# System Architecture of a Distributed Expert System for the Management of a National Data Network*

** Ioannis Vlahavas[1], Nick Bassiliades[1], Ilias Sakellariou[1], Martin Molina[2], Sascha Ossowski[2], Ivan Futo[3], Zoltan Pasztor[3], Janos Szeredi[3], Igor Velbitskiy[4], Sergey Yershov[4], Sergey Golub[4], and Igor Netesin[4]

[1] Department of Informatics, Aristotle University of Thessaloniki, 54006 Thessaloniki Greece
{vlahavas, nbassili, iliass}@csd.auth.gr
[2] Department of Artificial Intelligence, Technical University of Madrid, 28660 Boadilla del Monte, Madrid, Spain.
{mmolina, ossowski}@isys.dia.fi.upm.es
[3] ML Consulting and Computing Ltd, ML Kft, H-1011 Budapest, Gyorskocsi u. 5-7., Hungary.
{futo, szeredi}@ml-cons.hu
[4] International Software Technology Research Center Technosoft, 44 Glushkov avenue, Kiev, 252187, Ukraine.
{vel, yershov, golub, netesin}@netman.ts.kiev.ua

**Abstract.** The management of large data networks, like a national WAN, is without any doubt a complex task. Taking into account the constantly increasing size and complexity of today's TCP/IP based networks, it becomes obvious that there is a demanding need for better than simple monitoring management tools. Expert system technology seems to be a very promising approach for the development of such tools. This paper describes the system architecture of ExperNet, a distributed expert system for the management of the National Computer Network of Ukraine, and the implementation of the tools used for its development. ExperNet is a multiagent system built in DEVICE, an active OODB enhanced with high level rules, that uses CS-Prolog II to implement the communication facilities required. The system employs HNMS+ and BigBrother, two modified versions of existing network management tools, in order to obtain a complete view of the monitored network.

**keywords:** Distributed expert systems, agents, network management, distributed prolog.

* The work described in this paper is funded by the EU INCO-Copernicus project ExperNet: A Distributed Expert System for the Management of a National Network, No 960114

** The order in which the authors appear does not reflect their contribution to the work described in this paper.

## 1 Introduction

The exploitation of a large WAN cannot be effectively achieved without a user-friendly and intelligent network management software. Existing network management software cannot meet the requirements of such large-scale networks, mainly because it offers, in most cases, only monitoring tools. One of the most important directions for the practical application of network management software, is its enhancement with higher level decision support and diagnostic services.

ExperNet is a distributed expert system for the management of the National Network of Ukraine, developed in the framework of a joint EU funded research project. The development of expert systems for WAN management is only at research and experimental stage. There are difficulties in the formalization of such a task because of the incompleteness and lack of adequate information about network state, the large scale of behaviour characteristics, and the continuous evolution of the network environment. The absence of practical and verified expert systems for large, complex modern technological systems like WANs, demonstrates the complexity of the task and pose a great challenge ahead.

Distributed expert systems using co-operative problem solving strategies with new general conflict detection and conflict resolution mechanisms, seem to provide a feasible but also an elegant solution to the WAN network management problem. Such a distributed expert system, requires an sufficient communication facilities and an efficient expert system shell that is able to cope with large amounts of data and offer multiple knowledge representations. Another important point in the design of the system is that there must be an efficient way of determining the network state and capturing important network events, in other words an efficient monitoring schema. The implementation of such a functionality has to be in accordance with the existing network monitoring facilities, namely it has to adopt to the Simple Network Management Protocol (SNMP) in order to have control over the existing network devices.

This paper describes the ExperNet system architecture, that fulfils the above requirements, and presents the system components.

## 2 The ExperNet System Architecture

As the network management problem of the ExperNet project has turned out to be inherently distributed, we conceive the ExperNet architecture as multiagent system. At each management node there is one *agent*, specialised in managing the network area that the node is responsible for. In consequence, the structure of the system architecture goes in line with the structure of the pre-existing organisation of the experimental zone of the network. The overall architecture of the system is shown at figure 1. Each ExperNet agent is attached to an HNMS+ server; the latter provides necessary information about the state of the network to the former. Additional information is provided by Big Brother, a tool for monitoring local computer resources. The agents are developed in DEVICE, and communication facilities are provided by CS-Prolog II. All system components, shown in the figure, are described in the rest of the paper.

