1

Multi-level reasoning for managing distributed enterprises and their networks

J. Frey
L. Lewis
Cabletron Systems
P.O. Box 5005, Rochester, New Hampshire, USA 03866
j.frey@ieee.org, lewis@ctron.com

Abstract

In this paper we describe our experiences and directions regarding the management of multi-domain Enterprise Networks (ENs). Our views are (i) the management of the EN is a distributed problem-solving activity and (ii) recent results in automated problem-solving in the distributed artificial intelligence and robotics communities can contribute to our understanding, development, and deployment of management software for the EN during the 90s and into the 21st century. We discuss a distributed, multi-level architecture for EN management and describe examples in fault management and business process management. Our examples demonstrate an integration of two off-the-shelf Network Management (NM) products: the Spectrum NM platform from Cabletron Systems and NerveCenter from the Seagate Corporation.

Keywords

Distributed Artificial Intelligence, Enterprise Management, Network Management

1 INTRODUCTION

The idea of an intelligent, self-healing network has caught on. As in the robotics community, network researchers and managers foresee an autonomous, self-managed network that can negotiate problems with little human intervention. In the late 90's and into the 21st century, we should see a shift toward a view of the enterprise network (EN) as an intelligent, autonomous (or semi-autonomous) agent [1]. Where robots have sensors, sense-data, and effectors, the EN will have analyzers and monitors, network traffic and behavior data, and system commands to effect tasks such as failure detection/diagnosis/repair, upgrading software, dynamic reconfiguration, and controlling network devices. In the same way that researchers give robots instructions in the form of goals and the robot figures out how to achieve the goals, network managers will give

A. A. Lazar et al. (eds.), *Integrated Network Management V*

the EN a set of goals and the network will figure out how to achieve those goals. This view of the EN opens the way for thinking about outstanding issues with intelligent architectures in general and their contributions to understanding and achieving good network management solutions.

One such issue is task analysis. What network tasks are better performed by reasoning with raw message traffic data? With network events? With network alarms? What tasks are better performed by reasoning with a symbolic model of a network? What tasks require reasoning that is distributed over multiple layers of network abstraction, as opposed to cooperative reasoning systems over a single layer of abstraction? What control mechanisms will be needed? These sorts of issues have received considerable attention in the AI and robotics communities, and similar issues have surfaced in recent research on the management of enterprise networks.

Our objectives in this paper are as follows. First, we provide a brief account of distributed, multi-level reasoning in robotics and AI research. Next, we consider the problems and goals of EN management and show how the achievements in the former disciplines map to an EN management architecture. Finally, we discuss working examples of the architecture in fault management and business process modeling. Our examples demonstrate an integration of the Spectrum network management platform from Cabletron Systems and NerveCenter from the Seagate Corporation.

2 DISTRIBUTED AI AND ROBOT REASONING

2.1 What is Distributed Artificial Intelligence?

A common approach to problem-solving is to decompose a task into subtasks and to perform the subtasks sequentially or assign them to individual agents in parallel. This approach assumes that a central controller exists which decomposes the task, assigns subtasks to appropriate agents, and synthesizes the results. In contrast, distributed artificial intelligence (DAI) is concerned with problem-solving for which a central controller is not present and for which subtasks are interdependent [2,3,4]. Specifically, DAI is concerned with problems that exhibit the following characteristics:

1. *Control and expertise required for solving a problem are distributed over a number of agents.*
2. *An agent is not always able to complete his subtask alone.*
3. *No single agent can solve the entire problem alone.*
4. *The solution to the problem requires cooperation and communication among agents in such a way that the expertise of each agent combines to solve the problem.*

It is clear that EN management is a DAI problem. A first-order categorization of network management tasks includes fault, configuration, accounting, security, and performance management [5]. However, it is not clear that this is the best or a complete categorization of network management subtasks, and less clear how and in what ways the specialists performing the subtasks depend upon the other specialists. A good way to pose the question is "What does X need from Y, and when and how often is it

needed?", where X is a particular specialist and Y is the set of remaining specialists. A prior question, of course, is "What are the Xs?"

