ExperNet: An Intelligent Multi-Agent System for WAN Management*

** Ioannis Vlahavas, Nick Bassiliades, Ilias Sakellariou
{vlahavas, nbassili, iliass}@csd.auth.gr
Department of Informatics, Aristotle University of Thessaloniki,
54006 Thessaloniki Greece.

Martin Molina
mmolina@isys.dia.fi.upm.es
Department of Artificial Intelligence
Technical University of Madrid
28660 Boadilla del Monte, Madrid, Spain.

Sascha Ossowski
s.ossowski@escet.urjc.es
Escuela Sup. de Ciencias Exp. y Tecnología
Universidad Rey Juan Carlos
C\ Tulipán, s/n, 28933, Móstoles, Madrid, Spain.

Ivan Futo, Zoltan Pasztor, Janos Szeredi
{futo, pasztor, szeredi}@ml-cons.hu
ML Consulting and Computing Ltd, ML Kft, H-1011 Budapest,
Gyorskocsi u. 5-7., Hungary.

Igor Velbitskiyi, Sergey Yershov, Igor Netesin
{vel, yershov, netesin}@netman.ts.kiev.ua
International Software Technology Research Center Technosoft,
44 acad. Glushkov Avenue, Kiev, 252187, Ukraine.

Abstract

This paper describes ExperNet, an intelligent multi-agent system that was developed under an EU funded project to assist in the management of a large-scale data network. ExperNet assists network operators at various nodes of a WAN to detect and diagnose hardware failures and network traffic problems and suggests the most feasible solution, through a web-based interface. ExperNet is composed by intelligent agents, capable of both local problem solving and social interaction among them for coordinating problem diagnosis and repair. The current network state is captured and maintained by conventional network management and monitoring software components, which have been smoothly integrated into the system through sophisticated information exchange interfaces. For the implementation of the agents, a distributed Prolog system enhanced with networking facilities was developed. The agents' knowledge base is developed in an extensible and reactive knowledge base system capable of handling multiple types of knowledge representation. ExperNet has been developed, installed and tested successfully in an experimental network zone of Ukraine.

Keywords: Distributed Artificial Intelligence; Expert System; Intelligent Agents; Network Management; Active Knowledge Base System; Distributed Prolog

---

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

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

Introduction

The management of large data networks, such as a national WAN, is without doubt a complex and cumbersome task. Existing network management software cannot meet the requirements of the constantly increasing size and complexity of today's TCP/IP based networks, since in most cases it provides only simple monitoring tools. Therefore, the full exploitation of a WAN cannot easily be achieved without user-friendly, intelligent network management software, enhanced with diagnostic and decision support services.

Expert systems seem to provide a feasible but also an elegant solution to this direction. Currently the development of expert systems for WAN management is only at research and experimental stage. The formalization of such a task is difficult because information about network state is most of the times inadequate and incomplete, there is a large scale of behavior characteristics, and the network environment continuously evolves. The absence of practical and verified expert systems for large, complex modern technological systems, such as WANs, demonstrates the complexity of the task and poses a great challenge ahead.

An important point in the design of such a system[1] is that there must exist an efficient network-monitoring schema, i.e. determining the network state and capturing important network events. The implementation of such functionality has to be in accordance with the existing network monitoring technology; namely, the Management Information Base (MIB) and the Simple Network Management Protocol (SNMP), in order to have control over the existing real-world network devices. The MIB is the set of variables needed for monitoring and control the various components of a TCP/IP based network, i.e. the information required to manage the network elements. The SNMP is a protocol that defines how this information can be accessed and modified remotely.

Finally, network management requires coordination among operators of autonomous network nodes. In order to closely resemble the distributed nature of network management, multi-agent problem solving techniques[2] need to be employed to model the functionality of individual agents as well as the interactions that take place among them. Multi-agent technology is preferable to other distributed problem solving techniques since it offers enhanced modularity, reactiveness to environmental changes and reusability.

This article presents the architecture, implementation, operation and evaluation of the ExperNet system, a multi-agent expert system that we have developed for Ukraine's National TCP/IP WAN, under the framework of a joint EU funded project. ExperNet assists network operators at various nodes of a WAN to detect and diagnose network failures and traffic problems and suggests the most feasible solution, given resource and temporal constraints. In order to achieve the above functionality, ExperNet features the following:

- Monitoring tools for capturing the network state;
- A platform for logic-based applications that supports real-time communication between system components;
- An efficient and extensible expert system shell that offers multiple types of knowledge representation and is able to cope with the constantly changing network environment, through event-based programming (active rules);
- Fast, heuristic detection, diagnosis and repair of network failures;
- Co-operative problem solving strategies;
- A user-friendly web-based interface.

The ExperNet System Architecture

ExperNet is a multi-agent system since network management is inherently distributed. It consists of a hierarchically structured architecture, where each level of the hierarchy consists of one or more *management nodes* (**Figure 1**). Each node encapsulates one or more management subnodes of the lower level and is responsible for the management of a network area, assisted by a local *intelligent agent*. The hierarchical structure of the system is in accordance with the structure of the pre-existing organization of the experimental zone of the Ukrainian national network that is divided in regional, district and metropolitan-area subnetworks.

ExperNet assists network operators at various nodes of the Ukraine's national network to detect and diagnose network failure and traffic problems. Furthermore, it suggests the most feasible solution, given resource and temporal constraints. To achieve this goal, each agent is able to communicate with other agents on the network in order to collaborate towards problem diagnosis and repair, using social knowledge for co-ordination. Agents acquire network data that concern device installation, removal, reachability and operational parameters from conventional network management software.

We managed to put many different pieces of software to work together by extending the distributed Prolog system CS-Prolog II (see sidebar) with networking facilities and building sophisticated information exchange interfaces between each agent and its local network management/monitoring software components. Furthermore, we have developed the HNMS+ system by extending the HNMS network management software, and integrating the BigBrother monitoring tool for local computer resources (see sidebar), so that HNMS+ can dispatch to its local intelligent agent the desired information. Finally, we have built the DEVICE knowledge base system (see sidebar) on top of CS-Prolog II in order to provide a flexible, high-level expert system shell on which we have implemented the intelligent agents. The agents' knowledge base consists of both 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 the necessary information about the state of the network. Additional local computer resource information is provided by *Big Brother*. The agents themselves are developed in DEVICE, while communication facilities are provided by

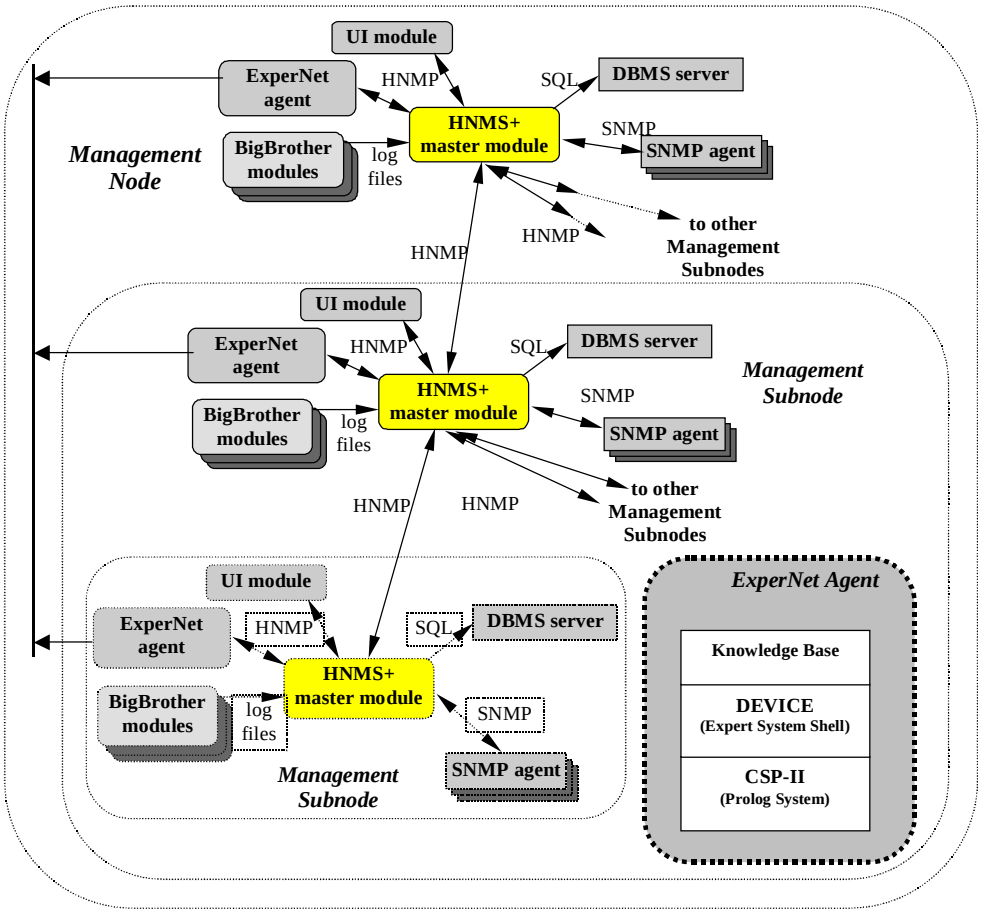

Figure 1. ExperNet system architecture.