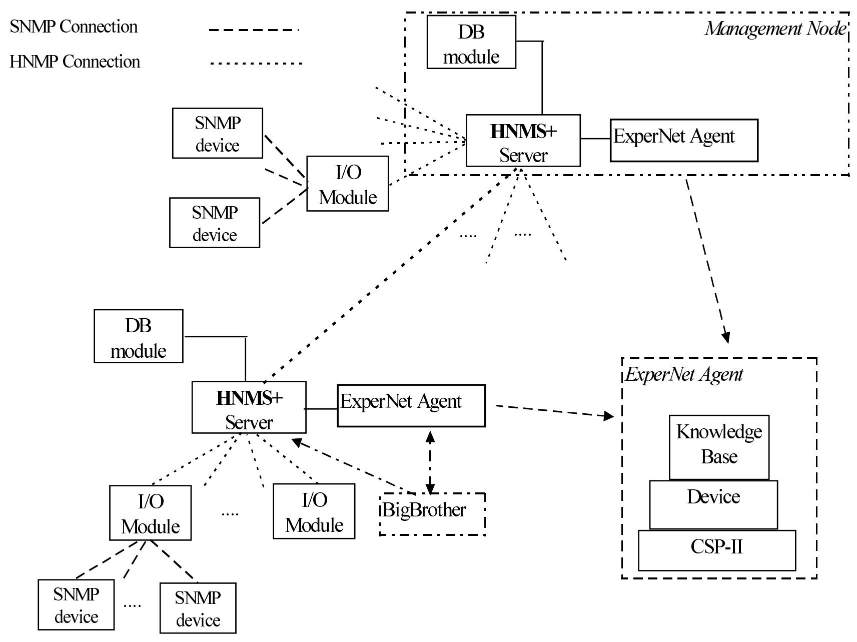

**Fig. 1.** ExperNet System Architecture

Each agent comprises two types of knowledge: local knowledge for individual problem-solving (i.e. for local network management) and social knowledge for coordination (i.e. for harmonising local network management with the activities of acquaintance nodes).

## 2.1 Local Problem-solving

In order to characterise the knowledge model of each agent we have applied advanced knowledge engineering techniques. The particular characteristics of the domain of network management include complex problem-solving tasks (classification, diagnosis, planning, etc.) which suggests to use the concept of model-based system development, that has recently become popular among researchers and knowledge engineers, for the development of large and complex knowledge-based systems. For instance, some recent methodologies and tools such as Kads [17], KSM [10], Protégé-II [15], follow this model-based approach. According to this, we have modelled the agents' problem-solving competence as a three step process: (1) *symptom detection*, where administrators watch out for symptoms of undesired network states and behaviours (e.g. a certain service -ftp, www, etc.- does not respond, a host is unreachable, over/under-utilisation of links or equipment, etc.); (2) *diagnosis*, which is done by discriminating hypothesis of different degrees of precision on the basis of network data and the result of exploratory actions to find the causes of symptoms (e.g. inadequate capacity for some resource, unbalance of workload and resources, resource malfunctions, etc.) and (3) *repair*, where a sequence of repair actions is proposed to solve the problem.

