defined based on facts and rules in the knowledge base. The KB language [13] is used to program the agents' behavior. Figure 9 shows the screens for the specification of a software agent. The KB language code is verified and, if syntactically correct, it is stored in the knowledge base.

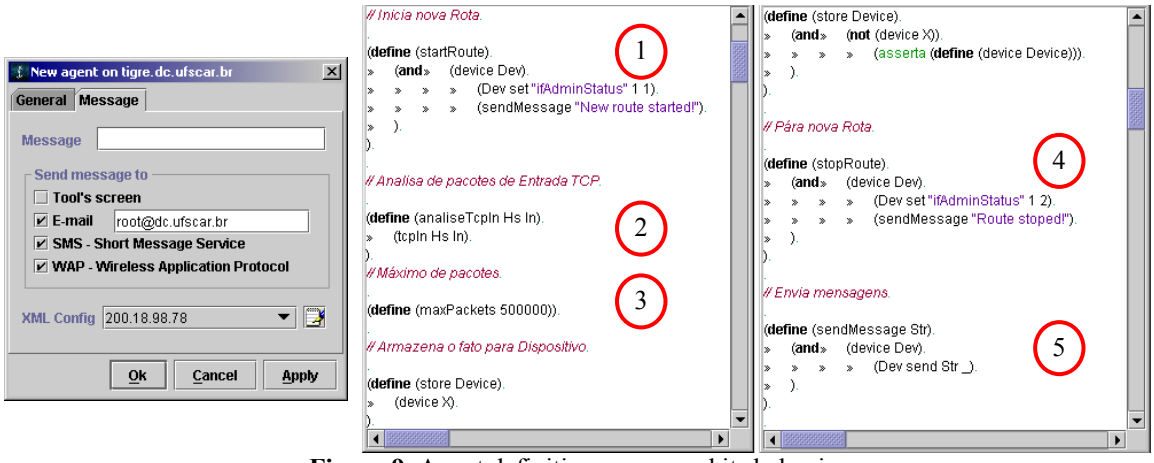

Figure 9. Agent definition screen and its behavior

Each agent interacts with a single device, identifying and correcting the problem analyzed in the previous step and acting according to the rules defined in the knowledge base. Based on the decision it has made, the agent can send a message to the Administrator, reporting on this decision.

The right hand side of Figure 9 shows part of the KB code that defines the behavior of the agent responsible for analyzing the number of packages that pass through the gateway, enabling a new route (1) if necessary. In this code, the facts that indicate the period and number of packages transmitted are created dynamically, maintaining the knowledge base updated. The traffic data are analyzed based on rules, consulting stored facts (2) and checking the need for a new route to be enabled, according to the maximum limit of packages (3) stipulated for the first route. In addition, the period of most intensive use of the network can be predicted, anticipating the action of enabling the new route, as well as the period of least intensive use of the network, allowing for the second route to be disabled (4). When decisions are made that interfere in the route, a message is generated and sent to an e-mail address and a cellular number phone (5), as shown on the left hand side of Figure 9.

Figure 10 shows the messages sent by the agent after it has analyzed the data, notifying the Administrator about the beginning and end of the operation of the new data transmission route.

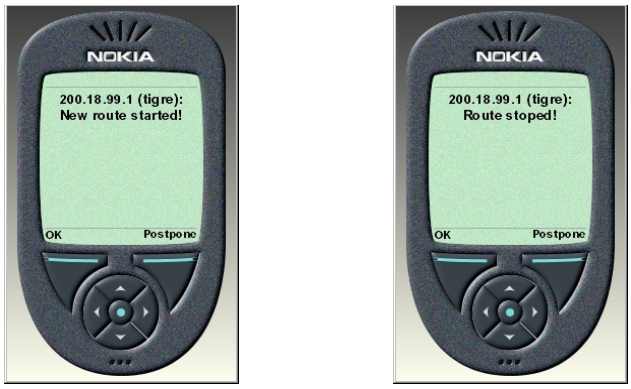

Figure 10. Notification message sent by the agent

Interaction with the tool can also be done through a cellular phone or any other WAP-compatible mobile device. Most of the functionalities defined in the tool's graphic interface can be accessed by cellular phone, with the advantage of remote interaction with the devices at any time. Figure 11 shows the screens for interaction with the tool via cellular phone, listing the registered devices, their respective agents and their Mib trees.

