2

Models and Support Mechanisms for Distributed Management[1]

J.-Ch. Grégoire[2]
INRS-Télécommunications
16, pl. du Commerce, Ile des Soeurs,
Verdun, Qc, CANADA H3E 1H6
gregoire@inrs-telecom.uquebec.ca

Abstract

We describe here an experimental environment for distributed network and system administration based on the integration of a small number of simple efficient conceptual models which support a variety of management paradigms. They are implemented in turn by a couple of simple, but powerful mechanisms and a customizable runtime environment. We describe how this environment has been realized around a small and efficient language.

Keywords: distributed systems management, delegation, worm, conceptual models, implementation support architecture.

1 Introduction

Network management has received a lot of attention from standardization bodies, network and computer equipment manufacturers, and has inspired various consortiums. In most cases, network management has been handled, to a large extent, as a distributed database problem, where the management information is acquired remotely then transferred to a central location to be processed [11, 2]. The data is organized as a hierarchical distributed, potentially *object-oriented model* [3, 4]. However, even when the model is object-oriented, it nevertheless supports direct data manipulation as well as a notion of *operation*[3]. In other words, the notion of object provides inheritance of properties and granularity of concepts, but not necessarily encapsulation. It is worth noting, in this case, that the database model is not explicitly recognized as the basis for the management *mechanisms*, and little effort has been made to integrate the results of developments in distributed database technology in standards and platform alike.

The major alternative offered to the database model is a distributed object-oriented *application*. The importance of this model appears to be increasing, even though it has been pushed forward mainly by consortiums [16, 15] rather than official standardization bodies, although the conceptual influence of Open Distributed Processing (ODP) [5] must

[1] parts of this work were submitted to DSOM'94

[2] this work was partially funded by the Chaire Cyrille Duquet en Logiciels de Télécommunications

[3] note that, in this document, operation may mean an action on an object or the operations of the distributed system/network

A. S. Sethi et al. (eds.), *Integrated Network Management IV*

be acknowledged. This model supports cooperative forms of management and appears to be quite well suited for higher levels of management. Because objects tend to be large grained, and their manipulation through trading and/or brokerage mechanisms may incur a significant operational overhead, this model is not really considered for low-end operations such as data acquisition at this stage.

System management has been more the focus of individual computer systems manufacturers as well as third party suppliers. This form of management is typically aimed at system configuration and information sharing. As such, its problems are different from network management as it focuses more on *dynamic* configuration through information distribution and sharing (e.g. with Hesiod or NIS) whereas configuration in network management tends to be more static. Yet, the support mechanisms used can be related to decentralized and/or hierarchical databases. Monitoring and performance oriented operations are typically done locally.

More recent developments, mainly from private companies, have introduced distributed platforms for operations management, again using a distributed object-oriented model. These new developments bring system management more in line with the concerns of network management, and we thus feel that it is legitimate to try and unify the two notions.

Each form of management uses a unique mechanism, either a distributed database or distributed objects, to support all management tasks. This mechanism is either limited in functionality for efficiency reasons (e.g. SNMP), or turns out to be a rather heavyweight generic tool (e.g. Tivoli, ANSAware). The lack of flexibility in mechanisms leads to inflexible solutions. There are indeed few tradeoffs available in computing power and bandwidth requirements between either mechanisms.

The focus of our study is the identification of a *basic set* of conceptual mechanisms and models (paradigms) necessary and sufficient to support management tasks. Using several mechanisms, as opposed to a single, general one, allows us to have minimal structural overhead for different operations. We can also mix different levels of support for different classes of devices. Overhead indeed increases dramatically as platforms increase in complexity, with a single general mechanism.

A toolkit supporting these conceptual mechanisms allows us to fine tune the *quality of service* for different operations. Performance, availability, integrity and safety are all factors that can be taken into consideration in the selection process. These toolkit consists of a programming language and its runtime environment, which supports remote execution and dynamic interactions.