CS-Prolog II.

Knowledge Model for ExperNet Agents

Each ExperNet agent is comprised of two types of knowledge:[3] local knowledge for individual problem-solving, i.e. local network management, and social knowledge for co-ordination, i.e. harmonization of local network management with the activities of acquaintance nodes. The above types of knowledge are utilized during the problem-solving cycle of an ExperNet agent (**Table 1**).

Local Problem-solving

The particular characteristics of the network management domain, which have been identified during the knowledge acquisition phase, include complex problem-solving tasks (classification, diagnosis, planning, etc.), naturally leading to model-based system development.[4,5] We have modeled the agents' problem-solving competence as a three-step process (**Figure 2**) and each step consists of customized generic knowledge modeling methods.[4]

Initially, during *symptom detection*, the system watches out for symptoms of undesired network states and behaviors (e.g. a certain service, such as FTP, WWW, etc., does not respond, a host is unreachable, over/under-utilization of links or equipment, etc.). The symptom detection phase in ExperNet is tackled as a *heuristic classification* task[6], based on the data-driven, reactive nature of ExperNet's operation. It follows three steps (*abstraction*, *matching* and *refinement*) which, in our model, are supported by two types of knowledge: knowledge about the network model for abstraction and refinement that includes a declarative representation of the network structure, and knowledge that uses a set of problem scenarios relating symptoms and observables.

Table 1. Problem solving cycle of ExperNet agents.

```
1.    Detect symptoms.
2.    Inform agents interested in the symptoms, 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, to isolate them.
5.    Inform agents interested in problems, to repair them.
6.    Generate a repair plan (if the agent is responsible).
            If necessary, ask agents for plan acceptance
```

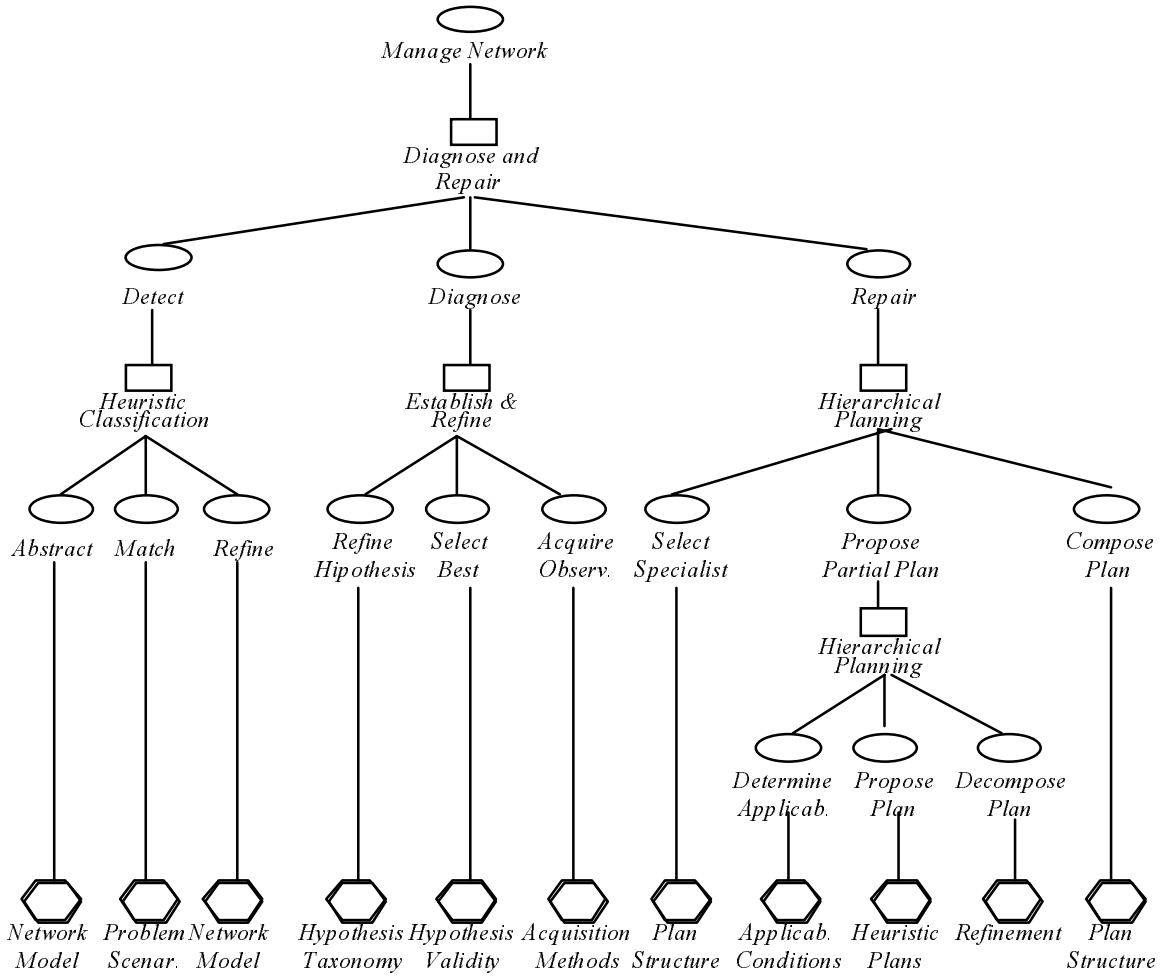

Figure 2. ExperNet agents' local problem solving.

Secondly, *diagnosis* is performed by discriminating hypotheses of different degrees of precision based on 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.) There are numerous network components, which in turn can malfunction in many possible ways; therefore, there are a huge number of possible diagnoses for ExperNet. Since speed is a crucial issue for the acceptable operation of ExperNet, we have chosen the *establish and refine* method[7] for diagnosis, in order to quickly focus on network malfunctions without using deep models of the network components. This method can be conceived as an abstract reasoning pattern based on a heuristic search in a taxonomy of hypotheses of problems and has been adapted to make use of three primitive inference steps:

- *Refine problem hypotheses*, which uses a knowledge base represented by a taxonomy of hypothesis classes related through the *is-a* relation;

- *Select best hypothesis*, which makes use of knowledge about the validity of hypotheses (represented by frames) to establish whether any of the input hypotheses can be proven;
- *Acquire additional observables*, which determines the sequence of exploratory actions to get additional observables by using a knowledge base about acquisition methods (represented by rules).

Finally, *repair* is performed, where a sequence of repair actions is proposed to solve the problem. Network problems may be complex to repair and they may require the co-ordination of skills from different specialists. This leads naturally to *hierarchical planning* 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 partial abstract plans, dynamically composed during the reasoning process.[8]

Social Co-ordination

An important fragment of a node administrator's time is not spent at local problem solving but coordinating his/her work with other administrators. *ExperNet* requires co-ordination in three types of situations, which have been identified during the knowledge acquisition phase:

- *Information acquisition*, when additional observations are needed. These observations are available within the agent society, but they are not accessible by the node itself.
- *Responsibility conflicts*, when different agents intend to perform similar tasks.
- *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*,[9] 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 have used 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 an inquiring agent asks some target agent for a certain observable or plan; the latter may reply either 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 for a proposed plan to be accepted.

Table 2. Types of messages and interactions between ExperNet agents.

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

Interactions within a conversation are based on a message-passing model, to closely reflect the cooperation model of human network operators. Every message that is exchanged during such interactions can be considered as a *Speech Act*, since the sender wants to influence the behavior of the receiver by emitting it.[10] **Table 2** 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 behavior options, e.g. 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. Thus, an agent maintains *agent models* of all acquaintances that it interacts with, including itself.[2]

ExperNet Implementation

The implementation of the knowledge model of ExperNet is based on the DEVICE knowledge base system, which was used not only as an expert system shell for developing the knowledge base itself (rules, facts and objects-frames), but also as an integrator for network information and agent communication. The ability of DEVICE to handle large collections of data, such as the status of network devices of a WAN, is important for the development of the ExperNet system. The object-oriented architecture and data types of DEVICE 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. For the needs of ExperNet, DEVICE was re-implemented in CS-Prolog-II, a language that, 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 of agent-based systems can be easily implemented.