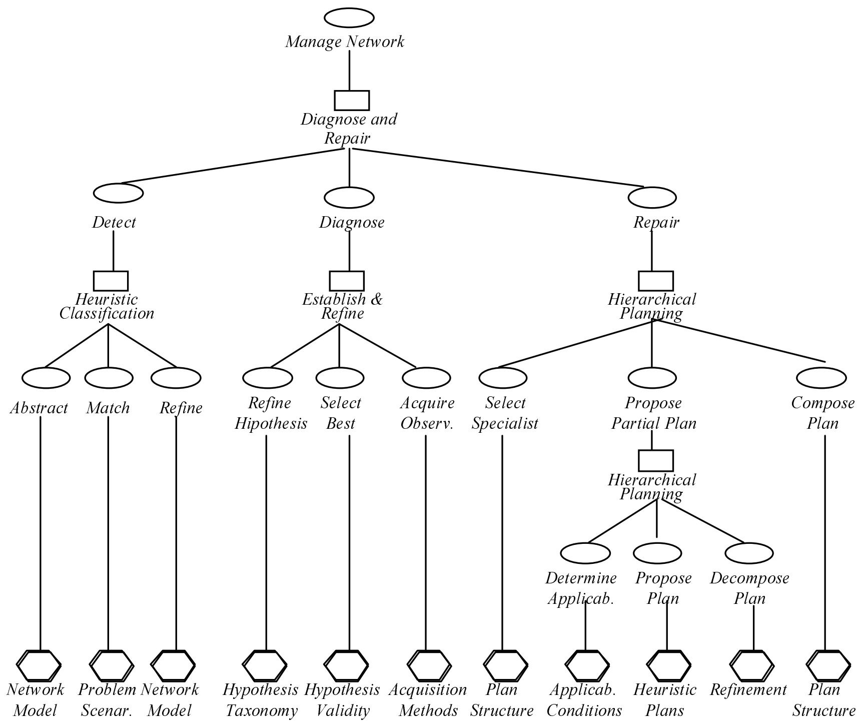

**Fig. 2.** Local Problem Solving

Each step is realised by customising generic knowledge modelling methods [17]. The *heuristic classification* problem-solving method [8] constitutes a typical reasoning structure for classification problems and is used for symptom detection. It follows three steps (abstraction, matching and refinement) which, in our model, are supported by two types of knowledge bases: one about the network model for abstraction and refinement, that includes a declarative representation of the network structure, and another that uses a set of problem scenarios relating symptoms and observables. For diagnosis, the *establish and refine* method is used [7]. This method can be conceived as an abstract reasoning pattern based on a heuristic search in a taxonomy of hypotheses of problems. Our particular adaptation of the *establish and refine* method makes use of three primitive inferences: (1) *refine* problem hypotheses uses a knowledge base represented by a taxonomy of hypothesis classes using the is-a relation; (2) *select best hypothesis* makes use of knowledge about the validity of hypotheses (represented using frames) to establish whether any of the input hypothesis can be proved, and (3) *acquire additional observables* determines the sequence of exploratory actions to get additional observables by using a knowledge base about acquisition methods (represented by rules). Finally, the *hierarchical planning* method is used for the repair task. This method is based on a search in a hierarchy of specialists that are knowledgeable about partial abstract plans, which are dynamically composed during the reasoning [5]. The particular instance of the hierarchical planning method that we use in the network management domain, makes use of four specialists (top level, fault detection, performance management and configuration) and uses five primitive inferences supported by four types of knowledge bases.

## 2.2 Social Co-ordination

An important part of a node administrator's time is not spent in local problem-solving, but in co-ordinating its work with other administrators. In the particular case of ExperNet, three types of situations require co-ordination: (1) *Information acquisition*, when additional observations are needed, which are available (or can be acquired) within the agent society, but are not accessible (or cannot be acquired) by the node itself. (2) *Responsibility conflicts*, when different agents intend to perform similar tasks. (3) *Interest conflicts*, when one agent does not agree with its role in a certain repair plan or with the effects that some plan will have on its local situation.

We model the process of co-ordination in the above situations as *conversations* [1], i.e. logically coherent sequences of agent interactions. Conversations that cope with responsibility conflicts are very simple, as they just involve one interaction, transferring the responsibility for some task from the sender to the receiver. We propose three kinds of conversations of this type: *diagnosis and repair delegation, repair delegation and isolation delegation.* Information acquisition problems are managed by means of the *observable acquisition* and the *plan refinement* conversations, in the course of which a needy agent asks some target agent for a certain observable or plan; the latter may either reply with this information or by notifying its inability (or unwillingness) to facilitate it. *Plan acceptance* conversations manage interest conflicts, where all affected agents need to agree in order that a proposed plan be accepted.

Interactions within a conversation are based on a message-passing model. Every message that is exchanged during such interactions can be considered as Speech Acts, as by emitting it the sender wants to influence the behaviour of the receiver [14]. The table 1 resumes the different messages that are used in the network management model as well as their intended effect on the receiver.