The structure of this paper is as follows. We first give some general background on network management and its terminology. We then discuss different computational structures used in, or of potential interest for network management. We then discuss another dimension of management, that is the nature of the operations that must be performed. This allows us to introduce our set of mechanisms and show how it can support the functionality required. We show how it can be used, and describe a prototype implementation. We close with a discussion and some conclusions.

2 Background

In this paper, we will be using the "standard" network management framework.

2.1 General notions

A *manager* communicates with *network elements* running *agents*. An agent interacts with the physical (or logical) process to create and maintain *managed object* abstractions. An agent can also act as a *proxy*, that is, hide and create a management compatible abstraction for parts of the network that use a different protocol.

Management is the realization of various functional categories, such as Operations, Administration, Maintenance and Provisioning (OAM&P) in the TelCo tradition, or Accounting, Fault, Configuration, Performance and Security in the OSI perspective.

Network management solutions address the problem of *network element* (or device) management. They incorporate important decisions wrt issues such as

- in band vs. out of band,
- connection based or connectionless,
- protocol efficiency and performance,
- agent resources requirements,
- manager resources requirements,
- complexity of access and manipulation of the information structure.

Network management protocols reflect a conceptual structure of managed information. The *database model* is the underlying structure in international network management standards such as SNMP or CMIS. Basically, the managed resources are treated as a collection of managed objects whose state can be queried and modified from a number of remote managers. The database model naturally suggests itself as long as one views the network as a collection of information sources to browse, and possibly to change.

SNMP, for example, is a connectionless protocol, suitable for small scale networks. Its use of polling to update the database information also generates a volume of traffic which can consume too much bandwidth as networks grow in size: a form of the so-called *probing effect* [13]. Its agents are however rather simple. Its data access paradigm is also quite simple, and consists mainly of variable manipulation. CMIP on the other hand is connection based. Its information model is richer than SNMP's and require more support from the agent. It is meant to be scalable to large networks, but it lacks, as does SNMP, a hierarchy of higher level, inter-manager, information exchange and cooperation structure.

2.2 Problems with current models

There are a number of problem with the current database approach to network management. First, the mechanism actually implemented in protocols is a restricted form of the database model, however, for efficiency reasons.

Atomicity of access is restricted to some operations when it is available at all. Operations can only be performed on a unique network element at a time. Consistency of information

retrieval across several network elements cannot therefore be guaranteed, i.e. we cannot manipulate distributed relations.

The complexity of the management work rests on one or several management station(s) which must be capable of browsing the information structure of the managed objects and recover, or modify, specific objects. Managed objects may however spontaneously notify a manager of some change in their status with *traps* or *notifications* (a notion similar to triggers in the database world).

The database model lacks notions of cooperation and grouping. There is no provision in the basic model for cooperation between managers, although the underlying mechanisms can be used to communicate information to another manager. There is also no way of grouping agents into a single element to give it a collective presentation.

In the case of in-band management, when agents have to be polled for updates, the database model may incur a significant load on the network which can be detrimental to normal operations. Scalability then becomes an important issue. Spontaneous notification mechanisms may somewhat alleviate the problem, however.

Finally, the different database models used in administration are non-hierarchical and another mechanism is required to integrate managers for domains that outgrow the model quantitatively of geographically.

2.3 Evolution

More recently, there has been a growing interest in using emerging "standard"[4] distributed OO platforms as a basis for object management or, in another case, at least to support inter-manager communication, acting as an integration platform.

In the first case, a managed entity is defined, accessed and manipulated like an object. Unlike the OSI management object model, operations are the only way to manipulate the state of an object. It is part of an object hierarchy, has an interface that defines the operations that can be performed on it and provides full encapsulation.

In the latter case, a "bridging manager" must provide a bridge between a lower level protocol's data model and the object model, and integrate their operations. The object model is used to allow cooperation between peer managers, rather than developing a manager/agent model.

Because their purpose typically is to be a general purpose communication and computation infrastructure, distributed object oriented platforms tend to carry with them unnecessary luggage in the form of features of marginal use, whose implementation, however, can negatively impact performance overhead. They provide highly flexible, dynamic communication structures whereas most of management's communication patterns tend to be fixed.

