ExperNet: An Intelligent Multiagent System for WAN Management

Ioannis Vlahavas, Nick Bassiliades, and Ilias Sakellariou, *Aristotle University of Thessaloniki*
Martin Molina, *Technical University of Madrid*
Sascha Ossowski, *Universidad Rey Juan Carlos*
Iván Futó, Zoltán Pásztor, and János Szeredi, *ML Consulting and Computing*
Igor Velbitskiyi, Sergey Yershov, and Igor Netesin, *Technosoft International Software Technology Research Center*

The authors describe ExperNet, an intelligent multiagent system developed to assist in managing large-scale data networks. The system assists network operators at various nodes of a WAN to detect and diagnose hardware failures and network traffic problems, suggesting the most feasible solution through a Web-based interface.

Managing a large data network, such as a national WAN, is a complex and cumbersome task. Existing network management software cannot meet the requirements of the increasing size and complexity of today's TCP/IP-based networks, because, in most cases, it provides only simple monitoring tools. We can't fully exploit a WAN without user-friendly, intelligent, network management software enhanced with diagnostic and decision support services.

Expert systems provide a feasible and elegant solution in this direction. Currently, the development of expert systems for WAN management is only at the research and experimental stages. Formalizing such a task is difficult because network state information is usually inadequate and incomplete, the scale of behavior characteristics is large, and the network environment continually evolves. Moreover, an efficient network-monitoring schema must exist.[1] The implementation of such functionality must be in accordance with existing network monitoring technology—namely, the Management Information Base (MIB) and the Simple Network Management Protocol (SNMP)—to control existing real-world network devices.

Network management also requires coordination among operators of autonomous network nodes. To closely resemble network management's distributed nature, multiagent problem-solving techniques can help model the functionality of individual agents as well as the interactions taking place between them.[2] Multiagent technology is preferable to other distributed problem-solving techniques because it offers enhanced modularity, reactions to environmental changes, and reusability. This article presents the architecture, implementation, operation, and evaluation of the ExperNet system, a multiagent expert system that we developed for Ukraine's national TCP/IP WAN under the framework of a joint EU-funded project. ExperNet helps network operators manage a WAN with its

- monitoring tools for capturing network state;
- platform for logic-based applications;
- efficient and extensible expert system shell;
- fast heuristic detection, diagnosis, and repair of network failures;
- cooperative problem-solving strategies; and
- user-friendly Web-based interface.

The system architecture

ExperNet consists of a hierarchically structured architecture, with each level consisting of one or more *management nodes* (see Figure 1). Each node encapsulates one or more lower-level management subnodes and manages a network area, assisted by a local intelligent agent. In our example, the system's hierarchical structure is in accordance with the structure of the Ukrainian national network's preexisting organization, which is divided into regional, district, and metropolitan area subnetworks.

ExperNet assists network operators at various nodes of Ukraine's national network by detecting and diagnosing network failures and traffic problems. Furthermore, it suggests the most feasible solution, given resource and temporal constraints. To achieve this goal, each agent communicates with other agents on the network to collaborate on problem diagnosis and repair, using social knowledge for coordination. Agents acquire network data that concern device installation, removal, reachability, and operational parameters from conventional network management software.

We put many different pieces of software together by extending the distributed Prolog system CS-Prolog II (see the related sidebar) and building sophisticated information exchange interfaces between each agent and its local network software components. We developed the HNMS+ system by extending the HNMS network management software and integrating the BigBrother monitoring tool for local computer resources (see the related sidebar), so that HNMS+ can dispatch information to local intelligent agents. Finally, we reimplemented the Device knowledge base system on top of CS-Prolog II to provide a flexible, high-level, expert system shell on which to implement the intelligent agents.[3] The agents' knowledge base consists of local problem-solving competence and social interaction for coordinating actions among them.

As Figure 1 shows, each ExperNet agent is attached to an HNMS+ server, which provides information about network state. BigBrother provides additional local computer resource information. The agents themselves are developed in Device, and CS-Prolog II provides communication facilities.

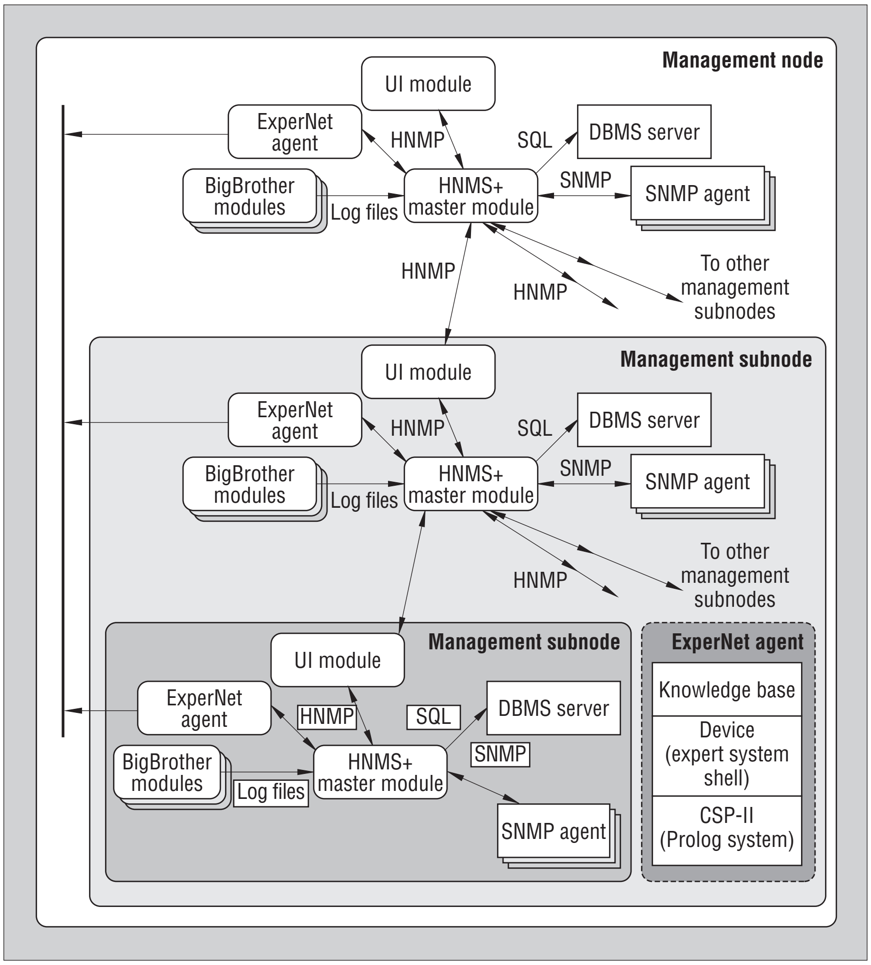


Figure 1. The ExperNet system architecture. The line on the left shows that ExperNet agents at all levels communicate with each other.

Knowledge model for ExperNet agents

Each ExperNet agent has two types of knowledge:[4] local, for individual problem solving (local network management), and social, for coordination (harmonization of local network management with acquaintance node activities). ExperNet agents use these types of knowledge during the problem-solving cycle, which involves

1. Detecting symptoms.
2. Having agents diagnose the symptoms.
3. Diagnosing the problem (if the agent is responsible). If there are missing observables, ask agents to acquire the corresponding value.
4. Informing agents interested in problems to isolate and then repair them.
5. Generating a repair plan (if the agent is responsible). If necessary, ask agents for plan acceptance.