For example, problem-solving that involves fault, configuration, and performance management often requires specialists in each area who compare notes in order to solve the problem. In addition, another specialist in budget control and expenditures is consulted in order to find the most reasonable solution from a cost perspective.

As an aside, it is interesting to note that a problem that could turn out to be a DAI problem is the re-creation of intelligent behavior exhibited by a single human agent. For example, a controversial issue in the robotics community is that of determining what mechanisms are required to build an autonomous agent [6]. On the one hand, it has been argued that an agent requires a single controller who coordinates tasks such as perception, cognition, acting, planning, and learning. In this view, intelligent behavior is not a DAI problem since a controller delegates these tasks to sub-agents, where the sub-agents do their work independently of each other and the controller synthesizes their results. Alternatively, it has been argued that intelligent behavior of a single agent is itself an example of DAI. Perception, cognition, etc. are performed by specialists who negotiate among themselves, and *agency* is an emergent property or epiphenomenon of the multiple agents [7].

As with any sufficiently large multiple-agent enterprise, including EN management and possibly a single human agent, these questions are hard. A thorough study of network management within the DAI framework has not been forthcoming. Comparatively, there is more work and experimentation with AI paradigms applied to specific management tasks, e.g. paradigms such as neural networks, fuzzy logic, case-based reasoning, and expert systems. There is less work describing how these pieces fit together. The reason is that network management is plain difficult, and the understanding and characterization of DAI problems in general are equally challenging.

Here we do not try to characterize a complete network management architecture. In section 3 we discuss work on a piece of the total solution, fault management, and describe a working DAI architecture for achieving distributed fault management in the EN. Other efforts toward defining the subtasks and dynamics of integrated network management include [8,9].

2.2 What is Robot Reasoning?

The work on implementing intelligence in a robot since the early 80s is an interesting story. If our hypothesis that intelligent networks will come to be viewed as autonomous agents turns out to be true, then the uncovering of false starts and promising approaches in the robotics community could help us in understanding intelligent network management. In this section we describe two prominent approaches to the development of intelligent architectures for robots. For further detail, see [6].

2.2.1 The Single Loop Architecture

The single loop architecture (SLA) for robot intelligence is shown in Figure 1. Intelligent behavior starts with robot sensors and ends with instructions that are executed by robot effectors. Initially, sensory input is passed through several layers of abstraction, e.g. signals, signs, and symbols. Each layer filters noise and extraneous data out of the

data passed to it, and transforms the data into fewer, more informative chunks of data. When data becomes manageable, usually at the symbolic level, it is compared with predefined knowledge about what instructions should ensue. This operation is usually defined as "coordination" and may be implemented in a number of ways, including simple look up tables or expert systems. The output of the coordination module is a set of symbolic instructions. These instructions are decomposed down through levels of decomposition until they are suitable for processing by the robot's effectors, usually in the form of signals

The important ideas in the SLA are (i) the compression of sensor data through layers of abstraction, (ii) the coordination of highly abstracted sensor data and goals, and (iii) the decomposition of instructions through layers of abstraction into a form that is executable by effectors.

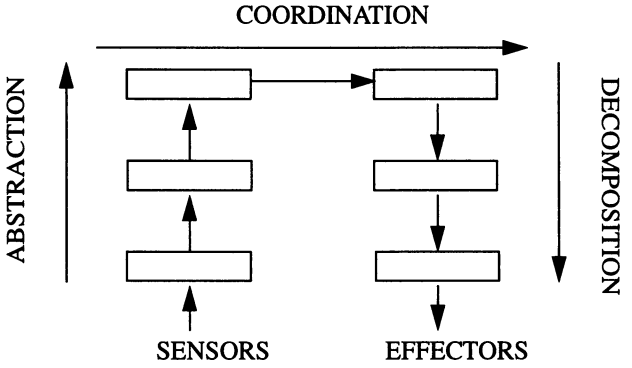

Figure 1 A single loop architecture for robot reasoning.

It is generally agreed that the SLA is unsuitable for dynamic, unstructured tasks. Sensor data must be abstracted into symbols before they can be coordinated with the goals of the system and initiate the decomposition of instructions for effectors. As a result, the processing time required to transform sensor data to control instructions through the loop is prohibitive. By the time the system figures out what to do in environment A, it is likely that environment A is obsolete and the instructions are no longer applicable. In other words, the rate at which the world changes over time is disproportionate to the amount of processing required for timely behavior.