2.4 Functional Categories

Network and system management are characterized by *functional categories*, that is, a classification of the various operations which can be performed in the context of management [1]. The functionality is important to us, as it gives us indication on the respective computation

[4]standard here denotes consortium activities, or platforms inspired by ODP

and communication requirements of these classes of functions. We have thus identified four classes of support operations required to implement the functions:

- data copy (e.g. configuration),
- data retrieval (e.g. logging, accounting),
- action (e.g. diagnostic, operation),
- notification (e.g. asynchronous event reporting).

Little is new here. However, we must make and additional distinction on the nature of the communication patterns, which may be between peers or organized hierarchically. Our notion of action is also dynamic, as its effect can be modified to reflect the changing nature of the network. Similarly, notifications, as they result from actions, can also be added dynamically to a system.

This perspective allows us to look more closely at the nature of structural support that is required for different functional categories. Of course, orthogonal to these classes, we have further parameters to take into account such as volume of information, atomicity or distributed actions, but we should not forget that the use of mechanisms becomes more marginal as they get more sophisticated. Furthermore, as is already done in some cases, separate, dedicated, protocols can be used to support very specific, demanding management operations, such as, say, bulk transfer. We shall refine this classification in the next section.

3 A new approach to distributed management

We are building a management environment for networks and applications based on a collection of conceptual mechanisms, such as:

- basic access,
- delegation,
- worm,
- cooperation,
- notification.

These conceptual mechanisms are supported by a *remote execution* and a *local interaction* mechanism.

3.1 Conceptual mechanisms

3.1.1 Basic access

We call *basic access* the simplest general support mechanism. It enables the configuration of the device, as well as accounting operations. It allows the reading, retrieving and modifying chunks or pieces of information. This is the major functionality provided by database-like mechanisms.

3.1.2 Delegation

Delegation is operation and diagnostic oriented. Delegation allows us to dynamically expand the functionality of the network element by transferring executable code to it [8, 17]. This code can either execute a function locally and report back its results, or create a higher level object which can be queried by other mechanisms. Delegation helps to regroup a set of operations on several objects into a single action.

Delegation has several benefits. Delegated management operations are executed locally one the network element, but in a flexible way as the operation can be modified dynamically at any time. It contributes to reducing the bandwidth required, as well as decreasing the latency in the discovery of potential problems and the execution of remedial actions.

3.1.3 Worm

The worm is a recursive form of delegation. In the pursuit of the root of a problem, it can be necessary to trace its symptoms across different machines. When the diagnostic is performed by browsing from machine to machine, a worm can be used to implement the procedure.

A worm can also be used for configuration and accounting style operations for a range of machines. It can also implement features such as topology discovery.

3.1.4 Cooperation

Cooperation is the interaction of several managed objects to achieve a collective modification. It is a peer to peer model, as opposed to the hierarchical function/library model.

The activities of the program are the result of the cooperation of several programs, rather than a single one.

3.1.5 Notification

A notification is an asynchronous, or rather unsolicited, message sent to signal an important change in the NE.

A notification can be sent to a manager, or to another NE.

3.2 Support mechanisms

3.2.1 Remote execution

The technique of remote execution simply means to transfer a program to a machine where it can be repeatedly executed. The transfer process must take care of architectural differences and manage an output channel to a manager.

Remote execution depends on the availability of a core functionality, such as access to management information, on the target platform. It requires an execution mechanism, remotely accessible which must also be reflected in the management information model. It requires a support language in which the management functions can be expressed and also has a type system rich enough to capture the details of the conceptual model.

Run-time safety is a prime concern. We want to guarantee that a program will not fail at run time. For most operations, this can be achieved with a type-safe language, with functional, rather than imperative, characteristics. Type-safe compilation and linking should guarantee that the data is available in the NE interface, represented as a library. A functional language has simple recursive data structures which are safer to manipulate than pointer-based structures.

Remote execution implements basic access, delegation and worm. It supports notification.

A program is the largest grain of atomicity provided in the model.