Within conversations there are various degrees of freedom for the involved agents, as they usually may choose from several behaviour options (in the simplest case to accept or to reject a request). An agent's choice is not just determined by information respecting its local situation, but also by its knowledge and experience with other nodes in the network. It thus maintains *agent models*

**Table 1.** Types of Messages and Interactions

| **Message types** | **Receiver's intended reaction** |
|---|---|
| ASK FOR observable | acquires observable & informs sender |
| ASK FOR plan acceptance | decides about acceptance & informs sender |
| ASK FOR plan refinements | refines plan & informs sender |
| DO diagnosis and repair | performs diagnosis and repair tasks |
| DO isolation | performs problem isolation |
| DO repair | performs repair task |
| ANSWER WITH observable | informs about observable |
| ANSWER WITH plan acceptance | informs about plan acceptance |
| ANSWER WITH plan refinements | informs about plan refinements |

(this type of knowledge is also refered to as "acquaintance model" [9]) of all acquaintances that it interacts with including itself [11]. These models endow the agent acquires with additional capabilities: (1) *problem interest:* checks whether the modelled agent is believed to interested in being notified about a problem (e.g. because it is indirectly affected by that problem and wants to isolate it in order to keep its effects as local as possible); (2) *plan interest:* checks whether the modelled agent needs to be notified about a given plan (either because it is involved in it or because its side-effects concern the modelled agent); (3) *plan rights:* checks whether there is a need to obtain the agreement of the modelled agent for enacting a given plan; (4) *observation capability*, checks whether the modelled agent is believed to by capable of acquiring the value of a given observable; (5) *diagnosis capability*, determines if the modelled agent is capable of performing diagnosis for a given symptom; (6) *plan repair capability*, checks whether it can elaborate a plan for a given problem; (7) *plan refinement capability*, analyses whether the agent may refine a given abstract plan for a given problem.

On this basis, the three step local problem-solving cycle of an agent can be extended, leading to the control loop shown in figure 3, followed by ExperNet agents.

1. Detect symptoms.
2. Inform agents interested in the symptoms, in order to diagnose them.
3. Diagnose problem (if the agent is responsible). If there are missing observables, ask agents for acquiring the corresponding value.
4. Inform agents interested in problems, in order to isolate them.
5. Inform agents interested in problems, in order to repair them.
6. Generate a repair plan (if the agent is responsible). If necessary, asks agents for plan acceptance

**Fig. 3.** Structure of the method followed by an agent to manage the network.

## 3 The Device Expert System Shell

For the implementation of the knowledge model, the DEVICE [2, 3] expert system shell is selected, since it presents a number of interesting features, like multiple rule type support and Object Orientation. DEVICE (**D**ata-driven & **EV**ent-driven rule **I**ntegration using **C**omplex **E**vents) is a sequential Knowledge Based System that runs on top of ADAM and EXACT. The former is an OODB built in Prolog, while the latter is an extension of ADAM with events and ECA rules (Figure 4). DEVICE is in fact, an active OODB enhanced with high-level rule facilities. It provides the infrastructure for the smooth integration of production and deductive rules into an active OODB that generically supports event-driven rules only. The integration is based on the compilation of the condition of both high-level rule types into a *discrimination network* that consists of simple and complex events which record and combine database modifications that could possibly make a rule fire.