2.2.2 The Multiple Loop Architecture

Robotics researchers observed that the coordination of abstracted sensory data and goals do not have to take place only at the symbolic level. Coordination could occur at any level of abstraction. Further, they observed that coordination at lower levels required increasingly less time, since upper levels would be by-passed. Therefore, problems which require a quick solution could be handled at lower levels of abstraction, and problems which require more time could be handled at upper levels of abstraction. This distinction is generally referred to as "reflexive" behavior for short-term solutions and "deliberative" behavior for long term solutions. Figure 2 shows a design that reflects these ideas, called the multiple loop architecture (MLA).

Each level of the MLA is a separate control loop that corresponds to a specific class of problems, where problems are partitioned and assigned to levels according to the amount of time and type of information required to solve them. For example, the short-term abstraction/coordination/decomposition loop at the lowest level provides quick reaction, bypassing upper level control mechanisms. In the network management domain, such tasks might include intelligent routing and temporary disconnection to a busy host. The medium-term loop provides reaction to more complex problems and operates on increasingly abstract input such as signs. Tasks of this sort might include alarm correlation in a busy network with multiple alarms, where some alarms are real and others are apparent, and the task is to distinguish the two and suppress all apparent alarms. The top level would provide reaction to problems that require more time. The classic example of a task of this kind is the reasoning involved in deciding to move a host from subnet A to subnet B because the majority of the host's clients reside on subnet B, thereby causing increased traffic on the link between A and B.

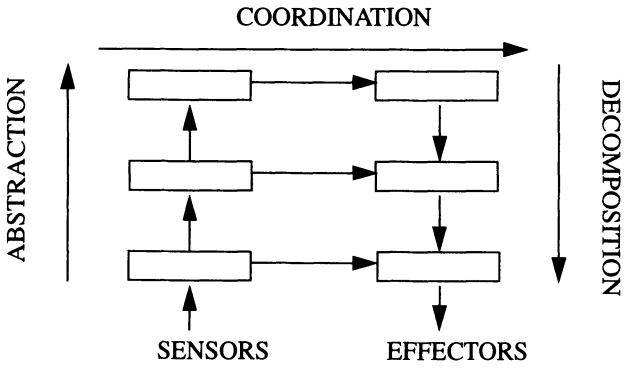

Figure 2 The multiple-loop architecture for robot reasoning.

In comparison with the SLA, the MLA is intuitive and shows promise for a clearer understanding of intelligent network management. However, the primary unresolved issues of the MLA approach are (i) a methodology by which to classify and divide tasks according to information available and time required to solve them, (ii) the particular algorithms on each level that execute its class of tasks, and importantly (iii) how to synthesize activities on multiple levels when a task requires multiple levels of coordination. The latter issue is the classic DAI problem.

3 MANAGING THE ENTERPRISE NETWORK

With the ideas and insights of the preceding section under our belts, let us now turn to the practical matter of managing the EN. Three important truths regarding EN management motivate our distributed, multi-level architecture for EN management:

1. The EN is inherently a distributed, multi-domain entity. ENs typically are partitioned in ways that help administrators understand and manage the EN, e.g. with respect to geographical domains, functional domains, or managerial domains.
2. The tasks involved in managing distributed ENs are too complex for a single controlling agent, and thus the tasks themselves must be partitioned into distributed, cooperative agents.
3. The data types for analyzing and reasoning about EN behavior come in various forms, e.g. traffic data, symbolic models of the network, events, alarms, et al.

3.1 Spectrum's Distributed Client/Server Architecture

For the reasons above, the Spectrum NM platform was designed as a distributed client/server architecture. The Spectrum servers, called SpectroSERVERs (SSs) monitor and control individual EN domains. The Spectrum clients, called SpectroGRAPHs (SGs) may attach to any SS in order to graphically present the state of that SS's domain, including topological information, event and alarm information, and configuration information.