Capturing Network Data

The data concerning the network entities and their operational parameters are acquired by DEVICE from the HNMS+ system via the DEVICE-HNMS+ interface and are divided in two classes:

- *HNMS+ MIB* type data that describe the network topology, the network entities and devices, and their current operational state. This set of objects consists the internal network representation in HNMS+.
- *Standard MIB (SNMP)* type data that describe in greater detail the operational parameters of a specific network device, e.g. the Maximum Transfer Unit (MTU) of an interface or whether the specific host has IP forwarding capabilities or not, etc. These are available through the HNMS+ server, through an explicit subscription process.

Each of the above mentioned data is represented with an appropriate object class in DEVICE. There is an 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 that describes. In addition, there is a superclass to which all other classes belong to, which defines common attributes for all network objects.

The MIB variables for each network entity (processor, IP address, interface, etc.) are represented as slots of the corresponding network object class. The slot names have the prefix "`mib_`" to be distinguishable from

Table 3. The `interface` class definition.

```
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
```

Table 4. 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,_).`

the corresponding HNMS+ variables. A typical class definition example of the *interface* HNMS class is shown in **Table 3**.

Structure of the Knowledge Base

Each basic inference method of the knowledge model is mapped to a DEVICE module. For example, the top level *Detect* task (**Table 4**) has three subtasks: *"abstract data"*, *"match symptom class"* and *"refine symptom class"*, which are directly mapped to corresponding DEVICE modules.

The advantages of such a modular knowledge base are numerous. First, rules are grouped into sets of a specific functionality and thus a logical partitioning of the knowledge base is achieved, which facilitates the addition of more rules in each module, without bearing the risk of running into unpredictable rule interactions. Another advantage is that the implementation of the agent model is facilitated and that co-ordination between agents is accomplished by remote invocation of the appropriate modules.

Each basic inference method uses a set of *inference* objects to pass data from one task to the other (**Figure 3**). All data are available to all modules; therefore, no consideration has to be taken concerning value-passing between different tasks. Inference objects belong to one of the classes: *symptom, hypothesis, problem*

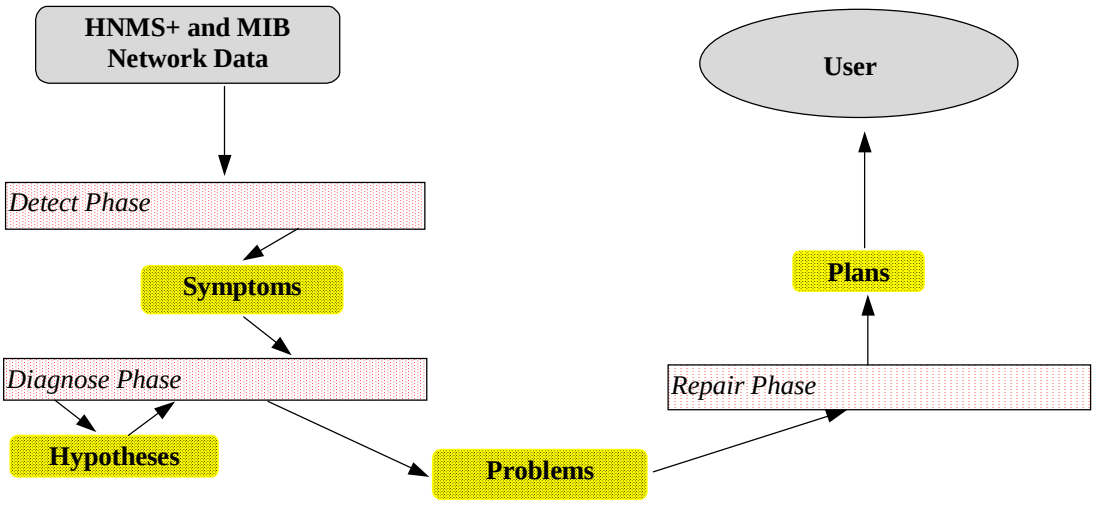

Figure 3. Generation of objects during an expert system's operational cycle.

and *plan.*

Symptom objects are created during the *detect phase* and store information about observed abnormal situations in the network. *Hypothesis* objects represent the initial hypotheses concerning the cause of an observed symptom and are created and "consumed" during the *diagnose phase*. *Problem* objects describe the cause for an observed symptom and are generated as an outcome of the *diagnose phase*, too. Finally, *plan* objects are solutions to a diagnosed problem, generated during the *repair phase*. **Figure 3** illustrates object generation during an operation cycle of the expert system.

Table 5. Symptom generation.

```
#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.
```

Rule Examples

The knowledge necessary for the implementation of the tasks is encoded in rules, which are responsible for creating and manipulating the inference objects of each task. For example, the rule in **Table 5** concerns the detection of the unavailability of a host. It belongs to the *detect phase* and more particularly to the "*match symptom class"* task. The meaning of the rule is rather straightforward: “If there is an object of type *processor* and its *hnmsObjReachStatus* is 3 (i.e. 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 host that is not responding, and the name of the *source_host*, which is the host that cannot reach the target host.

Table 6. Refining a symptom class.

```
#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.
```

Another example rule in **Table 6** belongs to the "*refine symptom class*" task and refines an already diagnosed symptom that concerns the unavailability of a WWW service. More specifically, the rule detects whether the symptom is valid by ensuring that the host on which the WWW 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 relative topology of the WWW host machine with respect to the source host.

Table 7. A control rule.

```
#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.
```

Finally, the rule in **Table 7** is a control rule that switches to module *notify-agents-to-repair*, when there is a symptom and a corresponding diagnosed problem but it has not been established yet whose agent's responsibility is to solve the problem.

System Operation

The operation of the ExperNet system consists of two phases: the *initialization phase* and the *normal operation.*

Initialization Phase

The initialization phase consists mainly of the initialization of the expert system itself 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 of the problems that were encountered was the initialization of the agent community. The main issue was how the agents will be notified of the existence of other agents in the network and of the information

required for setting up communication, i.e. their network address and their available ports for communication. The problem becomes harder considering that due to the distribution of the agents over the WAN, members of the agent community might become unreachable due to network failures, or even simply halt and restart at any time for various reasons.

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

As soon as the connection of the agent with HNMS+ is established, a list of all network objects is obtained, which is a superset of the network the specific agent is responsible for. 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 were discovered by lower level modules. 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 is used to create the corresponding DEVICE objects.

Finally, ExperNet prompts the user to specify which subnetworks correspond to leased lines between sites. This is necessary since in some cases the absence of SNMP manageable modem devices, does not allow the system to determine the above information. This final step completes the initialization phase of the system.

Normal Operation Phase

During *normal operation,* the system monitors the network state and reports to the user any abnormal situations through its web-based interface. The main cycle of ExperNet consists of three steps:

- The first step receives and interprets the messages from HNMS+, which mainly concern changes to the values of HNMS MIB and standard MIB variables of the subscribed objects. During this step, the database of the device parameters is modified but no reasoning is performed.
- During the second step, possible request messages originating from remote agents are received, executed and answered, invoking the corresponding rule modules.

- The final and most crucial step concerns rule execution based on the newly arrived network data. The system detects abnormalities in the network, performs diagnosis and reports to the user problems and a list of repair actions.

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

The output of ExperNet that includes all relevant information concerning the detected problem is displayed on the system's Web page, which is refreshed in a period of 5 seconds. When an error occurs in the network, a large red blinking indication informs the network operator. Following, there is a description of the error, information about the location of the malfunction, and necessary repair actions. Some times the detection of the malfunction cannot be performed automatically because certain pieces of information are not available to ExperNet (e.g. malfunction of modems that are not manageable through the SNMP protocol). In these cases, questions are asked to the network operator that is more likely to answer the question because he/she is physically closer to such hardware devices. In case the question is not answered in a specified time limit, a default answer is assumed, since the system cannot wait for an answer indefinitely to cater for a critical network error. Examples of the system's interface are given in the next section.

System Evaluation and Testing

The ExperNet system has been installed and tested in an experimental network zone in Ukraine that includes one metropolitan (national) level node, three district and eight regional level nodes (**Figure 4**). There is one ExperNet management node at each metropolitan and district nodes. ExperNet monitors and manages each node's LAN and WAN subnetwork. The LANs of the regional level nodes are not managed by ExperNet because they are private networks not owned by the Ukrainian ISP we have cooperated with.