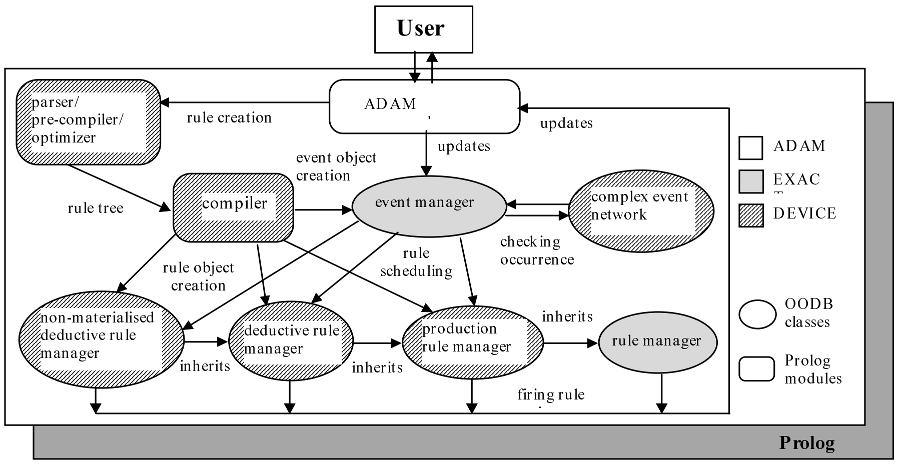

**Fig. 4.** The architecture of the DEVICE system.

A rule base in DEVICE, can be a mixture of ECA, production and deductive rules. The two latter are high-level rules, whose integration into the active OODB has been smoothly achieved in DEVICE. Backward chaining or goal-driven rules are also supported in DEVICE in the form of methods. Methods are pieces of Prolog code, therefore a backward chaining declarative language is provided. For the efficient matching of the production rules, DEVICE smoothly integrates a *RETE-like* discrimination network into an active OODB system as a set of first class objects by mapping each node of the network onto a complex event object of the active database system. In order to bring the full functionality of production systems into an active database system, heuristic conflict resolution strategies

(OPS5 approach), namely refractoriness, recency and specificity, have been incorporated into the rule selection mechanisms of the integrated environment. The production cycle of DEVICE is presented in figure 5.

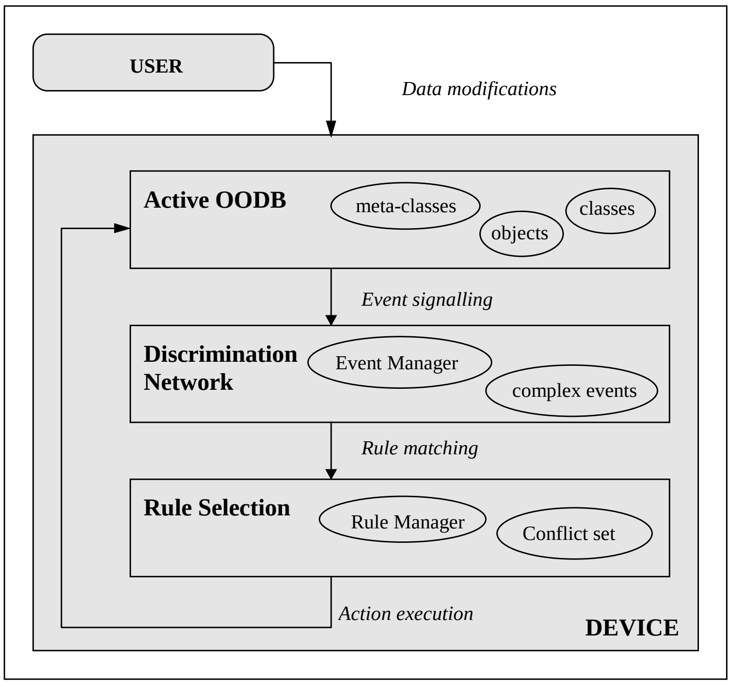

**Fig. 5.** The production cycle of DEVICE.

The resulting system is a flexible, yet efficient, KBS that gives the user the ability to express knowledge in a variety of high-level forms for advanced problem solving in data intensive applications.

The ability of DEVICE to handle large collections of data is important for the development of the ExperNet system, since the information in any WAN concerning the status of the various network devices is large. It has to be noted here, that the OO architecture and data types supported by DEVICE naturally adopt to existing representations of network management information, such as MIB and HNMS+ MIB, providing an easy mapping of network variables to DEVICE objects. For the needs of ExperNet, DEVICE has been implemented in CS-Prolog-II, a language which, among others, offers extended communication facilities. The latter, in conjunction with the ability of integrating Prolog code with production rules in a simple, clear and robust manner, offers an expert system shell in which communication can be easily implemented, thus offering a powerful platform for the development of any agent based system.

## 4 The CS-Prolog II System

In the development of any multiagent system, a crucial issue is the implementation of the communication facilities that are required for the co-operation and co-ordination of the involved agents. In ExperNet these facilities are developed using CS-Prolog II a distributed Prolog enhanced with networking facilities.

### 4.1 General overview

CS-Prolog II distributed Prolog system is being developed from 1995. The syntax and the built-in procedures of the language are based on the standard ISO/IEC 13211-1. It is extended with features that were not included in the standard, like modularity, multitasking, real-time programming and network communication.