Importantly, each domain can be viewed from a single SG. See Figure 3. If SG-1 is attached to SS-1, but the user wishes to see the domain controlled by SS-2, the user can click on an icon in SG-1 that represents SS-2. Figure 3 shows the primary client/server attachment between SG-1 and SS-1, where virtual attachments between SG-1 and other SSs are indicated by dotted lines.

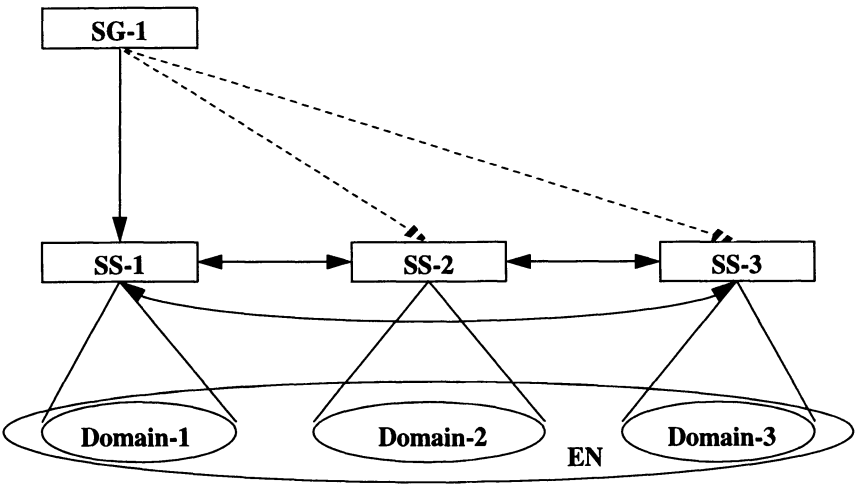

Figure 3 The SPECTRUM distributed client/server architecture.

3.1.1 Experiences with the Architecture in an Internal Testbed

We have tested the architecture in an internal testbed to establish operational feasibility. A virtual enterprise network was configured using 250 SSs situated in our U.S. and

U.K. facilities. A three-layered hierarchical topology was used, with one master SS connecting to 14 SSs, each of which in turn connecting to 15-20 more SSs. Each end-node SS was given a database with several hundred manageable devices.

Cabletron recommends a maximum 7:1 ratio among SSs that are configured hierarchically. This ratio is derived from workstation operating system characteristics rather than communications traffic load among SGs and SSs. Note that load would be a strong, adverse factor if we allowed a one-to-one correspondence between SGs and SSs, or if we allowed heavy communication among SSs.

In our internal testbed, we purposely exceeded the 1:7 ratio in order to stress the architecture. Performance, accuracy, and reliability were monitored over a period of several days, during which time a variety of simulated failures were introduced and the resultant behavior analyzed. A few communication flaws at the physical layer were identified and corrected, but as a whole the test was successful and the test bed operated without incident. Subsequent passes through this test plan are being used to exercise new and enhanced distributed and fault tolerance functions.

These initial tests provided a good, empirical argument for the scalability of the distributed, client/server architecture. We consider each SS as an intelligent agent, capable of presenting management data on demand to any client SG. This keeps inter-SS communications to a minimum. Each SS "knows about" its peer SSs, but is prohibited from extensive communication with them. In Section 3.2, we show how SSs may communicate by intermediary agents who reside at higher levels of NM abstraction.

3.1.2 Customer Installations

The distributed version of Spectrum has been installed at dozens of customer sites worldwide, with setups ranging from a few (2 or 3) SSs to several hundred. For the most part, customer ENs are divided into geographical domains, using an SS at each facility or campus, with a central master SS at a headquarters location.

A particularly challenging project currently in progress is the management of a telecommunications network in Eastern Germany [10], deployed by Deutsche Telekom. This project poses unusual requirements because it is purely non-SNMP, using only a proprietary management protocol. Spectrum was chosen as the management platform because (1) it has a distributed, client/server design, (2) it has APIs for developing non-SNMP management applications, (3) it has APIs for configuring intelligent SS agents, and (4) it enables representation of both devices and services involved in the EN.

Consider (3). With multi-domain ENs with corresponding SS agents, polling-based management can be costly in terms of bandwidth load. By restricting SS polling (i.e. only using it for testing basic element presence/status) and instead having managed elements forward management data to the SSs via traps, in-band management traffic is reduced considerably. This was a requirement for the application. Note that a transition from polling to trap-based management and intelligent agents is considered by some to be the future of EN management [11].