Local problem solving

The network management domain's characteristics, which the project consortium identified during the knowledge acquisition phase, include complex problem-solving tasks (such as classification, diagnosis, and planning), which lead to model-based system development.[5,6] We have modeled the agents' problem-solving competence as a three-step process—detect, diagnose, and repair (see Figure 2)—each step consisting of customized, generic, knowledge-modeling methods.[5]

Initially, during symptom detection, the system watches for signs of undesirable network states and behaviors—a certain service not responding, an unreachable host, or over- or underused links. ExperNet tackles the symptom detection phase as a heuristic classification task, based on the system's data-driven, reactive nature. The task is accomplished by three steps—abstraction, matching, and refinement—which, in our model, are supported by network model knowledge and a set of problem scenarios relating symptoms and observables.

Subsequently, an agent performs diagnosis by discriminating hypotheses of different degrees of precision based on network data. This diagnosis is a result of exploratory actions to find the causes of symptoms (such as inadequate capacity for some resource, workload and resource imbalance, and resource malfunctions). Numerous network components can malfunction in many ways, so there are several possible diagnoses for ExperNet. Because speed is crucial for ExperNet's operation, we chose the *establish and refine* method[7] for diagnosis, so that we could quickly focus on network malfunctions with-

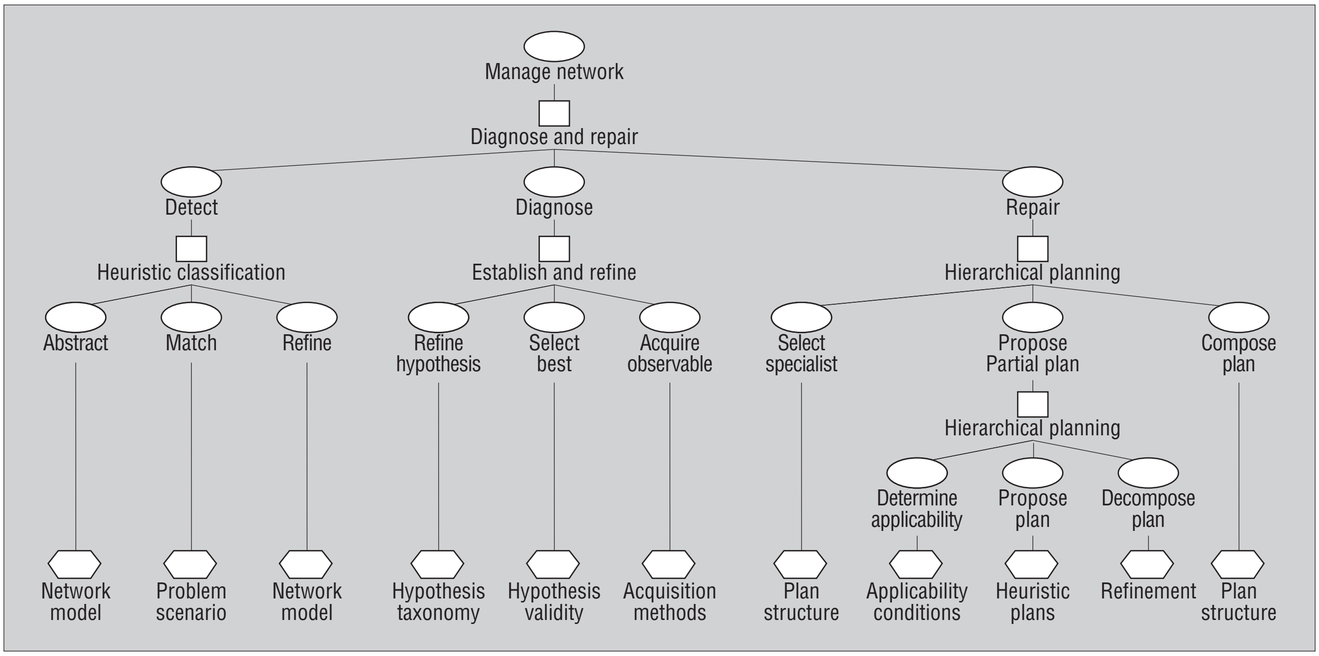


Figure 2. ExperNet agents' local problem-solving structure.

out using deep models of network components.[4] This method uses an abstract reasoning pattern based on a heuristic search in a taxonomy of problem hypotheses; we adapted it to use three primitive inference steps:

- *refine the problem hypotheses*—uses a knowledge base represented by a taxonomy of hypothesis classes related through the *is–a* relation;
- *select the best hypothesis*—uses knowledge about the validity of hypotheses (represented by frames) to establish whether we can prove any of the input hypotheses; and
- *acquire additional observables*—determines the sequence of exploratory actions to get additional observables by using a knowledge base of acquisition methods (represented by rules).

Finally, the local problem-solving strategy of ExperNet agents generates a sequence of repair actions. Due to the complexity of network problems, we apply a hierarchical planning strategy for repair, which is based on a search in a hierarchy of specialists (top level, fault detection, performance management, and configuration) that are aware of any partial abstract plans and dynamically composed during the reasoning process.[8]

Social coordination

An important fragment of a node administrator's time is not spent on local problem solving but on coordinating his or her work with other administrators. ExperNet requires coordination in three types of situations:

- *Information acquisition*—additional observations are needed, which are available within the agent society but are not accessible by the node itself.
- *Responsibility conflicts*—different agents intend to perform similar tasks.
- *Interest conflicts*—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 coordination process in these situations as *conversations*[9]—logically coherent sequences of agent interactions.[4] Conversations that cope with responsibility conflicts are simple because they just involve one interaction, transferring the responsibility for some task from sender to receiver. We have used three kinds of conversations of this type: diagnosis and repair delegation, repair delegation, and isolation delegation. Agents manage information acquisition problems by means of the observable acquisition and the plan refinement conversations, in the course of which an inquiring agent asks some target agent for a certain observable or plan. The agent could reply either with this information or by expressing its inability (or unwillingness) to facilitate it. Plan acceptance conversations manage interest conflicts, where all affected agents need to agree for a proposed plan to be accepted.

Interactions within a conversation are based on a message-passing model to closely reflect the cooperation model of human network operators. We can call every message exchanged during such interactions a speech act, because the sender wants to influence the receiver's behavior by emitting it.[10] Table 1 shows the different messages 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, because they usually choose from several behavior options. An agent's choice is not just determined by information regarding its local situation, but also by its knowledge and experience with other nodes in the network. Thus, an agent maintains agent models of all acquaintances with which it interacts, including itself.[2]

ExperNet implementation

The implementation of ExperNet's knowledge model is based on the Device active knowledge base system,[3] which runs on top of a Prolog-based active object-oriented database and provides many interesting features, such as support for multiple rule types (deductive, production, and event-driven rules) and object orientation. We used Device not only as an expert system shell for developing the knowledge base itself, but also as an integrator for network information and agent communication. Device's

The CS-Prolog II System

A crucial issue for the cooperation and coordination of agents in a multiagent system is the implementation of communication facilities. In ExperNet, we developed these facilities with CS-Prolog II, a distributed Prolog system.[1,2]

General overview

The language's syntax and built-in procedures are based on the standard ISO/IEC 13211-1. Features not included in the ISO standard, such as modularity, multitasking, real-time programming, and network communication, have helped extend the language.