CS-Prolog II, supports the communicating sequential process programming methodology in a Prolog environment. On a single processor machine the concurrent processes are controlled by a time-sharing scheduler. The inter-process communication is ensured by a rendezvous mechanism (synchronous message passing through communication channels). Processes can backtrack, however communication is not backtrackable. The channel based communication had recently been extended with networking capabilities. This makes possible message passing between different CS-Prolog II applications across the Internet. CS-Prolog II also provides communication with foreign (non CS-Prolog) applications, an interface to relational data base systems, real time programming methods like cyclic behaviour, reaction to predefined events, timed interrupts, etc.

The system consists of three main components: a compiler, a linker and a runtime system. The compiler contains a pre-processor similar to what is found in C compilers. The integrated development environment is based on OSF/Motif and runs on UNIX platforms. The main advantage of this environment is the multi-window trace utility in which the debugging messages of separate processes appear in separate windows.

### 4.2 Networking facilities

As a natural extension of CS-Prolog II channel concept, the external communication conceptually consists of unidirectional message streams. In order to facilitate speed-up of external communication, asynchronous message passing is introduced as an option. Send operation in this case still remains blocking but the condition for continuing execution is the availability of sufficient buffer space instead of the commencement of the matching receive operation.

For the Prolog programmer the communication environment appears as a homogenous address space (community). All partners will be accessed via channel messages. A separate mechanism is introduced for connecting channels to external partners. The most important entity for this task is the so-called port. Ports represent incoming message substreams. They are explicitly created and play the role of a sender for a CS-Prolog II channel specified at the time of port creation. The other end of the channel can be used in the same way as the receiving end of any internal channel. At port creation, a buffering parameter can be specified indicating the size of message buffer.

Another important notion in CS-Prolog II is the *connection.* A connection is the representation of an outgoing message stream. Its attributes include the local channel, the partner's name and the partner's port (if partner is not foreign) to where the stream is directed. Its size of the connection's message buffer can be set at creation. If the value of the buffering attribute is greater than zero then more than one message can be stored in the connection buffer, allowing several send operations to complete without blocking.

In a centralised subnetwork of CS-Prolog II applications managed by a (possibly foreign) manager program, the following types of partners can appear for a specific CS-Prolog II program:

- *Private partners;* their addresses have to be available in advance for the program (hardwired in the program, obtained from a file, e.t.c.).
- *Net partners,* which have signed up at the manager, and our program included them in its local picture of the network. The address of a net partner is obtained from the manager.
- *Latent partners,* who are known by manager, but our program didn't include them in its local network picture. The address of a latent partner (and some other attributes too) can be asked from the manager.

In the current TCP/IP implementation of the CS-Prolog II low-level communication protocol, in order to be able to communicate with a net partner, a configuration process has to be performed as for private partners. In other words the program has to add explicitly this partner using special built-in predicate.

In future CS-Prolog II versions, if the underlying network layer provides the possibility of communicating with partners with known addresses without building a specific transmission path to them, the explicit configuration of net partners can be omitted.

## 5 Capturing the Network State

The size and complexity of National Computer Network of Ukraine are of the most important issues in its management [13], creating problems in the area of full-fledged data collection for ExperNet intelligent agents. We have approached this problem by modifying the Hierarchical Network Management System (HNMS) and BigBrother network monitoring tool, in the way presented in the following.

### 5.1 The HNMS+ System

The HNMS system prototype version was developed to cover the network management needs that arose because of the continuing installation of large, high speed local and wide-area networks for the Numerical Aerodynamic Simulation (NAS) Faculty at the NASA Ames Research Center [12]. This prototype version of HNMS is available on the Internet.

The prototype version of the available HNMS consists of two types of modules, which typically reside on separate hosts throughout the network. The *Server module* is the hub for the network data; it provides a center for dissemination of global topology and status information. The *User Interface (UI)* module resides on workstations with graphics capabilities and provides access to real-time or logged data. All inter-module communication is done using the Hierarchical Network Management Protocol (HNMP) described in [12]. The protocol requires the use of new HNMS MIB, which defines a set of variables in addition to standard SNMP variables [6, 16]. HNMS MIB objects represent IP network elements within HNMS system. Each object is identified by a unique number, its HNMS id, which is assigned by the server. Objects belong to one of the following classes which represent network entities or other useful information about network management: Internet, Network, Subnet, Interface, Processor, Site, Equipment, Administrator, or Address.