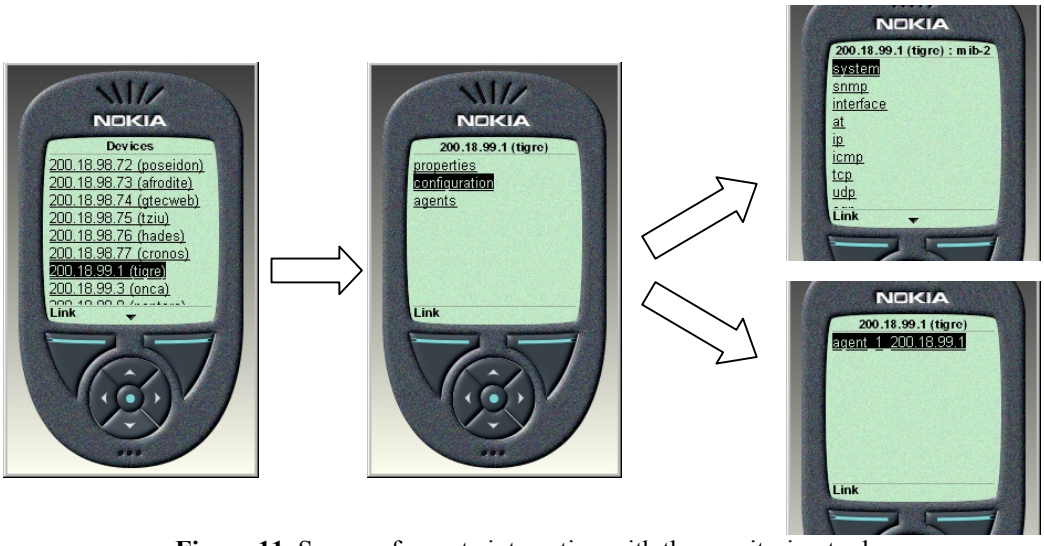

Figure 11. Screen of remote interaction with the monitoring tool

A second interface was also implemented for remote connections, using J2ME platform classes [6, 27]. This involves specialized JAVA classes for small devices that can execute the bytecodes of JAVA language. The network administrator can interact through this interface with the MoDPAI tool, as illustrated in Figure 12, which shows the remote access of the MoDPAI tool via the emulators of a pervasive device.

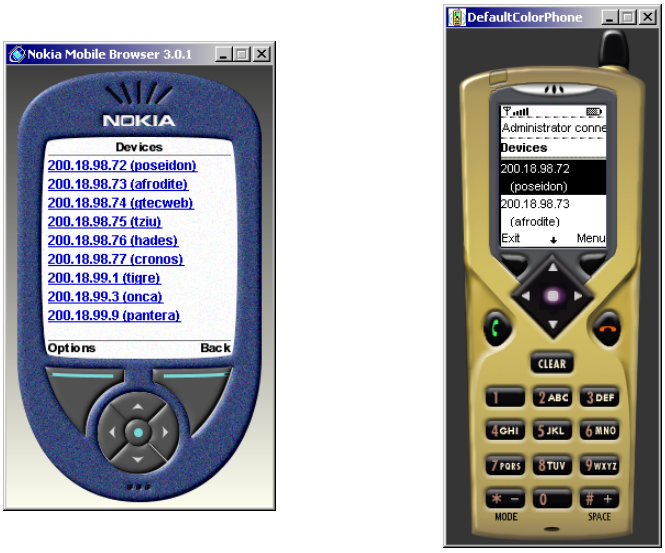

Figure 12. The remote access of the MoDPAI tool

4 Conclusion

This article presented and discussed a device monitoring tool that uses pervasive computing and intelligent software agents. The tool allowed for validation of the idea of integrating the SNMP protocol and intelligent software agents described in KB to monitor a network's devices.

Different techniques have been used by researchers seeking solutions for network management problems, given the variety of hardware and software platforms utilized, which involve different connected devices and require the increasing dedication and attention of administrators to properly manage them. Therefore, much concern focuses on management services, seeking to automate a large part of the administrator's tasks. The main characteristic of the tool presented herein is the use of software agents that help the administrator's decision-making activities, speeding up the monitoring and control of the monitored devices. The administrator and agents can exchange information directly through the tool or through pervasive devices connected to the Internet, allowing queries and responses for guidance of the monitoring activities to be carried out at any point.

Although the use of pervasive devices with the Internet is already known, this system is differentiated mainly by its use of current software engineering, computer network and artificial intelligence resources in the tasks of monitoring different devices distributed in a local network or in the Internet.

Additional contributions derive from the integration of the logical and object-oriented paradigms, using software components. This integration allows for the fundamental characteristics of each paradigm to be used in the construction of the intelligent software agent. The implementation of an Intelligent Software Agent facilitates its use in several other areas besides network device monitoring. These areas include information searches in the Internet, e-business and other areas in which object orientation and artificial intelligence have been applied successfully.