CS-Prolog II supports the communicating sequential processes programming methodology of Hoare[3] (hence "CS" in the name) in a Prolog multitasking environment. Prolog processes can be distributed among several processors on a multiprocessor machine; a time-sharing scheduler controls the concurrent processes running on a single processor. The interprocess communication is achieved with a rendezvous mechanism (synchronous message passing through unidirectional communication channels). Processes can backtrack, but communication is not backtrackable. The system also provides an interface to relational database systems, real-time programming methods such as cyclic behavior, reaction to predefined events, and timed interrupts.

Networking facilities

As a natural extension of the CS-Prolog II channel concept, the external communication conceptually consists of unidirectional message streams. To speed up external communication, asynchronous message passing is an option. Communication with foreign (non–CS-Prolog) applications is also possible.

For the Prolog programmer, the communication environment appears as a homogenous address space, called a *community*, which consists of one or more fellow applications with which the program can communicate (called partners). Partners are accessed through channel messages, while a separate mechanism connects channels to external applications (called foreign partners). The most important entity for this task is the so-called port, which represents an incoming message substream. Ports are explicitly created, and they play the role of sender for a CS-Prolog II channel, which is specified at the time of port creation.

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

In a centralized subnetwork of CS-Prolog II applications managed by HNMS+, the following types of partners can appear for a specific CS-Prolog II program:

- *Private partners*—addresses must be available in advance for the program.
- *Net partners*—signed up at HNMS+ and included in the network's local picture.
- *Latent partners*—known by HNMS+ but not included in the local network picture.

To communicate with a net partner, the current TCP/IP implementation of the low-level communication protocol requires the program to explicitly add that partner to its communication environment in advance, using a special built-in predicate.

Working with foreign partners

Foreign applications do not understand the message format used in Prolog-to-Prolog communication, so they need an agent (or mediator) to perform the appropriate data and protocol conversion. At present, there are two defined mediators: ASCII, for plain-text communication, and HNMS, for communicating with the HNMS server.

Conceptually, a local mediator communicates with a remote mediator, hosted at the foreign partner, to address the dock it offers. Data sent by the remote mediator are accepted at the local mediator's dock. Docks are similar to ports in the sense that they play the same role in communication. The difference is in the way a dock is prepared for operation and connected implicitly by the mediator on the foreign partner's behalf. To configure a foreign partner, the application program should create a dock and create an appropriate mediator, and configure the desired foreign partner.

Once the foreign partner is successfully created, the procedure to follow in message exchange is almost the same as for any Prolog partner. The most important restriction in communicating with foreign partners is in the set of rules specifying what kinds of Prolog terms they accept and produce.

References

1. I. Futó, "A Distributed Network Prolog System," *Proc. 20th Int'l Conf. Information Technology Interfaces*, SRCE Univ. Computing Centre, Univ. of Zagreb, Croatia, 1998, pp. 613–618; www.ml-cons.hu/dload-e.html#csprof (current 23 Aug. 2001).

2. I. Futó, "Prolog with Communicating Processes: From T-Prolog to CSR-Prolog," *Proc. 10th Int'l Conf. Logic Programming*, D.S. Warren, ed., MIT Press, Cambridge, Mass., 1993, pp. 3–17.

3. C.A.R. Hoare, "The Communicating Sequential Processes," *Comm. ACM*, vol. 21, no. 8, Aug. 1978, pp. 666–677.

ability to handle large data collections, such as the status of a WAN's network devices, is important for developing the ExperNet system. Device's object-oriented architecture and data types naturally correspond to the representation of network management information, such as standard SNMP MIB and HNMS+ MIB variables, providing an easy mapping to Device objects. (The MIB is the set of variables needed to monitor and control TCP/IP-based network components, and the SNMP is a protocol that defines how to access and modify this information remotely.) For ExperNet, we reimplemented Device in CS-Prolog II, a language that 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 we can easily implement agent-based system communication.

Capturing network data

Device acquires data concerning network entities and their operational parameters from the HNMS+ system through the Device HNMS+

The HNMS+ Network Management System

Experience has shown that traditional monolithic network management systems cannot address all the issues involved with full-fledged data collection on large TCP/IP networks. The Hierarchical Network Management System[1] (HNMS) is a software system designed to assist the network operator in managing such networks. In ExperNet, we approach network monitoring and management by

- developing HNMS+ (extending HNMS functionality to suit our needs and correcting several shortcomings of its prototype version), and
- integrating the BigBrother network-monitoring tool with HNMS+.

HNMS+: Extending the prototype

The HNMS prototype did not fully adhere to its specification, making the collection of network state information for ExperNet difficult. The prototype did not support multiple separate I/O modules, because their functionality was merged into the single server module, and consequently, there was no true hierarchy. The server module is the system's hub; it provides a center for disseminating global topology and status information. Its responsibility is to maintain an up-to-date network model and process all requests from other modules that connect to the server. I/O modules directly monitor their LANs through SNMP and promiscuous Ethernet monitoring, then forward the status of network objects to the server. Multiple I/O modules were necessary for collecting local information about subnetwork behavior; I/O modules should be able to serve subscription requests from intelligent agents. As a result, in the new HNMS+ system, we combined server and I/O module functionality on a single module, called the master module, but also provided the ability to have a hierarchy of many such modules.

Master modules reside on hosts located at strategic points within a WAN and use the SNMP protocol[2] for local data collection from the SNMP agents attached on their LANs' actual network devices, playing the role of I/O module. They also accept information from low-level modules, playing the role of server module, and build a network representation, passing filtered management data up to the immediate higher-level modules with the HNMP protocol. Data is propagated only when their values change to avoid bottlenecks created when the network is flooded with management information to ExperNet agents. This process is recursive and terminates on the highest network level. We needed three such levels to cover Ukraine's experimental network zone.

HNMS's HNMP extends SNMP by introducing a new set of management information base variables (HNMS MIB) in addition to standard MIB variables.[2] HNMP is generally suitable for HNMS+, but we needed one vital change in the announcement of new network objects. New ExperNet agents that subscribe to their local master module are announced recursively upward and downward in the hierarchy, because each agent should know all existing agents, regardless of hierarchical position.

The network administrator can connect to a master module at any level using an UI module, shifting the scope of network monitoring. Similarly, any agent can connect to any master module and obtain information on the corresponding subnetwork state. However, in ExperNet we decided that to minimize traffic, agents should connect only to their local master module.

We added a new class of network entities in the HNMS+ MIB (called hnmsService) to provide support for TCP/IP high-level services (such as FTP) and any system resource on a network host (such as processor load). Furthermore, we developed a database module to store regularly the network status and performance data on a PostgreSQL server (www.postgresql.org) to support statistical analyses. The database module interacts with the HNMS+ master module only when the values of the local master module's variables change to avoid network overloading by SQL requests.

Integrating BigBrother

An important issue in network management is the evaluation of TCP/IP network service quality and reliability. SNMP agents cannot provide such information, so we integrated BigBrother into HNMS+ to monitor TCP/IP services and remote computer resources with ExperNet agents.

BigBrother is a free Web-based system monitor (www.iti.qc.ca/users/sean/bb/ bb.html) that consists of simple shell scripts to keep track of vital local system resources such as disk usage, CPU load, transfer protocol servers, and so on. We also extended HNMS+ MIB to incorporate the additional monitoring values of BigBrother's status matrix. The HNMS+ master module analyzes a local log file created by BigBrother and fills out the previously mentioned MIB variables.