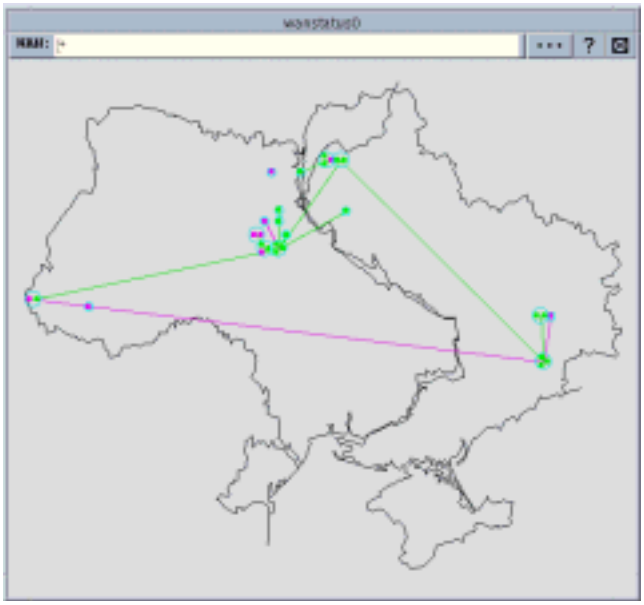

Figure 4. The experimental network zone of ExperNet in Ukraine

This experimental installation was initially used to test the correctness of the system. Later, the local network operators were adequately trained for using ExperNet for their day-to-day business. The evaluation of ExperNet from the network operators proved that the system is robust on a number of representative cases of management that the operators of the Ukrainian national network meet in every day practice. Performed tests cover typical cases of fault and performance management, such as the following:

- *A router is unreachable*, because either the interface of the router is down or because the process `inetd` that controls the network operations of the router does not exist.
- *A host is unreachable*, because of either an interface problem or the absence of the `inetd` process in the memory of the processor or some other cabling problem.
- *The HTTP or FTP service is not responding*, because either the `httpd` or `ftpd` process does not exist in the memory of the host or the response time of the specific host was bad or the processor was overloaded.
- *Bad Leased Line*. The TCP/IP connection over a leased line is malfunctioning because either there was a problem with the modem devices or there were physical errors on the line or the MTU value of the interfaces was badly configured or the line was simply overloaded. In the case that the leased line is shared between two agents, co-operation between them occurs sometimes in order to resolve the situation.
- *A modem is not working properly*. The modem connection over a leased line is broken due to some hardware or cabling fault.

The above malfunctioning cases were used to evaluate the performance of ExperNet during the demonstration that was prepared at the end of the project for all cooperating parties. Due to space limitations,

we will present only the most representative cases. Of course, all these errors could not possibly have happened during the demonstration accidentally, so we have deliberately caused them by switching off devices or artificially flooding the network connections with packets.

Figure 5 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 in order to repair the malfunction. The ExperNet's recommendation to decrease the value of MTU on the malfunctioning leased line was justified by the fact that on such a low bandwidth line "long" packets may cause more errors and packet-discards than shorter ones. The network operator would continue to get this message (if the line would remain overloaded) until he/she has manually changed the MTU on both interfaces. ExperNet does not automatically alter critical network parameters because network operators do not feel comfortable with such an automatization, even if the system gets their permission first.

A similar malfunctioning case is shown in **Figure 6**, where the line between two nodes of the second and third levels was down due to physical problems. However simple the diagnosis may seem, it requires reasoning from ExperNet in order to exclude every other possible cause. More specifically, the system has to determine first that network devices work properly on both sides of the leased line, and then that the parameters of corresponding interfaces are correct (for example the MTU, etc.) Finally, having excluded all other possible causes of problem, the system computes the percentage of packets discarded and those rejected due to errors

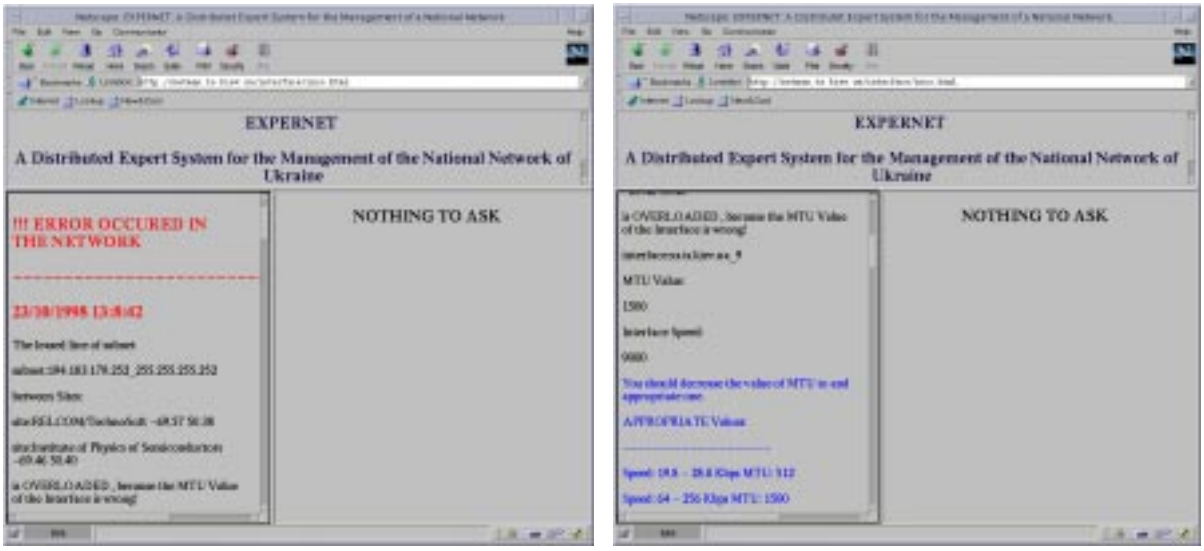

Figure 5. Overloaded leased line due to wrong MTU.

over the number of total transmitted packets, through the appropriate MIB variables on both interfaces. A percentage over 15% is a strong indication that the most probable cause of error is that the line has a physical error. Such reasoning could only be followed by an experienced network operator, while it would require a lot of time to check all these values.

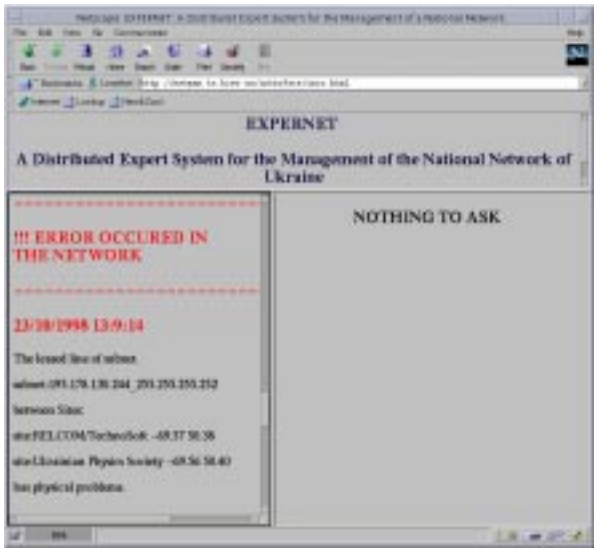

Figure 6. Bad physical line.

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, so a communication between them occurred via the exchange of messages concerning the problem. The messages were mainly inquiries about the status and operational parameters of the interfaces. Having determined that the problems did not lie anywhere else but in the MTU value, network operators on both sides were notified with a similar message as the one shown in **Figure 6** about the cause and remedy of the problem.

Another case of problematic leased line operation is shown in **Figure 7**. This case has been classified by the corresponding ExperNet specialist as a performance management problem (message in green color) and not as a hardware fault. The differentiation was based on the MTU value, which was a valid one, and the sum of errors and discards, which were less than 15%. Such a differentiation is difficult to be achieved by a network operator, since it requires complex reasoning during "busy", concerning network traffic, periods of day.

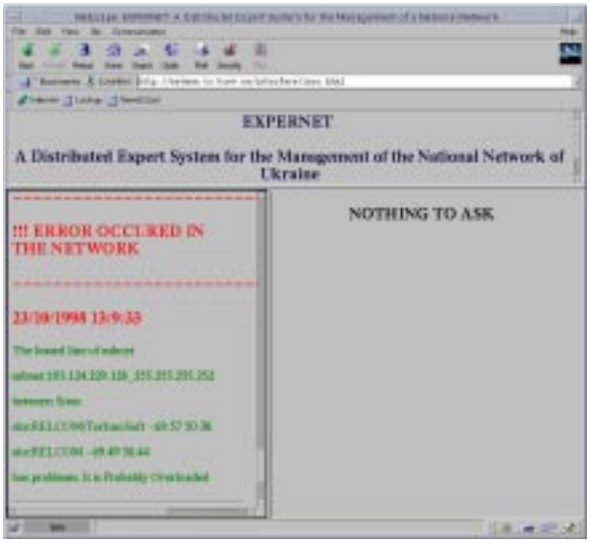

Figure 7. Overloaded leased line due to high traffic.