5 References

[1] Stallings, Willian. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. 3. ed. Addison-Wesley, 1999. 619p.

[2] Tanenbaum, A. S. *Computer Networks*. 3 ed. Prentice Hall, 1996. 848p.

[3] Hansmann, U.; ... [et al]. *Pervasive Computing Handbook*. Springer-Verlag, 2001. 409p.

[4] Helal, S. *Pervasive Java*. IEEE Pervasive Computing: Mobile and Ubiquitous Systems. p 82-85. n 1. v 1. January-March 2002.

[5] WAP Forum Specifications. *WAP Forum*. URL: http://www.wapforum.com/what/technical.htm. Accessed in 26/01/2001.

[6] Sun Microsystems, Inc. *Java™ 2 Platform Micro Edition (J2ME™) Technology for Creating Mobile Devices*. White Paper. 2000. URL: http://java.sun.com/products/cldc/wp/KVMwp.pdf. Accessed: 2002, Janeiro. 42p.

[7] Bianchini, C.P., Sobral, D.S., Prado, A.F. *A Distributed Software Agents Platform Framework*. 1st. International Software Engineering for Large-Scale Multi-Agent Systems – SELMAS. Proceeding of 23rd. International

IEEE/ACM Conference of Software Engineering. Orlando, Florida – USA. May 19 2002.

[8] Maes, P. *Artificial life meets entertainment: lifelike autonomous agents. Communications of the ACM*, v. 38, n. 11, p. 108-114, 1995.

[9] Franklin, S., Graesser, A. *Is it an agent or just a program?: a taxonomy for automonous agents*, In: International Workshop On Agent Throries, Architecture And Languages, 3, 1996.

[10] Nascimento, A.; Franklin, M.; Oliveira, M. *Desenvolvendo agentes inteligentes para a gerência pró-ativa de redes ATM*. XVII Simpósio Brasileiro de Redes de Computadores, Salvador, BA, Brasil, p.681-696, 1999.

[11] Carvalho, E.; Belchior, A. D.; Souza, J. N. *Gerenciamento pró-ativo distribuído baseado em lógica difusa*. XVII Simpósio Brasileiro de Redes de Computadores, Salvador, BA, Brasil, p.649-664, 1999.

[12] Schulze, B.; Madeira, E. R. M., Ropelatto, P. *MomentA: Gerenciamento de serviços usando agentes móveis em ambiente CORBA*. XVII Simpósio Brasileiro de Redes de Computadores, Salvador, BA, Brasil, p.649-664, 1999.

[13] Lumina Corporate Solution, *Clipping Lumina* URL: http://www.luminacorp.com/clipping.htm. Accessed 26/01/2001.

[14] Furlan, J. D. *Modelagem de Objetos através da UML*. 1. ed. Makron Books, 1998. 329p.

[15] Booch, G., Rumbaugh, J., Jacobson, I. *UML, guia do usuário*. 1. ed. Editora Campus, 2000. 472p.

[16] Clocksin, W.F.; Mellish, C.S. *Programming in prolog*. 2. ed. Springer-Verlag, 1984. 297p.

[17] Sterling, L., Shapiro, E. *The Art of Prolog*. 2. ed. The MIT Press, 1994. 509p.

[18] Bianchini, C. P. *Ferramenta para Monitoramento de Dispositivos utilizando Computação Pervasiva e Agentes de Software Inteligentes*. São Carlos: UFSCar, 2002. (Master Dissertation)

[19] D'Souza, W. *Objects*, *Components and Framework with UML – The Catalysis Approach*. Addison Wesley, 1998.

[20] Booch, G. *Object-Oriented Analysis and Design with Application*. Addison-Wesley, 2nd Edition, 1994. 589p.

[21] Larman, C. *Utilizando UML e Padrões*. Bookman, 2001. 494p.

[22] Dubuisson, O. *ASN.1 Communication between heterogeneous systems*. 1. ed. Morgan Kaufman Publ Inc, 2000. 562p.

[23] www.phrack.org. *Project Neptune*. vol 7. article 48. archive 13. 1996, July. URL: http://www.phrack.com/phrack/48/P48-13. Accessed: 2002, January.

[24] Autor anônimo. *Segurança Máxima*. Editora Campus, 2000. 823.

[25] Stevens, W. R. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1999. 575p.

[26] Network Working Group. *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*. 1991. 70p.

[27] java.sun.com - The Source for Java(TM) Technology. *Java 2 Platform SE v 1.3*. URL: http://java.sun.com/j2se/1.3/docs/api/index.html Accessed: 2001, January.