Furthermore, we developed a Unix daemon that offers the possibility of remote Unix command invocation. ExperNet agents can acquire information that cannot be obtained directly from HNMS+ but only through command execution on the monitored remote hosts (such as the traceroute and tcpdump packet-monitoring utilities). Although we could use the usual rsh Unix command for this purpose, we prefer the above solution because it restricts the set of allowed commands, through appropriate configuration of the module, thus leading to a more flexible and secure system.

References

1. G.A. Jude and L.E. Schecht, "The NAS Hierarchical Network Management System," *Integrated Network Management III*, H.-G. Hegering and Y. Yemini, eds., Elsevier, Netherlands, 1993, pp. 301–312.

2. J. Case et al., *A Simple Network Management Protocol (SNMP)*, tech. report RFC 1157, Network Working Group, 1990; www.cis.ohio-state.edu/cgi-bin/rfc/rfc1157.html (current 4 Sept. 2001).

interface and divides them in two classes:

- HNMS+ MIB-type data, which describe the network topology, the network entities and devices, and their current operational state. This set of objects consists of the internal network representation in HNMS+.
- Standard MIB (SNMP)-type data, which describe in greater detail a specific network device's operational parameters, such as an interface's Maximum Transfer Unit (MTU) or whether the specific host

has IP forwarding capabilities. These are available through the HNMS+ server, through an explicit subscription process.

Each of these data is represented with an appropriate object class in Device. There is a one-to-one correspondence between each class of objects in the HNMS+ MIB definition and a Device class: agent, internet, network, subnet, ipaddr, site, processor, interface, equipment, administrator, service, and module. Each such class has specific slots corresponding to the attributes of the network entity it describes. In addition, there is a superclass to which all other classes belong that defines common attributes for all network objects.

The MIB variables for each network entity (processor, IP address, interface, and so on) are represented as slots of the corresponding network object class. The slot names have the prefix mib_ to distinguish them from the corresponding HNMS+ variables. Figure 3 shows a typical class definition example of the interface HNMS class.

Knowledge base structure

Each basic inference method of the knowledge model maps to a Device module. For example, the top-level detect task (see Table 2) has three subtasks: abstract data, match symptom class, and refine symptom class, which directly maps to corresponding Device modules.

The advantages of such a modular knowledge base are numerous. First, rules are grouped into sets of a specific functionality, thus providing a logical partitioning of the knowledge base, which facilitates the addition of more rules in each module without risking unpredictable rule interactions. Another advantage is that remote invocation of the appropriate modules facilitates the agent model implementation and helps coordination between agents.

Each basic inference method uses a set of inference objects to pass data from one task to the next (see Figure 4). All data are available to all modules; therefore, we don't have to worry about value passing between different tasks. Inference objects belong to one of these classes: symptom, hypothesis, problem, and plan.

Production rule actions create symptom objects during the detect phase and store information about observed abnormal situations in the network. Hypothesis objects represent the initial hypotheses concerning an observed symptom's cause and are created and consumed during the diagnosis phase. Problem objects describe an observed symptom's cause and are generated as an outcome of the diagnosis phase. Plan objects solve diagnosed problems generated during the repair phase. Figure 4 illustrates object generation during an expert system's operation cycle.

Table 1. Types of messages and interactions between ExperNet agents.

Message types	Receiver's intended reaction
Ask for observable	Acquires observable and informs sender
Ask for plan acceptance	Decides about acceptance and informs sender
Ask for plan refinements	Refines plan and 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

Table 2. Mapping of the detect subtasks to Device modules.

Tasks	Modules
abstract_data	#module(abstract_data,_).
match_symptom_class	#module(match_symptom_class,_).
refine_symptom_class	#module(refine_symptom_class,_).

```
new([interface,[
        is_a([genObj]),
        slot(slot_tuple(hnmsObjPhysParent, global, single, optional, plog)),
        slot(slot_tuple(hnmsObjAgent, global, single, optional, plog)),
        slot(slot_tuple(hnmsIfProcIfIndex, global, single, optional, integer)),
        slot(slot_tuple(mib_ifMtu, global, single, optional, integer)),
        slot(slot_tuple(mib_ifInOctets, global, single, optional, float)),
        slot(slot_tuple(mib_ifOutOctets, global, single, optional, float)),
        slot(slot_tuple(mib_ifSpeed, global, single, optional, integer)),
        slot(slot_tuple(mib_ifInDiscards, global, single, optional, float)),
        slot(slot_tuple(mib_ifOutDiscards, global, single, optional, float)),
        slot(slot_tuple(mib_ifInErrors, global, single, optional, float)),
        slot(slot_tuple(mib_ifOutErrors, global, single, optional, float)),
        slot(slot_tuple(mib_ifAdminStatus, global, single, optional, integer)),
        slot(slot_tuple(mib_ifOperStatus, global, single, optional, integer))
]]) => mm_entity_class
```

Figure 3. The interface class definition.

Rule examples

The knowledge necessary for implementing tasks is encoded in rules, which create and manipulate each task's inference objects. For example, the rule in Figure 5 concerns detect-

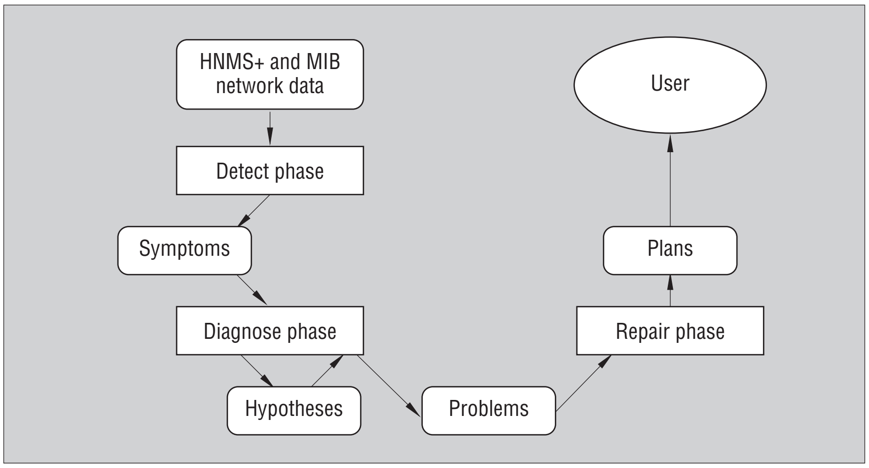


Figure 4. Generation of objects during an expert system's operational cycle.

ing a host's unavailability. It belongs to the detect phase and more particularly to the match symptom class task. The rule's meaning is rather straightforward: "If there is an object of type processor and its hnmsObjReachStatus is three (the mentioned host is unreachable), then create a new symptom object of the class host_is_unreachable." The created object has the appropriate description of the abnormal situation as well as additional information required by later stages of the diagnosis and repair phases. Such information is the name of the target_host, which is the nonresponding host, and the name of the source_host, which is the host that cannot reach the target host.

Another example rule in Figure 6 belongs to the refine symptom class task and refines an already diagnosed symptom that concerns a Web service's unavailability. More specifically, the rule detects whether the symptom is valid by ensuring that the host on which the Web service is located is reachable (if it were not, there should be a symptom of the type host is unreachable for that host) and then determines the Web host machine's relative topology with respect to the source host.

Finally, the rule in Figure 7 is a control rule that switches to module notify-agents-to-repair when there is a symptom and a corresponding diagnosed problem, but we don't know which agent is responsible for solving the problem.

System operation

The ExperNet system's operation consists of two phases: initialization and normal operation.