HNMS provides four types of status diagrams, each representing the view of state of a network element using a colour code. These diagrams are updated by the server, reflecting changes of the element's status. The *WAN diagram* depicts the state of the IP network and the routers over a geographical reference (e.g. a map of Ukraine). The *Site diagram* represents all LANs that are connected to the routers at a given site. The *Custom diagram* allows the user to construct a diagram with any set of network elements he wishes to observe. Finally, the *Object diagram* is a textual display of the HNMS variables.

Although *Input/Output (IO)* modules were mentioned in the general architecture of HNMS, the prototype version did not support multiple IO modules, and as a consequence, there was no true hierarchy in its structure. Therefore, the implementation of IO module functionality was necessary to collect local information about the behaviour of particular subnetworks that are compound parts of National Network of Ukraine. The IO modules reside on hosts located at strategic points within WAN (regional, district, metropolitan-area subnetwork) and handle actual data collection. Our IO modules use SNMP [6, 16] protocol for local data collection from the SNMP agents attached on the actual network devices. These modules pass filtered management data, up to the *server module.* In accordance with the overall architecture of HNMS, data are sent from IO to servers only when their values change. Thus the hierarchical installation of IO modules allows to avoid flooding the network with management traffic and creating bottlenecks when management information is directed to ExperNet agents. The new HNMS is named HNMS+ and it is a true hierarchical distributed system which fully supports HNMS functionality and extends it with new features.

Additionally, the fourth type of module mentioned in the HNMS external specification, the *database module*, was developed. The database module is an SQL front-end process that stores HNMS+ MIB variable values in a PostgreSQL database with a frequency given by the user (usually approx. 1 minute). The database module interacts with the HNMS+ server/IO module store only when variables of local server/IO modules change in order to avoid network overloading by SQL requests.

Finally, an interface of the ExperNet agents with HNMS+ was implemented in CS-Prolog II as a special *Knowledge-based intelligent processing (KBIP).* The KBIP module is an application that obtains, through HNMP protocol, information about network traffic and utilisation of the network elements.

On each node HNMS+ provides to the ExperNet agents an immediate perception of the state of network. Using KBIP modules, ExperNet agents not only are able to immediately determine the general state of the network but also be notified by HNMS+ about important network events.

### 5.2 The BigBrother Monitoring System

In order to sufficiently monitor the network state and services availability, the information obtained by standard SNMP agents is not enough. An important issue is the evaluation of particular TCP/IP network services quality (like ftp, http, smtp and nntp), services reliability and local host resources like CPU, disk and so on.

Big Brother is a free Web-based UNIX Systems monitor, developed by Sean MacGuire [4]. Big Brother consists of simple shell scripts which periodically monitor local system conditions (*Local System Monitor* or *bb-local.sh*) and network connectivity (*Network monitor* or *bb-network.sh*) as well as *Intra-machine communications programs (bb, bbd, nettest).* Disk usage, CPU loading, ftp, smtp and http servers, and important processes can be kept track of. The results of monitoring are reported in a status matrix (using a colour code) for each system/area combination, which is displayed on a *central monitoring station (Display Server)* and presented through a Web based user interface.

For the needs of ExperNet, we have integrated HNMS+ and BigBrother in order to achieve monitoring of TCP/IP services and remote computer resources by the ExperNet intelligent agents. HNMS+ MIB was extended to incorporate the additional monitoring values of the status matrix of BigBrother that correspond to all services/resource types included in the latter. HNMS+ server (or IO module) analyses a local log file created by BigBrother and fills out the previously mentioned MIB variables.

Additionally to existing BigBrother processes, a UNIX daemon (module) was developed that offers the possibility of remote UNIX command invocation. This was necessary since, ExperNet intelligent agents, in some cases, require information that cannot be obtained directly from HNMS+, but only through command execution on the monitored remote hosts, as for example information obtained by the "traceroute" and "tcpdump" packet monitoring utilities. Although in such cases the usual "rsh" UNIX command could be used, the above solution was preferred since it offers the possibility to restrict the set of commands that are allowed, through appropriate configuration of the module, thus leading to a more flexible and secure system.