A common case occurs when a remote machine, hosting important networking services (e.g. DNS, WWW, FTP, etc.) is unreachable (**Figure 8**). 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. This test could not have been performed automatically by ExperNet through remote command invocation, because local administrators have had strong objections concerning security. Therefore, having no other alternative, the second level network operator called the remote host operator on the phone, who answered that the interface could not be brought up. Then 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 actually switched off. ExperNet's significance in this case lies on the fact that this problem could not have been detected and repaired manually by the network operators, until the users of the service, hosted by the remote machine, have complained (by phone or e-mail) to them. Capturing abnormal situations like the above leads to an increased availability of the offered services and consequently to more satisfied clients.

A more complicated faulty case successfully handled by ExperNet is shown in **Figure 9**. More specifically, ExperNet had detected that all remote hosts became unreachable and therefore, there should be something wrong with the leased line through which the remote hosts were connected and not with the hosts

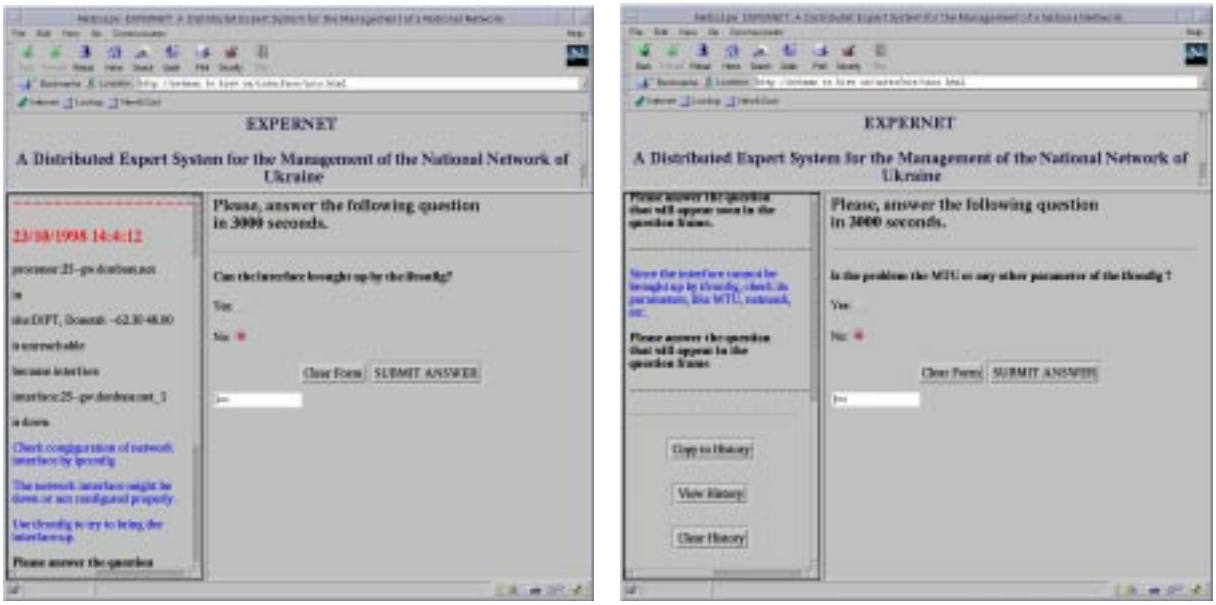

Figure 8. Remote host is unreachable.

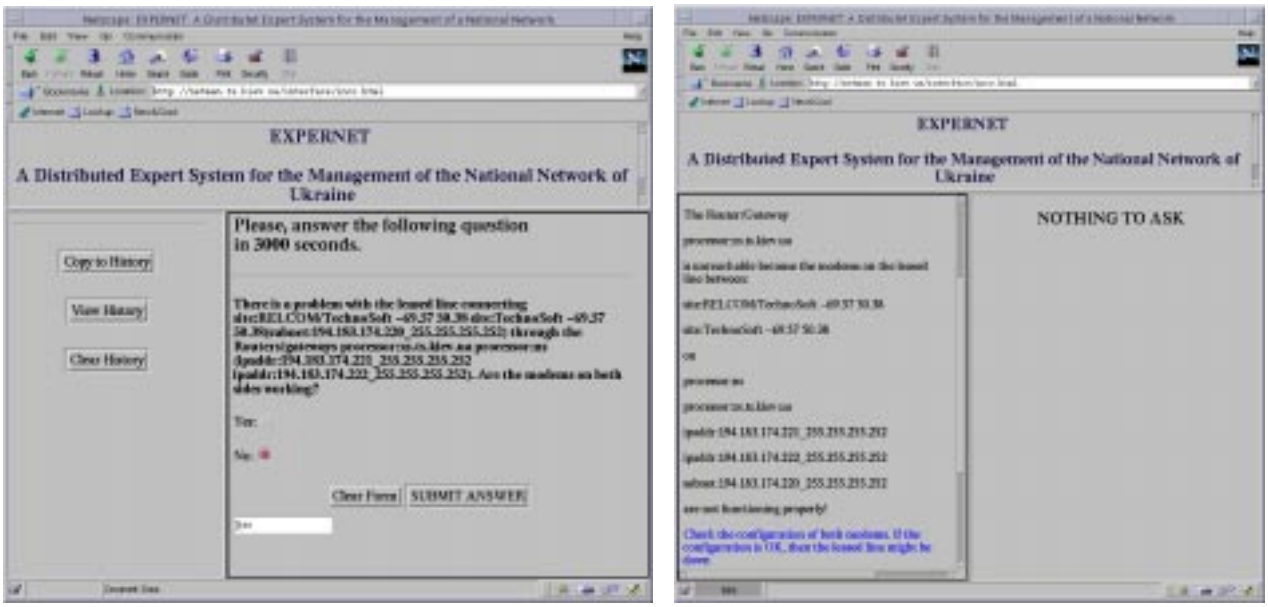

Figure 9. ExperNet reporting modem failure and suggesting actions.

themselves. Consequently, it asked the user if the modems on both sides of the line were working. ExperNet could not obtain automatically this information, because the modems were not SNMP-compliant. In this test case, the local modem was switched off and the network operator replied negatively. The system suggested that there was something wrong with either the modem configuration or the leased line itself, which can cause malfunctions on the modems. In this case, ExperNet's contribution lies on the fact that the unreachability of multiple hosts is aggregated (alarm correlation) and the case is not treated as multiple unreachability cases but as a single line's problem.

Practical Experience

The system was developed under the strict time limitations of an EU-funded project, reflecting on the system’s ability to cover a broad range of failure cases. During the development of the system we encountered several difficulties, presented in the sequel. Firstly, the knowledge acquisition procedure for building such a large and complex system requires a lot of time and resources, more than it was originally available. Additionally, the network experts that were to be interviewed were located in a different country (Ukraine) than the one the knowledge engineers were (Spain and Greece). This limited the communication to the exchange of questionnaires via Internet and few face-to-face interviews. Moreover, there were no experienced

users of the HNMS+ monitoring system among the network operators that were involved at the knowledge acquisition phase. Should such operators were available, they would have been able to deliver to the knowledge engineers, expert knowledge and everyday rules of thumb concerning network management in a better way. The development and integration of the various subsystems in a robust and efficient product was a time consuming task, since:

- The original HNMS system had to be extended in order to provide true hierarchical structure and various bugs had to be fixed;
- An interface between CS-Prolog II and the new HNMS+ system had to be implemented;
- An additional interface between the intelligent agents and the HNMS+ system had to be developed, based on the networking primitives of CS-Prolog II, so that networking data was adequately receipted and translated;
- Debugging of the various subsystems and of the interfaces between them was a hard task due to the distributed nature of the application.

Our initial design of the system did not include interaction with the network operators; ExperNet would have to perform all detection, diagnosis and repair actions automatically. However, several unexpected problems came up that reverted our plans to include asking questions to the operators. More specifically, the most important problems for the full automatization proved to be:

- The technology situation at the experimental installation zone that included non SNMP-compliant devices (e.g. modems) that could not be automatically monitored. Therefore, humans should visually inspect devices and ExperNet should get this piece of information as an answer to a question.
- The existence of single-line connections between network nodes that caused the complete unreachability of hosts should modems, lines, or routers break down. Therefore, when a line broke down network operators should phone each other to find out what's wrong with which devices and then answer ExperNet's questions.
- The fragmentation of the network's ownership to several organizations that caused responsibility conflicts (*Why should an external network management node manage my own private subnetwork?*) and security objections (*Is it safe to allow an external agent to run tests on my machines?*).