Initialization phase. This phase consists mainly of initializing the expert system and its connections to the HNMS+ master module and the existing ExperNet agents. During connection initialization, ExperNet initializes the agent's community, connects to a specific HNMS+ master module in the hierarchy, discovers other ExperNet agents in the network (partners), and creates the necessary advertised ports through which the agent communicates.

One problem we encountered was the agent community's initialization. The main issue was how to notify agents about other agents in the network and of the information required for setting up communication, such as their network addresses and available communication ports. The problem becomes harder considering that due to agent distribution over the WAN, members of the agent community might become unreachable due to network failures or might simply halt and restart at any time for various reasons.

We tackled this problem by introducing a new type of object in the HNMS+ MIB through which agents could announce their existence along with any other necessary information. The announcement takes place during the agent's connection to the HNMS+ master module. During normal operation, each agent learns about societal changes through alert events generated by the CS-Prolog-to-HNMS+ interface and then updates its social knowledge accordingly.

As soon as the agent's connection with HNMS+ is established, the agent obtains a list of all network objects, which is a superset of the network for which that agent is responsible. This occurs because in the hierarchical architecture of HNMS+, each module contains not only the objects that it has discovered directly but also the set of all objects that the lower-level modules have discovered. To resolve this, each network object is marked with an indication of the HNMS+ module that originally discovered it. The system then subscribes only to network variables of the objects that belong to the agent's area of responsibility. This information helps create the corresponding Device objects.

ExperNet prompts the user to specify which subnetworks correspond to leased lines between sites. In some cases, the absence of SNMP-manageable modem devices does not let the system determine that information. This final step completes the system's initialization phase.

Normal operation phase. During this phase, the system monitors the network state and reports any abnormal situations to the user through its Web-based interface. ExperNet's main cycle consists of three steps:

1. Receive and interpret messages from HNMS+, which mainly concern changes to the values of HNMS MIB and standard MIB variables of the subscribed

```
#rule(host_unreachable,_,_).
 "if     Target_id@processor(hnmsObjReachStatus=3) and
         Module@module(hnmsModuleHostName:HostName) and
         Source_id@processor(hnmsObjUName=HostName)
 then    new([_,[description(['host is unreachable']),
                 target_host([Target_id]),
                 source_host([Source_id]),
                 resp([unknown])
         ]])=>host_unreachable"
#endrule.
```

Figure 5. Symptom generation.

```
#rule(www_lan_placement,_,_).
 "if     Sym@www_service_is_down(target_host:Target_oid,source_host:Source_oid) and
         not Sym2@host_unreachable(target_host:Target_oid) and
         prolog{relative_topology(Target_oid,Source_oid,Place)}
 then    put_placement([Place]) => Sym,
         put_focus([yes]) => Sym"
#endrule.
```

Figure 6. Refining a symptom class.

```
#rule(switch_to_ntf_agents_to_repair,_,_).
 "if     Sym@symptom(focus=yes) and
         Problem@problem(symptom_oid:Sym, resp=unknown)
 then    switch_task(ntf_agents_to_repair)"
#endrule.
```

Figure 7. A control rule.

objects. During this step, the system modifies the device parameter database, but the rules perform no reasoning.
2. Receive, execute, and answer possible request messages originating from remote agents, invoking the corresponding rule modules.
3. Execute rules based on the newly arrived network data. The system detects abnormalities in the network, performs diagnosis, and reports problems and a list of repair actions to the user.

The normal operation cycle is executed until the user explicitly terminates it by generating an appropriate system interrupt.

ExperNet's output includes all relevant information concerning the detected problem, which it displays on the system's Web page and refreshes in a period of five seconds. When an error occurs in the network, a large, red, blinking indicator informs the network operator. Sometimes, ExperNet cannot detect malfunctions automatically because certain pieces of information are not available to it (such as modem malfunction). In these cases, the system questions the network operator that is physically closer to such hardware devices, and he or she is more likely to answer them. In case the operator cannot answer the question in a specified time limit, ExperNet assumes a default answer, because the system cannot wait for a response to a critical network error indefinitely.

System evaluation and testing

We installed and tested the ExperNet system in an experimental network zone in Ukraine that included one metropolitan (national), three district, and eight regional-level nodes (see Figure 8). There is one ExperNet management node at each metropolitan and district node. The system monitors and manages each node's LAN and WAN subnetwork. ExperNet does not manage the regional-level nodes' LANs because the Ukrainian ISP does not own private networks.

We initially used this experimental installation to test the system's correctness. Later, local network operators received adequate training for using ExperNet in their day-to-day business. Our tests cover typical cases of fault and performance management, such as the following:

- *A router is unreachable*. Either the router interface is down or the process **inetd** that controls the router's network operation does not exist.