To conclude, the modified BigBrother provides information to ExperNet intelligent agents through HNMS+, not only about the status of the most important TCP/IP network services, but also about the operational parameters of the individual monitored hosts. It also offers a relatively secure remote command invocation that allows the system to better monitor or even control the network.

## 6 Current Status and Future Work

Currently the major part of the project has been successfully completed. This part comprises the design of the overall system, as well as the implementation of the various components that have been described in the present paper. A large part of the knowledge base has been encoded in the language of DEVICE. We are now approaching the final phase of the implementation which consists of the development of a graphical user interface, the installation of the final system in Ukraine and the verification phase.

One simple but yet very important way in which ExperNet can be extended is the enrichment of the knowledge base so that it will be able to handle a larger set of network failures. Currently the system covers a rather limited number of such cases, since our main goal was to have a pilot system that will successfully demonstrate the applicability of expert system technology to the management of large networks.

Depending on the results of the application of ExperNet in the management of the National Network of Ukraine, the system could be adopted to provide services in larger data networks in Europe as well as other countries, and lead to an improvement of the end-user services, as well as relieve the administrators of much of the burden they have to face in their everyday practice. Taking into account the growth rates of the TCP/IP based networks world-wide and their constantly increased complexity, such functionality might not only be desirable but also essential in the near future.

## References

1. Barbuceanu M., Fox S.: COOL: A Language for Describing Coordination in Multi Agent Systems. Proc. ICMAS, 1995
2. Bassiliades N. and Vlahavas I.: "DEVICE: Compiling Production Rules into Event-Driven Rules Using Complex Events", Information and Software Technology, Vol. 39(5), pp. 331-342, Elsevier Science, 1997.
3. Bassiliades N. and Vlahavas I: "Processing Production Rules in DEVICE, an Active Knowledge Base System". Data & Knowledge Engineering, Vol. 24(2), pp. 117-155, Elsevier Science, 1997.
4. BigBrother. A Web-based Unix Network Monitoring and Notification System. Available at URL: http://www.iti.qc.ca/users/sean/bb/bb.html
5. Brown, D. and Chandrasekaran, B.: Design Problem-solving: Knowledge Structures and Control Strategies. Morgan Kaufman, 1989
6. Case J., Fedor M., Schoffstall M., Davin J. Simple Network Management Protocol, RFC 1157, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, 1990.

7. Chandrasekaran, B., Johnson, T., and Smith, J.: Task-Structure Analysis for Knowledge Modelling. Communications of the ACM 35 (9), 1992
8. Clancey W.: Heuristic Classification. Artificial Intelligence 27, 1985
9. Cockburn, D. and Jennings, N.: ARCHON: A Distributed Artificial Intelligence System for Industrial Applications, Foundations of DAI. O'Hare and Jennings (eds.), Wiley, 1996
10. Cuena J., Molina M.: KSM: An Environment for Knowledge Oriented Design of Applications Using Structured Knowledge Architectures. Applications and Impacts. Information Processing 94. Vol 2 K. Brunnstein and E. Raubold (eds.). Elsevier, 1994. (see also: http://www.isys.dia.fi.upm.es/ksm).
11. Cuena J., Ossowski S.: Distributed Models for Decision Support. To appear in Introduction to Distributed Artificial Intelligence. Weiss and Sen (eds.) AAAI/MIT Press, 1998
12. George Jude A., Schecht Leslie E. The NAS Hierarhical Network Management System In "Integrated Network management III", H.-G. Hegering and Y. Yemini (Editors), Elsevier Science Publishers, Amsterdam, 1993.
13. Matov Alexander. The development of Internet-like networks in Ukraine Networks and Telecommunications, Kiev, no.2, 1997. - pp.4-11.
14. Muller, H.-J.: Negotiation Principles in Foundations of DAI. O'Hare and Jennings (eds.), Wiley, 1996
15. Puerta A.R., Tu S.W. and Musen M.A.: Modelling Task with Mechanisms. International Journal on Intelligent Systems. Vol 8, 1993.
16. Rose M. The Simple Book: An Introduction to Management of TCP/IP-based Internets. Prentice-Hall, Inc., New Jersey, 1991.
17. Wielinga B.J., Schreiber A.T., Breuker J.A.: "KADS: A Modelling Approach to Knowledge Engineering". Knowledge Acquisition, 1992.