Consider (4). Data collected via the management system is being utilized in two ways. First, network devices are represented topologically in order to monitor and control the operations of the telecommunications network as a whole. Second, a service-based representation is used to monitor/manage usage and repairs so that, for example, large business customers may be given relatively higher priority for repairs than resi-

dential customers. This secondary representation can be easily accomplished by giving relative weights to each managed element's alarms.

As of the end of 1996, the number of SSs deployed for this application was nearing 1000 (490 primary servers, each with a fault-tolerance backup server). The NM configuration is similar to our internal testbed save that (i) each end-node SS manages both devices and services, (ii) there are only two tiers (i.e. there is not yet a top-level master SS), and (iii) there are more end-node SSs per second tier SS (up to 40). Performance and capability results thus far have been excellent, and the project is on track towards a planned deployment of 4000 total SSs (first tier plus second tier). The success achieved thus far has already resulted in commitments for similar projects by telecommunications providers in several other countries.

3.2 Multi-Level Reasoning

Let us stop a moment to think about the management tasks that occur within network domains and management tasks that occur across domains. We will use fault management as our first example.

Fault management consists of event monitoring and filtering, event correlation, escalation of events to alarms, alarm correlation, and diagnosis/repair. The SSs that monitor individual network domains perform these tasks with Spectrum's inductive modeling technology (IMT) [12]. We may refer to this as *intra*-domain event/alarm correlation.

With multi-domain ENs, the requirement now is to perform the same function across domains. For example, a failed router in Domain-1 may affect the applications running in Domain-2. Conversely, the cause of an application failure in Domain-2 may be identified as the failed router in Domain-1. We refer to this as *inter*-domain event/alarm correlation. Since we have limited inter-communication among SSs, we need to find some other way to do inter-domain alarm correlation.

In light of our discussion of multi-level reasoning in Section 2.2, we can conceptualize the inter-domain event/alarm correlation task as shown in Figure 4. The figure, however, is somewhat misleading because it is in two dimensions. The bottom-most

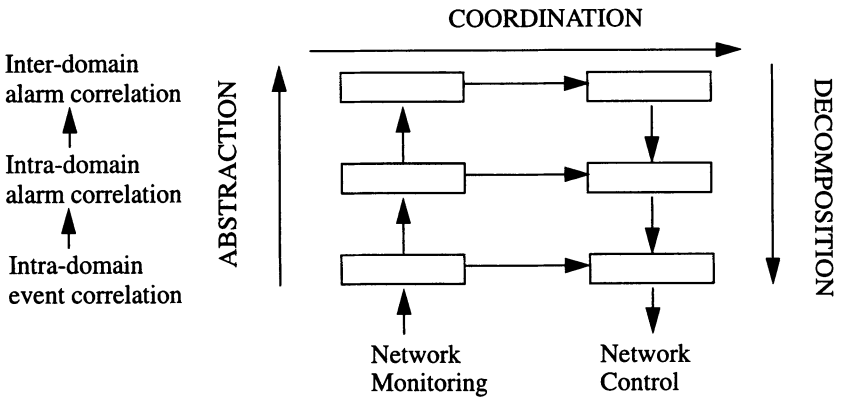

Figure 4 The Multiple-Loop Architecture for multi-domain fault management.

two levels can be performed by SSs that monitor and control individual domains in the EN. The number of SSs is the (implicit) third dimension. The agent that resides on the top level collects alarms from multiple SSs and carries out inter-domain alarm-correlation, communicating with other SS agents as appropriate. Note that the SS agents communicate indirectly via the intermediary coordination agent on the top level.

What reasoning paradigm is appropriate for the coordination agent at the top-most level? Several reasoning paradigms are at our disposal, including simple look-up tables, expert systems, case-based reasoning systems, state-transition graphs, et al. Several commercial products that incorporate some one or other of these paradigms are available. At present, we are integrating NerveCenter (which uses the state-transition graph paradigm) with Spectrum, where NerveCenter is the top-most agent [13].