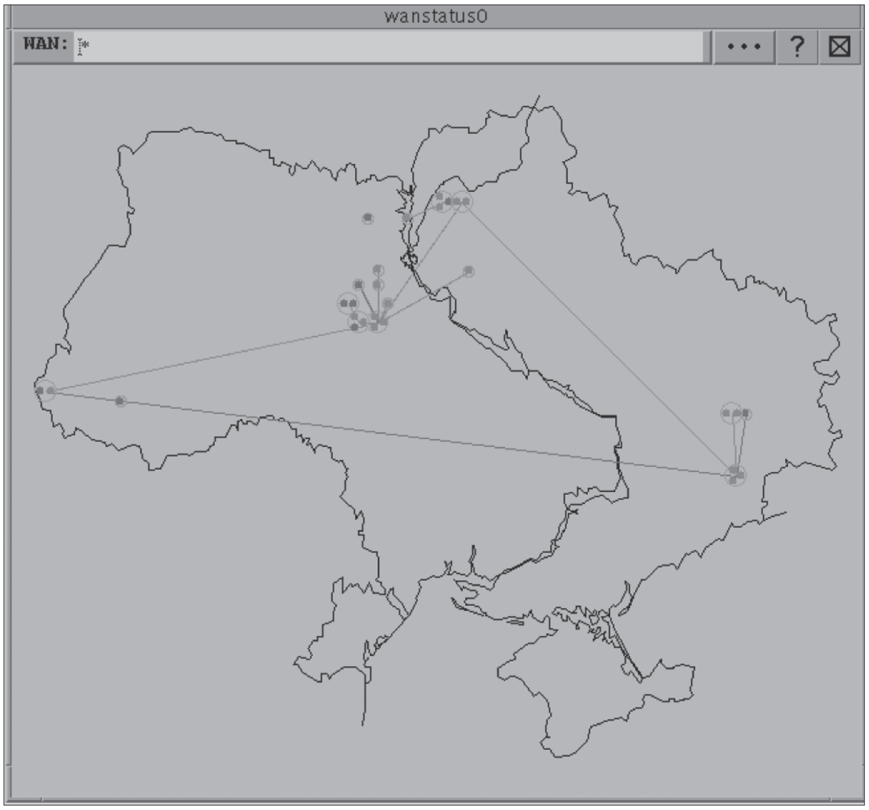


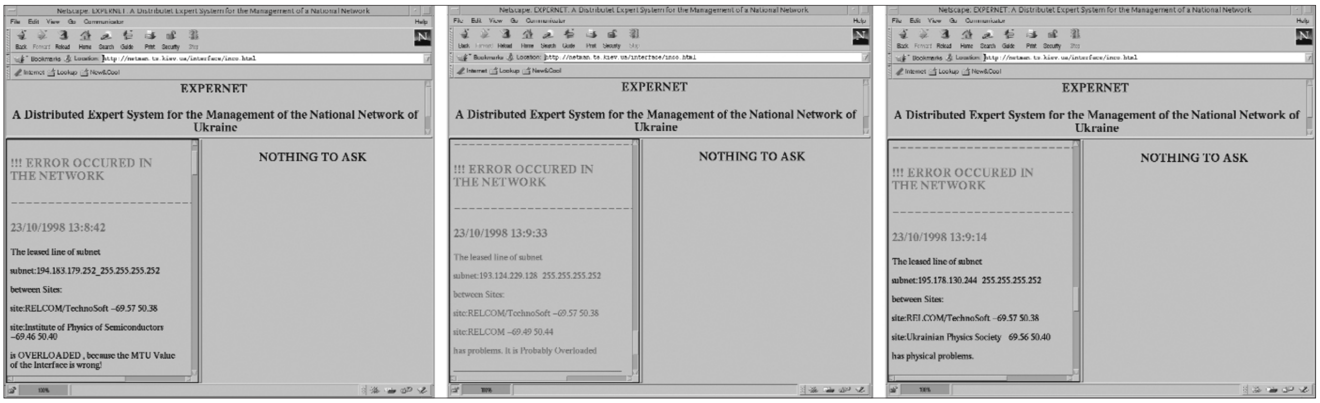
Figure 8. The experimental network zone of ExperNet in Ukraine.

- *A host is unreachable*. An interface problem exists, the **inetd** process is missing from the processor's memory, or there is some other cabling problem.
- *The HTTP or FTP service is not responding*. The **httpd** or **ftpd** process does not exist in the host's memory, the specific host's response time was bad, or the processor was overloaded.
- *Bad leased line*. The TCP/IP connection over a leased line is malfunctioning because there was a problem with the modem devices, there were physical errors on the line, the interface's MTU value was badly configured, or the line was simply overloaded.
- *A modem is not working properly*. The modem connection over a leased line is broken due to a hardware or cabling fault.

We used these malfunctioning cases to evaluate ExperNet's performance during the demonstration that we prepared at the end of the project for all cooperating parties. Due to space limitations, we present here only the most representative cases. Of course, all these errors could not possibly have happened during the demonstration accidentally, so we deliberately caused them by switching off devices or artificially flooding the network connections with packets.

Figure 9a presents a case where the leased line between a district and a regional node was overloaded due to a wrong MTU value. ExperNet suggested the correct values for the MTU to repair the malfunction and recommended decreasing the MTU's value on the malfunctioning leased line. This solution is justified because on such a low-bandwidth line, "long" packets can cause more errors and packet discards than shorter ones. The network operator would continue to get this message (if the line remained overloaded) until he or she manually changed the MTU on both interfaces. ExperNet does not automatically alter critical network parameters because network operators do not feel comfortable with such automation, even if the system gets permission first.

Figure 9b shows a similar malfunction in which the line between two nodes of the second and third levels came down because of physical problems. However simple the diagnosis might seem, it requires reasoning from ExperNet to exclude every other possible cause. The system must first determine if network devices work properly on both sides of the leased line and then that the parameters of corresponding interfaces are correct. After excluding all other possible problem causes, the system computes the percentage of packets discarded and those rejected due to errors over the number of total transmitted packets. A number over 15 percent indicates that the


Netscape: EXPERNET: A Distributet Expert System for the Management of a National Network
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://netman.ts.kiev.ua/interface/inco.html
Internet Lookup New&Cool
100%

EXPERNET

A Distributed Expert System for the Management of the National Network of Ukraine

(a)

!!! ERROR OCCURED IN THE NETWORK

23/10/1998 13:8:42

The leased line of subnet

subnet:194.183.179.252_255.255.255.252

between Sites:

site:RELCOM/TechnoSoft –69.57 50.38

site:Institute of Physics of Semiconductors –69.46 50.40

is OVERLOADED , because the MTU Value of the Interface is wrong!

NOTHING TO ASK

(b)

!!! ERROR OCCURED IN THE NETWORK

23/10/1998 13:9:33

The leased line of subnet

subnet:193.124.229.128 255.255.255.252

between Sites:

site:RELCOM/TechnoSoft –69.57 50.38

site:RELCOM –69.49 50.44

has problems. It is Probably Overloaded

NOTHING TO ASK

(c)

!!! ERROR OCCURED IN THE NETWORK

23/10/1998 13:9:14

The leased line of subnet

subnet:195.178.130.244 255.255.255.252

between Sites:

site:RELCOM/TechnoSoft –69.57 50.38

site:Ukrainian Physics Society 69.56 50.40

has physical problems.

NOTHING TO ASK

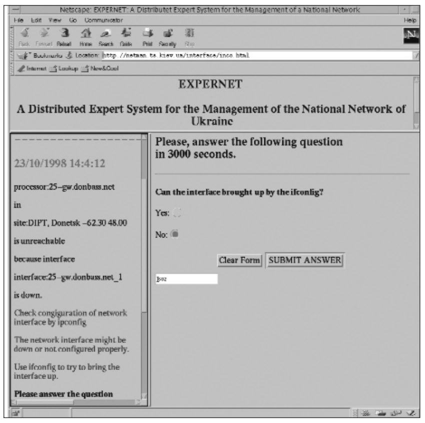
Figure 9. Various leased line problems: (a) an overloaded line due to wrong maximum transfer unit, (b) a bad physical line, and (c) an overloaded line due to high traffic.

most probable problem is a physical error on the line. Only an experienced network operator could follow such reasoning, spending a lot of time to check all these values.

A similar case concerned an overloaded leased line between two sites on a node of the same level. In this case, agents on each side had partial information about the problem—communication between them occurred through the exchange of messages about the problem. The messages consisted of inquiries about the interfaces' status and operational parameters. Having determined that the problems did not lie anywhere else but in the MTU value, the agents notified network operators on both sides with a message similar to the one in Figure 9b about the problem's cause and remedy.

Figure 9c shows another case of problematic leased line operation. The corresponding ExperNet specialist has classified this case as a performance management problem and not a hardware fault. The differentiation is based on the MTU value, which was valid, and the sum of errors and discards, which were less than 15 percent. Such a differentiation is difficult for a network operator to achieve, because it requires complex reasoning during busy periods of the day.

A common case occurs when a remote machine that hosts important networking services is unreachable (see Figure 10). In this scenario, ExperNet detected which processor and interface was unreachable and asked the operator whether the remote interface could be brought up by using the **ifconfig** command. ExperNet could not perform this test automatically through remote command invocation, because local administrators have strong objections concerning security. So, the second-level network operator called the remote host operator on the phone—he answered that the interface could not be brought up. ExperNet asked again whether there were any problems with the interface's MTU or any other parameters. The remote operator answered negatively again, and ExperNet concluded that hardware or configuration fault might be involved, which was true because the remote host was switched off. Network operators could not have detected and repaired this problem manually until the service's users complained (by phone or email). Fixing abnormal situations like this leads to an increased availability of services and, consequently, more satisfied clients.


Netscape: EXPERNET: A Distributet Expert System for the Management of a National Network
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://netman.ts.kiev.ua/interface/inco.html
Internet Lookup New&Cool

EXPERNET

A Distributed Expert System for the Management of the National Network of Ukraine

23/10/1998 14:4:12

processor:25–gw.donbass.net

in

site:DIPT, Donetsk –62.30 48.00

is unreachable

because interface

interface:25–gw.donbass.net_1

is down.