Another major problem encountered for the installation of ExperNet in several nodes of Ukrainian network was that although network administrators showed much interest for the ExperNet's network monitoring and repairing facilities, the practical application of ExperNet, HNMS+ and BigBrother required:

- Significant additional computer resources (e.g. additional UNIX host dedicated to the execution of the HNMS+ server and BigBrother);
- Additional negotiations with network administrators concerning the security of the information about their computers, which is captured in HNMS+ through SNMP agents;
- Allocation and additional training of human resources for operating the network monitoring software (HNMS+/BB).

Conclusions and Future Work

The explosive growth of the demands for networking in the last decade has increased the need for advanced management software that offers intelligent administration services. The application of expert system technology to this field can significantly increase the functionality of the current management systems and provide the means to deliver higher quality services to the end user.

ExperNet is a multi-agent intelligent system that offers constant monitoring of the state of its target network, including both the critical parts, such as routers and leased lines, and common network services, such as FTP or HTTP. The system’s error reporting and diagnosis capabilities can significantly decrease the downtime of the network components, thus leading to an increased availability of the overall network which is one of the most important goals of any administrator. ExperNet also offers suggestions about repair actions that should be taken by the administrator in order to resolve the abnormal network situations.

The system is based on existing and widely used management protocols (SNMPv2), which makes its application to any existing network possible. System’s distributed architecture makes it suitable for the management of Wide Area Networks, and permits the creation of autonomous systems for each management node, thus leading to a system that closely reflects the organizational structure of the WAN.

The development of ExperNet involved the development of a set of tools and the implementation of the interfaces between them. These have shown adequate performance and robustness in their operation in a real network environment. However, these tools can be also used independently. More specifically:

- *CS Prolog-II* is a general purpose programming language that can be used to develop any kind of logic-based system and especially systems, which require parallel and real time features. It should be noted that CS-Prolog II offers significant advantages in developing agent-based applications, through its TCP/IP communication facilities and other real-time features.
- The *HNMS+* network monitoring system in conjunction with the *Big Brother* host-monitoring tool can serve as a stand-alone management tool. The hierarchical structure of HNMS+ makes it an excellent tool for large size networks.
- *DEVICE* is a knowledge base system that can be used as a development tool for any rule based system. The object-oriented facilities and its ability to handle large collections of data in conjunction with the support for multiple types of rules provide a flexible platform for the implementation of expert systems.

The ExperNet system has been installed and tested in a real network environment in Ukraine and has performed well, in monitoring a network of significant size. As we saw from the performed tests, the system has shown robustness on a set of typical management cases that network operators of the Ukrainian national network meet in real, every-day practice. ExperNet is now working on two operating systems: Solaris 2.5 and FreeBSD 2.2.6. There are plans for extending the area of application, if the problems with local administrators are resolved.

Future Work

Probably the most significant extension that could add valuably to the current implementation concerns the knowledge base itself; new management cases should be added, covering the full range of management areas, i.e. fault, performance, configuration, security and accounting management. It is quite possible that the addition of these new cases would require none or minor modifications to the existing system. Additionally a number of vendor specific knowledge bases can be developed, for important network equipment providers like CISCO or ALCATEL that will aim to exploit the management characteristics of each network device. This modular approach will increase the mobility and flexibility of the present system.

More explanation facilities should be added in ExperNet to increase the trust of the user to its results and suggestions and provide a platform for tutoring resolution methods for network management problems. In addition, HNMS+ can be extended to co-operate with SNMPv3 agents; such an extension requires some

modifications to the core of the monitoring tool. However, these modifications are expected to be rather simple, since the system structure is modular and changes would not affect other parts of the system.

We are currently working on the development of an integrated Wed-based user interface that will host both ExperNet and HNMS+ data and messages, based on PHP and XML technologies. This will allow the remote monitoring of the network status, i.e. the network operator will not have to physically be on the site where the visual HNMS+ user interface is installed.

References

[1] I. Vlahavas, N. Bassiliades, I. Sakellariou, M. Molina, S. Ossowski, I. Futo, Z. Pasztor, J. Szeredi, I. Velbitskiy, S. Yershov, S. Golub, and I. Netesin, "System Architecture of a Distributed Expert System for the management of a National Data Network," *Proc. 8th International Conference on Artificial Intelligence*, AIMSA'98, LNAI 1480, Springer, 1998, pp. 438-451.

[2] J. Cuena, S. Ossowski, "Distributed Models for Decision Support," *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Weiss (Ed.), AAAI/MIT Press, 1999.

[3] M. Molina, S. Ossowski, "Knowledge Modelling in Multiagent Systems: The Case of the Management of a National Network," *Proc. 6th Int. Conf. on Intelligence in Services and Networks*, in *Intelligence in Services and Networks, Paving the Way for an Open Service Market*, H. Zuidweg, M. Campolargo, J. Delgado, A. Mullery, (Eds), LNCS 1597, Springer, April 1999.

[4] B.J. Wielinga, A.T. Schreiber, J.A. Breuker, "KADS: A Modelling Approach to Knowledge Engineering," *Knowledge Acquisition*, 1992.

[5] J. Cuena, M. Molina, "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*).

[6] W. Clancey, "Heuristic Classification," *Artificial Intelligence*, Vol. 27, 1985.

[7] B. Chandrasekaran, T. Johnson, J. Smith, "Task-Structure Analysis for Knowledge Modeling," *CACM*, Vol. 35. No. 9, 1992.

[8] D. Brown, B. Chandrasekaran, *Design Problem-solving: Knowledge Structures and Control Strategies*, Morgan Kaufman, 1989.

[9] M. Barbuceanu, S. Fox, "COOL: A Language for Describing Co-ordination in Multi Agent Systems," *Proc. ICMAS*, 1995

[10] H.-J. Müller, "Negotiation Principles," *Foundations of DAI*, O'Hare and Jennings (eds.), Wiley, 1996.

Sidebars

The Device Knowledge Base System

DEVICE (**D**ata-driven & **EV**ent-driven rule **I**ntegration using **C**omplex **E**vents) is an active Knowledge Base System[1] that runs on top of a Prolog-based active OODB[2] (**Figure 10**) and provides a number of interesting features, such as multiple rule type support and object-orientation. The infrastructure of DEVICE is based on the smooth integration of production and deductive rules using event-driven rules. Declarative rule conditions are compiled into a *discrimination network* that consists of simple and complex events which record and combine database modifications that could possibly make a rule fire.

A rule base in DEVICE can be a mixture of low-level event-condition-action (ECA) rules, as well as higher level production and deductive rules[3]. Goal-driven rules can also be used in DEVICE in the form of methods, which are Prolog code fragments. For the efficient matching of production rules, DEVICE smoothly integrates a *RETE-like* discrimination network as a set of first class objects by mapping each node of the network onto a complex event object of the database. 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. Furthermore, rule priorities can be established to order rule execution. The production

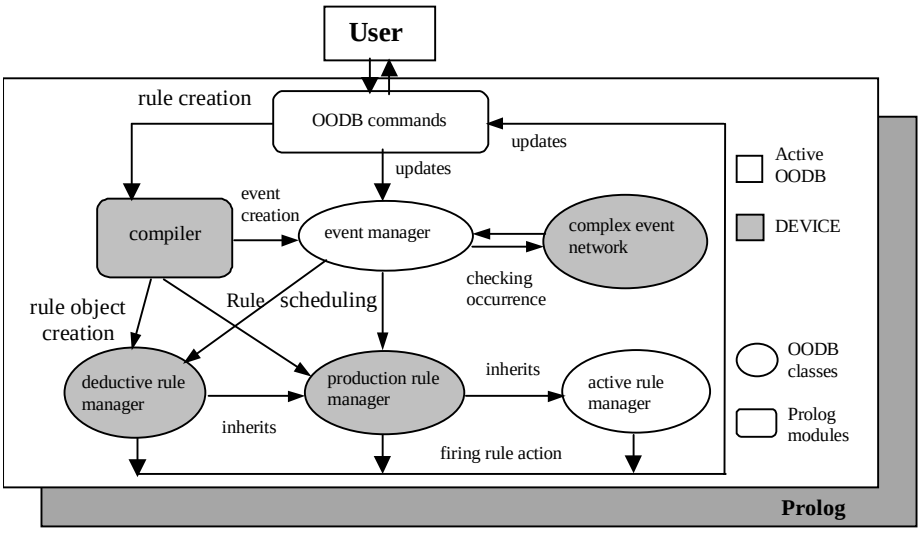

Figure 10. The architecture of the DEVICE system.