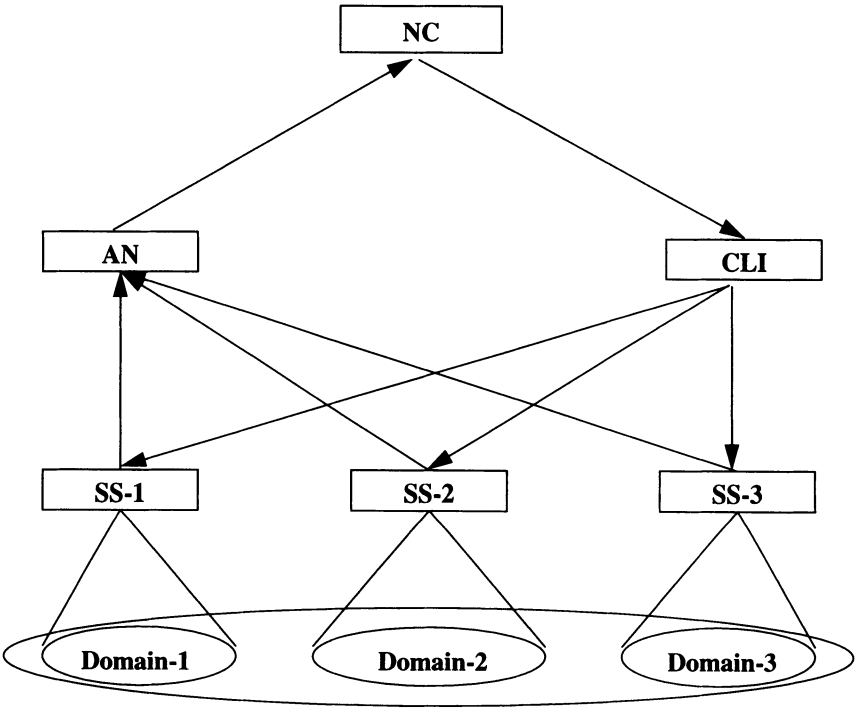

Figure 5 An integrated architecture with SPECTRUM and NerveCenter.

A clearer picture of the integration architecture is shown in Figure 5 (we have left out the SG clients). The Spectrum Alarm Notifier (AN) is a client daemon that collects alarms from all domains in the EN. The AN can be configured to allow only select alarms to be passed to NerveCenter (NC). NerveCenter performs high-level reasoning over multi-domain alarms and if appropriate, communicates with other SS agents via the Spectrum Command Line Interface (CLI). Communications may include requests for further bits of information, or notification of inter-domain alarms.

At this juncture our integration with NerveCenter is in development/testing stage. Once complete, we wish to examine the ease with which one can encode knowledge in the top-level agent. In the AI community, this task is called the problem of knowledge acquisition. For example, we may ask: How easy is it to encode new knowledge in a state-transition graph? A rule-based expert system? A case-based reasoning system? Another important issue we are studying is the ability of the top-most agent to learn and adapt itself to new problems, given its past experience [14]. This latter task is quite ambitious, but it takes us closer to the realization of an intelligent network.

3.3 Business Process Modeling

In the final analysis, the network is simply an element in a larger picture which includes all aspects of an enterprise's business processes. These processes depend on the network and its various nodes in order to successfully complete their stated goals. When a network element is not functioning properly, fault management will help us to isolate it and correct the problem, but what else was affected? A router failure may cause a marketing forecast report to fail, or a file server crash might interrupt a nightly software distribution. Let's take a simple example and see how these higher-level functions can be managed via the architecture that we've been describing.

When a standard business function such as "Accounts Receivable" is described, we discover that it is, as any business process, composed of resources and workflows. Required resource components may include client workstations, compute servers, file servers, database applications, peripherals, and the voice and data networks connecting them all together. Workflows often consist of standard procedures, contingency plans, and data flows. Human operators are also involved, but since they lack a standard management interface, they will be ignored for purposes of this discussion. Computing systems and network elements have long been manageable, and applications are becoming increasingly so via proprietary interfaces and current standards efforts [15].