Check congiguration of network interface by ipconfig

The network interface might be down or not configured properly.

Use ifconfig to try to bring the interface up.

Please answer the question

Please, answer the following question in 3000 seconds.

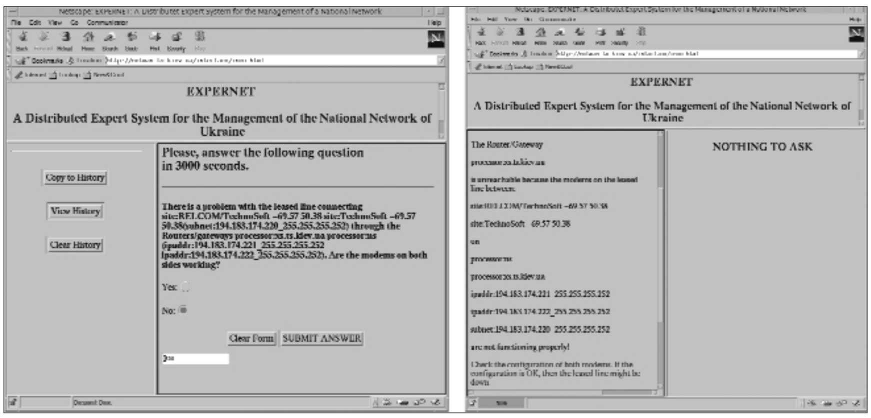
Can the interface brought up by the ifconfig?

Yes: [✓ No:]

Clear Form | SUBMIT ANSWER

Figure 10. Remote host is unreachable.

Figure 11 shows a more complicated case that ExperNet successfully handled. The system detected that all remote hosts were unreachable and thus the leased line connecting the remote hosts had a problem, not the hosts themselves. Consequently, it asked the user if the modems on both sides of the line worked (ExperNet could not automatically obtain this information because the modems were not SNMP-compliant). In this test case, the local modem was switched off, so the network operator replied negatively. The system suggested a problem with either modem configuration or the leased line itself. In this case, ExperNet's contribution lies in the fact that the unreachability of multiple hosts has been aggregated (alarm correlation) as a single line's problem.


Netscape: EXPERNET: A Distributet Expert System for the Management of a National Network
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location
Internet Lookup New&Cool
Copy to History
View History
Clear History
Document: Done.

EXPERNET

A Distributed Expert System for the Management of the National Network of Ukraine

Please, answer the following question in 3000 seconds.

There is a problem with the leased line connecting site:RELCOM/TechnoSoft –69.57 50.38 site:TechnoSoft –69.57 50.38(subnet:194.183.174.220_255.255.255.252) through the Routers/gateways processor:xs.ts.kiev.ua processor:xs (ipaddr:194.183.174.221_255.255.255.252 ipaddr:194.183.174.222_255.255.255.252). Are the modems on both sides working?

Yes: [✓ No:]

Clear Form | SUBMIT ANSWER

The Router/Gateway

processor:xs.ts.kiev.ua

is unreachable because the modems on the leased line between:

site:RELCOM/TechnoSoft –69.57 50.38

site:TechnoSoft 69.57 50.38

on

processor:xs

processor:xs.ts.kiev.ua

ipaddr:194.183.174.221 255.255.255.252

ipaddr:194.183.174.222_255.255.255.252

subnet:194.183.174.220 255.255.255.252

are not functioning properly!

Check the configuration of both modems. If the configuration is OK, then the leased line might be down

NOTHING TO ASK

Figure 11. ExperNet reporting modem failure and suggesting actions.

Practical experience

We developed the system under an EU-funded project's strict time limitations. During development, we encountered several difficulties. First, the knowledge acquisition procedure for building such a large and complex system requires a lot of time and resources, more than were originally available. Additionally, our network experts lived in a different country (Ukraine) from the knowledge engineers (Spain and Greece). This limited our communication to questionnaires exchanged over the Internet and a few face-to-face interviews. Moreover, we had no experienced HNMS+-monitoring system users among the network operators involved in the knowledge acquisition phase. If we had such operators, they could have delivered expert knowledge and everyday rules of thumb concerning network management to our knowledge engineers. Developing and integrating the various subsystems in a robust and efficient product was a time-consuming task because we had to

- extend the original HNMS system to provide a true hierarchical structure and fix various bugs;
- implement an interface between CS-Prolog II and the new HNMS+ system;
- develop an additional interface between the intelligent agents and the HNMS+ system based on CS-Prolog II's networking primitives to adequately receive and translate networking data; and
- debug the various subsystems and the interfaces between them.

Our initial system design did not include interaction with the network operators; ExperNet had to perform all detection, diagnosis, and repair actions automatically. Several unexpected problems came up that changed our plans to include questioning the operators. The most important problems for full automation proved to be

- technology in the experimental installation zone included non–SNMP-compliant devices that the system could not automatically monitor;
- single-line connections between network nodes caused the complete unreachability of hosts should modems, lines, or routers break down; and
- the fragmentation of the network's ownership caused responsibility conflicts and security objections.

We encountered another major problem when we installed ExperNet in several nodes of the Ukrainian network. Although network administrators showed interest in ExperNet's network monitoring and repairing facilities, the practical application of ExperNet, HNMS+, and BigBrother required

- significant additional computer resources;
- additional negotiations with network administrators concerning data security; and
- allocation and additional training of personnel for operating the network-monitoring software (HNMS+/BB).

The explosive growth in demand for networking in the last decade has increased the need for advanced management software that offers intelligent administration services. Our multiagent intelligent system can significantly decrease the downtime of network components, thus leading to an increased availability of the overall network.

ExperNet is based on existing and widely used management protocols (SNMPv2), which makes its application to any existing network possible. We installed and tested the system in a real network environment, and it has performed well. It is now working on two operating systems—Solaris 2.5 and FreeBSD 2.2.6—and we plan to extend its area of application.

The most significant extension that could add to its current implementation concerns the knowledge base itself. New management cases should be added to cover the full range of management areas, including fault, performance, configuration, security, and accounting management. Adding these new cases would require minor modifications (if any) to the existing system. Additionally, a number of vendor-specific knowledge bases could help exploit each network device's management characteristics. This modular approach will increase the present system's mobility and flexibility.

More explanation facilities could increase the user's trust in the system and provide a platform for tutoring resolution methods for network management problems. In addition, HNMS+ could be extended to cooperate with SNMPv3 agents, but such an extension would require modifications to the monitoring tool's core. These modifications should be rather simple because the system structure is modular, and changes would not affect other parts of the system.

We are currently developing an integrated Web-based user interface (based on PHP and XML technologies) that will host both ExperNet and HNMS+ data and messages. This will allow remote monitoring of network status so that the network operator does not have to physically be on the same site as the visual HNMS+ user interface. ■

Acknowledgments

The EU INCO-Copernicus project, *ExperNet: A Distributed Expert System for the Management of a National Network*, No. 960114, funded the work described in this article (see www.csd.auth.gr/~lpis/projects/inco/lnco.html).