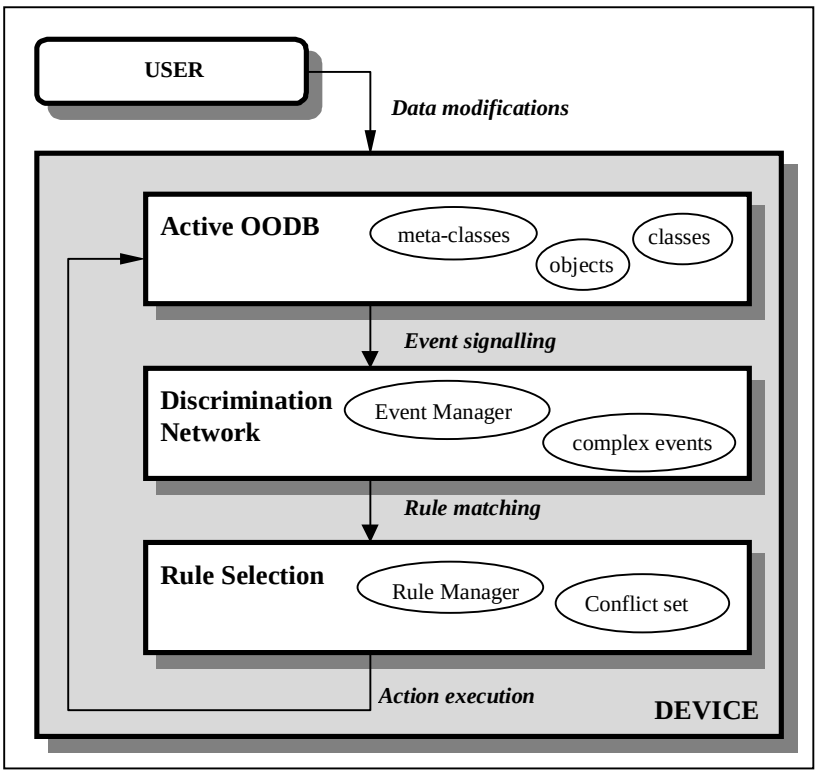

Figure 11. The production cycle of DEVICE.

cycle of DEVICE is shown in **Figure 11**.

Furthermore, DEVICE supports rule modules, a mechanism that provides a robust and easy way to control rule execution and allows the focusing of the inference procedure on a rule set that performs a specific task. *Control rules* manage task/module switching whenever it is required.

The resulting system is a flexible, yet efficient, active Knowledge Base System that gives the user the ability to express knowledge in a variety of high-level forms for advanced problem solving in data intensive applications, as well as applications that require re-activeness to external changes.

References

[1] N. Bassiliades, I. Vlahavas, "Processing Production Rules in DEVICE, an Active Knowledge Base System," *Data & Knowledge Engineering*, Vol. 24(2), pp. 117-155, 1997.

[2] O. Diaz, N. Paton, P.M.D. Gray, “Rule management in object oriented databases: A uniform approach,” *Proc. Int. Conf. on Very Large Databases*, Morgan-Kaufman, Barcelona, Spain, 1991, pp. 317-326.

[3] N. Bassiliades, I. Vlahavas, A.K. Elmagarmid, “E-DEVICE: An Extensible Active Knowledge Base System with Multiple Rule Type Support,” *IEEE Trans. On Knowledge and Data Engineering*, Vol. 12(5), pp. 824-844, 2000.

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 NAS Hierarchical Network Management System[1] (or simply HNMS) is a software system designed to assist the network operator in managing such networks. In ExperNet, we have approached network monitoring and management by:

- Developing HNMS+, by extending the functionality of HNMS to suite our needs and by correcting quite a few shortcomings of the prototype version of HNMS;
- Integrating the BigBrother network-monitoring tool with HNMS+;
- Adding special interfaces in CS-Prolog II, which translate the communication protocols of the above tools into knowledgeable Prolog data structures.

The HNMS Prototype

The prototype version of the HNMS system was developed to cover the network management needs 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.[1]

HNMS was originally designed to incorporate three types of software modules, which typically reside on separate hosts throughout the network: server module, input/output (IO) module and user interface (UI) module. It was also mentioned that it would be easy to append two new modules into HNMS architecture, namely a Rule Based Intelligent Processor (RBIP) module and a DBMS front-end module. The primary architecture of HNMS assumed that there would be one server module, any number of UI modules and one IO module in each local subnetwork.

The *server module* is the hub of the system; it provides a center for dissemination of global topology and status information. Its responsibility is to maintain an up-to-date model of the network and process all requests from other modules that connect to the server. *IO modules* directly monitor their LAN via SNMP and promiscuous Ethernet monitoring and forward the status of network objects to the server. The *User Interface* (*UI*) module resides on workstations with graphics capabilities and provides to the user access to real-time or logged data about the current network state in visual form, by connecting to the server.

All inter-module communication is performed using the Hierarchical Network Management Protocol (HNMP).[1] HNMP extends SNMP by introducing a new set of MIB variables (HNMS MIB) in addition to standard MIB variables.[2] SNMP is still used by IO modules to poll SNMP agents in the network. HNMP was specially developed to reduce the network management traffic and to avoid bottleneck problems in high loaded networks. HNMS MIB objects represent IP network elements or other useful information about network management.

HNMS provides several types of status diagrams (**Figure 12**), each representing the view of state of a network element using a color code. These diagrams are updated by the server, reflecting changes of the elements' status. Examples of diagrams are: the *WAN diagram*, which depicts the state of the IP network and the routers over a geographical reference; the *Site diagram*, which represents all LANs that are connected to the routers at a given site; the *Object diagram*, which is a textual display of the HNMS variables, etc.

HNMS+: Extending the Prototype

The prototype version of HNMS did not fully adhere to its specification, making the collection of network state information for ExperNet very difficult. We have implemented HNMS+, a new, extended version of HNMS, which is truly a hierarchical, distributed system that fully supports HNMS functionality and extends

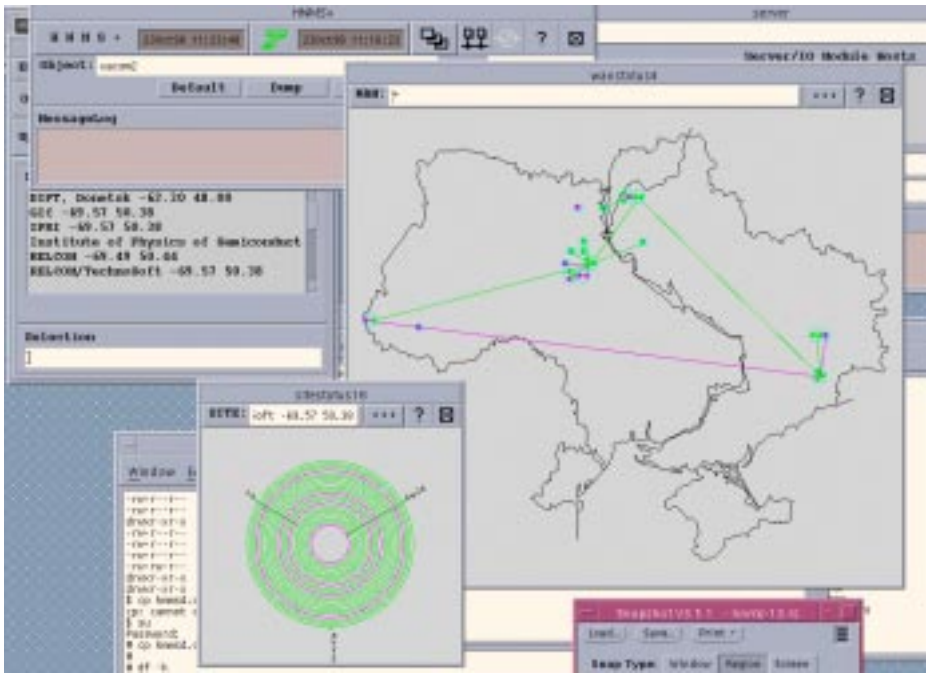

Figure 12. Typical administration terminal of HNMS+.

it with new features.

More specifically, the prototype version of HNMS did not support multiple separate IO modules because their functionality was merged into the single server module, and consequently, there was no true hierarchy. The existence of multiple IO modules was necessary to collect local information about the behavior of the particular subnetworks. On the other hand, IO modules should be able to serve subscription requests from intelligent agents. As a result, in the new HNMS+ system, we kept the functionality of the server and IO modules into a single module, called *master module*, but we provided the ability to have a hierarchy of multiple such modules.

Master modules reside on hosts located at strategic points within WAN and handle actual data collection and propagation. More specifically, they use the SNMP protocol for local data collection from the SNMP agents attached on the actual network devices of their LAN, playing the role of an IO module. They also accept information from low-level modules, playing the role of a server module. Finally, they build a network representation and they pass filtered management data up to the immediate higher-level modules, using the HNMP protocol. Data is propagated only when their values change, in order to avoid bottlenecks created when the network is flooded with management information towards the ExperNet agents.This process is recursive and it terminates on the highest-level of the network. Three such levels were required to cover the experimental network zone of Ukraine.