We find that Inductive Modeling techniques and multi-level reasoning can also be applied to business process management. Firstly, the systems, servers, applications, PBXs, peripherals, network devices, power supplies, (and operators, for that matter) can be modeled, along with the connection and dependency relationships between them. The objective is then to monitor the presence and health of these elements and roll the collective results up to a higher level representation of the "Accounts Receivable" business process. This monitoring data can also be fed to state machines (or other techniques) used to model the workflows. Since the elements are distributed across multiple domains in a large enterprise, we must consider that the information will be coming from several agents and must be correlated, perhaps by the means described in section 3.2. With this aggregate information, it becomes possible to focus upon mission critical common resources, understand the breadth of impact when a common element experiences troubles, and proactively manage problems on a business priority basis. This topic area is the subject of additional work by the authors [16].

4 SUMMARY AND CONCLUSION

We have examined the problem of EN management from two related perspectives. We looked at ongoing work on intelligent architectures in the distributed artificial intelligence community and in the robotics community. Our view is that there are elements in this work that will contribute to our understanding and implementation of EN management software as we approach and enter the 21st century. From a practical perspective, we described working architectures for solving the problem of intra- and inter-domain event/alarm correlation and business process modeling/management in multi-domain ENs which involved integrating the Spectrum NM platform and NerveCenter from Seagate Corporation.

Acknowledgments

The authors wish to acknowledge the key engineer and architect responsible for distributed Spectrum: Eric Rustici. Additional thanks for conceptual support and contributions by David Taylor, Bill Tracy, Mike Soper, and A. J. Noushin. Special thanks to Frank Andrus and Michael Troitzsch for details on the Deutsche Telekom project.

5 REFERENCES

[1] L. Lewis. AI and Intelligent Networks in the 1990s and into the 21st Century. In *Worldwide Intelligent Systems*. Edited by J. Liebowitz and D. Prerau. IOS Press, Amsterdam. 1995.

[2] N. Sridharan. Workshop on Distributed AI. *The AI Magazine*. Summer 1987.

[3] M. Huhns (editor). *Distributed Artificial Intelligence*. Morgan Kaufman, London, 1987.

[4] K. Decker, E. Durfee, and V. Lesser. Evaluation Research in Cooperative Distributed Problem solving. Chapter 19 in *Distributed Artificial Intelligence* (Volume II) (edited by L. Gasser and M Huhns). Morgan Kaufman, London, 1989.

[5] *IEEE Communications Magazine. Special Issue: OSI Network Management Systems.* Edited by R. Pyle. May 1993 Vol. 31 No. 5.

[6] P. Maes (editor). *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back.* MIT Press, Cambridge, Mass., 1991.

[7] M. Minsky. *The Society of Mind.* Simon and Schuster, Inc. 1985.

[8] K. Olesen. Network Management in Large Networks. In *Information Networks and Data Communications II,* D. Khakhar and V. Iverson (editors). Elsevier Science Publishers, North-Holland, 1988.

[9] J. Westcott. A Simple Model for Integrating Network Management. In *Information Networks and Data Communications II,* D. Khakhar and V. Iverson (editors). Elsevier Science Publishers, North-Holland, 1988.

[10] W. Weipert. The Evolution of the Access Network in Germany. *IEEE Communications Magazine*. February 1994.

[11] R. Oliveira, D. Sidou, J. Labetoulle. *Customizing network management based on application requirements.* Proceedings, First IEEE International Workshop on Enterprise Networking (ENW-96). Dallas, June 1996.

[12] W. Hamscher, L. Console, and J. de Kleer (editors). *Readings in Model-Based Diagnosis.* Morgan Kaufmann, San Mateo, 1992.

[13] L. Lewis and A. Noushin. Outline of the Spectrum/NerveCenter Integration. Technical Note lml-aj-95-10. Cabletron Systems. 1995.

[14] L. Lewis. *Managing Computer Networks: A Case-Based Reasoning Approach.* Artech House, Boston. 1995.

[15] J. Saperia, C. Krupczak, R. Sturm, J. Weinstock, *Definitions of Managed Objects for Applications,* IETF Application MIB Working Group draft, October, 1996.

[16] L. Lewis and J. Frey. *Incorporating Business Process Management into Network and Systems Management.* Third International Symposiaum on Autonomous Decentralized Systems, Berlin, Germany, April 1997.