References

1. I. Vlahavas et al., "System Architecture of a Distributed Expert System for the Management of a National Data Network," *Proc. 8th Int'l Conf. Artificial Intelligence*, Springer, New York, 1998, pp. 438–451.
2. J. Cuena and S. Ossowski, "Distributed Models for Decision Support," *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, AAAI/MIT Press, Cambridge, Mass., 1999, pp. 459–504.
3. N. Bassiliades, I. Vlahavas, and A.K. Elmagarmid, "E-DEVICE: An Extensible Active Knowledge Base System with Multiple Rule Type Support," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 5, Sept./Oct. 2000, pp. 824–844.
4. M. Molina and S. Ossowski, "Knowledge Modeling in Multiagent Systems: The Case of the Management of a National Network," *Intelligence in Services and Networks, Paving the Way for an Open Service Market*, H. Zuidweg et al., eds., Springer, New York, 1999, pp. 501–513.
5. B.J. Wielinga, A.T. Schreiber, and J.A. Breuker, "KADS: A Modeling Approach to Knowledge Engineering," *Readings in Knowledge Acquisition and Learning*, B. Buchanan and D. Wilkins, eds., Morgan Kaufmann, San Francisco, 1992, pp. 92–116.
6. J. Cuena and M. Molina, "KSM: An Environment for Knowledge Oriented Design of Applications Using Structured Knowledge Architectures," *Applications and Impacts: Information Processing 94*, K. Brunnstein and E. Raubold, eds., Elsevier, Netherlands, 1994, pp. 143–148; www.isys.dia.fi.upm.es/ksm (current 23 Aug. 2001).
7. B. Chandrasekaran, T. Johnson, and J. Smith, "Task-Structure Analysis for Knowledge Modeling," *Comm. ACM*, vol. 35. no. 9, Sept. 1992, pp. 124–137.
8. D. Brown and B. Chandrasekaran, *Design Problem-Solving: Knowledge Structures and Control Strategies*, Morgan Kaufmann, San Francisco, 1989.
9. M. Barbuceanu and S. Fox, "COOL: A Language for Describing Coordination in Multi-Agent Systems," *Proc. Int'l Conf. Multiagent Systems*, AAAI/MIT Press, Cambridge, Mass., 1995, pp. 17–24.
10. H.-J. Müller, "Negotiation Principles," *Foundations of Distributed Artificial Intelligence*, O'Hare and Jennings, eds., Wiley, New York, 1996, pp. 211–225.

The Authors

Ioannis Vlahavas is an associate professor in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. His research interests include logic programming, knowledge base systems, and AI applications. He received a PhD in computer science (logic programming machines) from the Aristotle University. He is a member of the Greek Physics and Computer societies, a member of the IEEE, and a member of the Association for Logic Programming. Contact him at Dept. of Informatics, Aristotle Univ. of Thessaloniki, 54006 Thessaloniki, Greece; vlahavas@csd.auth.gr; vlahavas@csd.auth.gr.

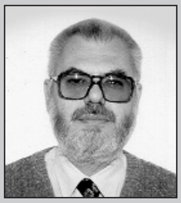
Nick Bassiliades is a part-time lecturer in the Department of Informatics, Aristotle University of Thessaloniki. His research interests include deductive object-oriented databases, active databases, knowledge base systems, parallel database systems, and Web databases. He received a BS in physics from the Aristotle University of Thessaloniki, an MS in applied artificial intelligence from the University of Aberdeen, Scotland, and a PhD in parallel knowledge base systems from Aristotle University. He is a member of the Greek Physics, Computer, and Artificial Intelligence societies; the IEEE; and the ACM. Contact him at Dept. of Informatics, Aristotle Univ. of Thessaloniki, 54006 Thessaloniki, Greece; nbassili@csd.auth.gr.

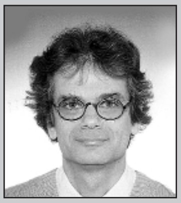
Ilias Sakellariou is currently pursuing his PhD in distributed constraint logic programming in the Department of Informatics at Aristotle University. His research interests include expert systems, planning and scheduling, and parallel and constraint logic programming. He received a BS in physics from the Aristotle University of Thessaloniki and an MS in knowledge based systems from the University of Edinburgh, Scotland. He is a member of the Greek Physics Society, the IEEE, and the ACM. Contact him at Dept. of Informatics, Aristotle Univ. of Thessaloniki, 54006 Thessaloniki, Greece; iliass@csd.auth.gr.

Martin Molina is an associate professor in the Department of Artificial Intelligence, Technical University of Madrid, Spain. He also leads a research group in the Department of Artificial Intelligence. His research interests include knowledge-engineering methodologies, problem-solving methods, multiagent architectures, and intelligent Web-based applications. He received a BSc and PhD from the Technical University of Madrid, Spain. Contact him at the Dept. of Artificial Intelligence, Faculty of Computer Science, Technical Univ. of Madrid, Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid, Spain; mmolina@fi.upm.es.

Sascha Ossowski is an associate professor at the University Rey Juan Carlos. His research interests include intelligent agents and multiagent systems. He received an MSc in informatics from the University of Oldenburg, Germany and a PhD in informatics from the Technical University of Madrid. Contact him at E.S.C.E.T., Univ. Rey Juan Carlos, Campus de Mostoles, Calle Tulipan s/n, Edificio Departamental I, E-28933 Mostoles, Madrid, Spain; s.ossowski@escet.urjc.es.

Iván Futó is a professor at the Budapest University of Economics and State Administration and the IT vice president of the Hungarian Tax Control Administration. His research interests include simulation, logic programming, and multiprocessor systems. He received a BS in electrical engineering from the Technical University of Budapest. He has also received the Award of the Hungarian Academy of Sciences, the Hungarian State Award, and the Award of the US Society for Computer Simulation. Contact him at ML Consulting and Computing Ltd, Frankel Leó út 45, Budapest 1023, Hungary; futo@ml-cons.hu.


Zoltán Pásztor works at ML Consulting and Computing in Budapest. He received an MS in electrical engineering from the Moscow Institute of Chemical Engineering and an MS in applied mathematics from Eötvös Loránd University, Budapest. Contact him at ML Consulting and Computing Ltd, Frankel Leó út 45, Budapest 1023, Hungary; pasztor@ml-cons.hu.


János Szeredi works at ML Consulting and Computing in Budapest. He received an MS in mathematics from Eötvös Loránd University, Budapest. Contact him at ML Consulting and Computing Ltd, Frankel Leó út 45, Budapest 1023, Hungary; szeredi@ml-cons.hu.

Igor Velbitskiyi is the general director of Technosoft. His research interests include software engineering, network management, intelligent systems, and visual programming. He received a PhD in physical and mathematical sciences from the Institute of Cybernetics of the National Academy of Sciences of Ukraine, Kiev. Contact him at Technosoft, 4, Glushkov Ave., Kiev, 03680, Ukraine; vel@technosoft.kiev.ua.

Sergey Yershov works at Technosoft and is an assistant professor in the Ukrainian Institute of Statistics. His research interests include software engineering methods, intelligent network management, electronic business, and e-Government. He received an MS in applied mathematics from the Kiev Technical University and a PhD from the Institute of Cybernetics of the National Academy of Sciences of Ukraine, Kiev. Contact him at Technosoft, 4, Glushkov Ave., Kiev, 03680, Ukraine; yershov@technosoft.kiev.ua.

Igor Netesin is a deputy director of Technosoft. His research interests include artificial intelligence, network management, and software engineering for communication networks. He received an MS in mathematics from the Moscow State University and a PhD from the Institute of Cybernetics of the National Academy of Sciences of Ukraine, Kiev. Contact him at Technosoft, 4, Glushkov Ave., Kiev, 03680, Ukraine; netesin@technosoft.kiev.ua.