The HNMP protocol of HNMS was generally suitable for HNMS+ but one vital change was needed in the announcement of new network objects. Specifically, new ExperNet agents that subscribe to their local master module are announced recursively both upwards and downwards in the hierarchy, since each agent should be aware of all existing agents, regardless of their position in the hierarchy.

The network administrator can connect to a master module of 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 have decided that agents should connect only to their local master module to minimize traffic.

We have 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, HTTP, SMTP, NNTP, or any system resource on a

network host, such as processor load, hard disk usage, warnings and alarms in system's log file, or information on important active system processes. Furthermore, we have developed the *database module*, mentioned in the original HNMS specification. This is an SQL front-end process whose function is to regularly store the current network state and network performance data to a DBMS server (PostgreSQL[3]), over a network connection and, as a result, to support statistical analyses of past network states. The database module interacts with the HNMS+ master module only when the values of the variables of the local master module change, in order to avoid network overloading by SQL requests.

Integration of the BigBrother Monitoring System

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 and network connectivity, as well as intra-machine communication programs. Using Big Brother, administrators can keep track of disk usage, CPU loading, FTP, SMTP, HTTP servers, and other important processes. The results of monitoring are reported in a status matrix, using a color code for each system/area combination, which is displayed on a central monitoring station and presented through a Web based user interface (**Figure 13**).

An important issue in network management is the evaluation of the quality and reliability of particular TCP/IP network services, such as FTP, HTTP, SMTP and NNTP, and local host resources, such as CPU,

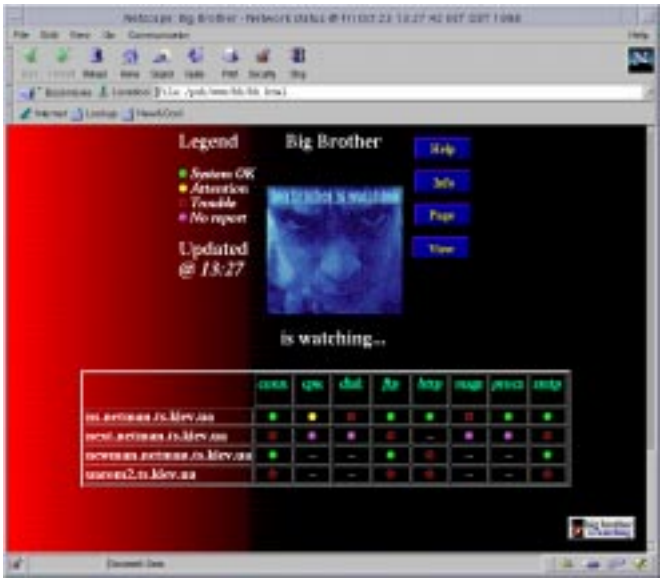

Figure 13. Big Brother indicating that the HTTP service is down.

disk, and so on. SNMP agents cannot provide such information, so we have integrated BigBrother into HNMS+ in order to achieve monitoring of TCP/IP services and remote computer resources by the ExperNet agents. We have also extended HNMS+ MIB to incorporate the additional monitoring values of the status matrix of BigBrother. HNMS+ master module analyzes a local log file created by BigBrother and fills out the previously mentioned MIB variables.

Furthermore, we have developed a UNIX daemon that offers the possibility of remote UNIX command invocation. In this way, ExperNet agents can acquire information that cannot be obtained directly from HNMS+ but only through command execution on the monitored remote hosts (e.g. the "`traceroute`" and "`tcpdump`" packet monitoring utilities). Although we could use the usual "`rsh`" UNIX command for this purpose, we preferred 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.

Interfacing CS-Prolog II to HNMS+

In order to translate the HNMP communication protocol into Prolog data structures that ExperNet agents could manage, we have developed in CS-Prolog II a special interface with HNMS+. The interface is a special *Knowledge-Based Intelligent Processing* (KBIP) module in HNMS+, which is an application that obtains, through the HNMP protocol, information about network traffic and utilization of the network elements. On each node, HNMS+ provides to the ExperNet agents an immediate perception of the network state. Using KBIP modules, ExperNet agents not only can immediately determine the general state of the network, but they are also notified by HNMS+ about important network events.

References

[1] G.A. Jude, L.E. Schecht, "The NAS Hierarhical Network Management System," *Integrated Network management III*, H.-G. Hegering and Y. Yemini (Eds.), Elsevier Science Publishers, Amsterdam, 1993.

[2] J. Case, M. Fedor, M. Schoffstall, J. Davin, "Simple Network Management Protocol," RFC 1157, *SNMP Research*, Performance Systems International, MIT Laboratory for Computer Science, 1990.

[3] *http://www.postgresql.org*

[4] *http://www.iti.qc.ca/users/sean/bb/bb.html*

The CS-Prolog II System

In the development of any multi-agent system, a crucial issue for the co-operation and co-ordination of the involved agents is the implementation of the required communication facilities. In ExperNet, these facilities are developed using CS-Prolog II[1] a distributed Prolog system enhanced with networking facilities.

General overview

CS-Prolog II is a distributed Prolog system, which has been developing since 1995.[2] The syntax and the built-in procedures of the language are based on the standard ISO/IEC 13211-1. Furthermore, the language has been extended with features that were not included in the ISO standard, like modularity, multitasking, real-time programming and network communication.

CS-Prolog II supports the communicating sequential process[3] programming methodology in a Prolog environment. On a single processor machine, a time-sharing scheduler controls the concurrent processes. The inter-process communication is ensured by a rendezvous mechanism, i.e. synchronous message passing through communication channels. Processes can backtrack, however communication is not backtrackable. The channel-based communication had been extended with networking capabilities, so that possible message passing between different CS-Prolog II applications across the Internet is made possible. It also provides communication with foreign (non CS-Prolog) applications, an interface to relational data base systems, real-time programming methods like cyclic behavior, reaction to predefined events, timed interrupts, etc.

CS-Prolog II 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.

Networking facilities

As a natural extension of CS-Prolog II channel concept, the external communication conceptually consists of unidirectional message streams. In order to speed-up 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, called *community*, which consists of one or more fellow applications that the program can communicate with, called *partners*. Partners are accessed via channel messages, while a separate mechanism is introduced for connecting channels to external applications, called *foreign partners*. The most important entity for this task is the so-called *port*, which represents an incoming message sub-stream. Ports are explicitly created and they play the role of a sender for a CS-Prolog II channel, which is 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*, 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 partner is not foreign) to where the stream is directed. The 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 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*, whose addresses have to be available in advance for the program (hardwired in the program, obtained from a file, etc.).
- *Net partners*, which have signed up at HNMS+, and the CS-Prolog II program included them in its local picture of the network. The address of a net partner is obtained from HNMS+.
- *Latent partners*, who are known by HNMS+ but the CS-Prolog II program didn't include them in its local network picture. The address and some other attributes of a latent partner are asked from HNMS+.

In order to be able to communicate with a net partner, a configuration process has to be performed in the current TCP/IP implementation of the low-level communication protocol. The program has to add explicitly this partner 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 that performs the appropriate data and protocol conversion, called *mediators*. At present, there are two mediators defined:

- ASCII, for plain text communication.
- HNMS, for communicating with the HNMS server.

Conceptually, a local mediator is communicating with a remote mediator, hosted at the foreign partner, addressing the dock it offers. Data sent by the remote mediator are accepted at the dock of the local mediator. In fact, the remote dock is a virtual entity that abstracts the capability of the foreign application that it can be connected to.

Docks are similar to ports in the sense that they play the same role in the communication. The difference is in the way a dock is prepared for operation and is connected implicitly by the mediator on behalf of the foreign partner. In order to configure a foreign partner, the application program shall:

a) Create a dock;

b) Create a mediator of the appropriate kind, naming a free dock for it;

c) Configure the desired foreign partner, naming the mediator in the list of the configuration parameters (implying thereby that the partner is foreign).

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 are accepted and produced by them.

References

[1] I. Futo, “A Distributed Network Prolog System,” *Proc. of 20th Int. Conf. on Information Technology Interfaces*, ITI 99, Pula, 1998, pp. 613-618. (see also *http://www.ml-cons.hu/dload-e.html#csprof*)

[2] I. Futo, “Prolog with Communicating Processes: From T-Prolog to CSR-Prolog,” *Proc. of the 10th Int. Conf. on Logic Programming*, ICLP 93, ed. D.S Warren, MIT Press 1993, pp. 3-17.

[3] C.A.R. Hoare, "The Communicating Sequential Processes," *CACM*, Vol. 21, Aug. 1978.