3.2.2 Interaction

Interactions exist at two different levels: either between co-resident or between remote (e.g. on a manager station) programs.

Co-resident interaction can be handled through a simple type message passing interface. An interface must be defined for every type of communication. Two partners exchanging information can exchange some form of token to guarantee that they are using the right interface, as is done in presentation layer negotiation schemes. Remote interaction can be treated as a combination of remote execution and co-resident interaction.

Interaction implements cooperation and support notification in its remote form. The managers must have an interface to capture the interactions.

3.3 Management environment

Since our work is experimental in nature, we are aiming at simplicity and flexibility in the construction of the management support environment. The major complexity of implementing administration with our mechanisms is that, since they are at a lower conceptual level, i.e. they act as enabling mechanisms, and their access is language-based, operations may require some programming. One should note, however, that our mechanism can be enabled by management platform technology similar to what is in use in the industry. Graphical browsers and mouse-based operations activation can hide the assembly, compiling and transfer of a program. By using a lightweight, efficient interpreter environment, the compilation and linking overhead can be kept to a minimum and close to performance levels similar to marshalling/unmarshalling operation times. The information structure can be mapped from a conceptual object oriented structure to the type system of the programming language.

3.4 Complementary mechanisms

Other mechanisms that we have to consider to expand our capability are a mass transfer mechanism, and a multi-way communication structure.

The first one is definitely useful to retrieve, typically, logging or accounting information. In the telecommunication industry, this is done with a different file oriented transfer protocol, such as FTAM.

A multi-way communication structure is a simple way to share information between different parties. Combined with a causal communication structure [6], we can build globally consistent information updates and built consistent views of parts of the network. Such

an infrastructure has proved useful to implement distributed monitoring [13], but it has a significant overhead, however, and would be best done by a dedicated, separate structure, installed only as required.

4 Implementation

We have built an experimental delegation/worm environment at INRS-Télécommunications [9, 7]. It is a lightweight environment, flexible and quite suitable for experimentation. It is smaller in size of code and runtime image than the SNMP libraries and SNMP agents we have studied[5].

The environment was built around the CAML language and the CAML-LIGHT virtual machine [12]. This pragmatic, (mostly) functional language has most of the features we required, namely strong, polymorphic typing, separate compilation, an exception mechanism and a rich data model. Its implementation gives us ease of extension, portability, architecture-dependent conversions postponed to linkage time, and a compiler/virtual machine implementation. We have added to it multithreading, that is, the capacity of executing several CAML programs concurrently with preemption, remote loading of compiled code, remote control and monitoring of the threads, inter-thread communications, remote linking and a worm mechanism. The data model of the language is rich, dynamic and flexible and it has been proved to be capable of emulating OO structures.

The interface to managed objects is done through an encapsulated, typed interface (an abstract data type). An interface defines the structure of the information and the operations which can manipulate it. The virtual machine is responsible for retrieving the information relevant to all managed objects and updates the corresponding data structures at regular intervals, as required by the applications. The virtual machine also supports atomicity of access and manipulation to managed objects. It is possible to write different interfaces to the same objects, for different access rights. The interface one uses thus limits the manipulation of the data. The management of access rights is done entirely out of our model. If necessary, the communications between platforms could be encoded, although we have not implemented it.

Interaction between threads is done through type-safe interfaces, implemented using techniques similar to marshalling. Unfortunately, because it uses compilation, the CAML-LIGHT environment does not keep type information at run-time, and we had to introduce our own mechanism. These interfaces are available only locally. For two threads running on different machines to interact, a intermediate, interaction thread must be transferred to the machine where the interaction will occur. The use of such intermediate interaction threads is hidden in communications libraries.

Any administrative task is implemented by a piece of code. This code is compiled from the administration environment, transferred to the target machine where it is linked and executed as a thread. Libraries of executable threads can be managed on the target machines, if the memory is available. Similarly, libraries of precompiled tasks can be stored in the administration environment and transmitted as required. More importantly, each virtual

[5]typically the ISODE snmp and the CMU packages

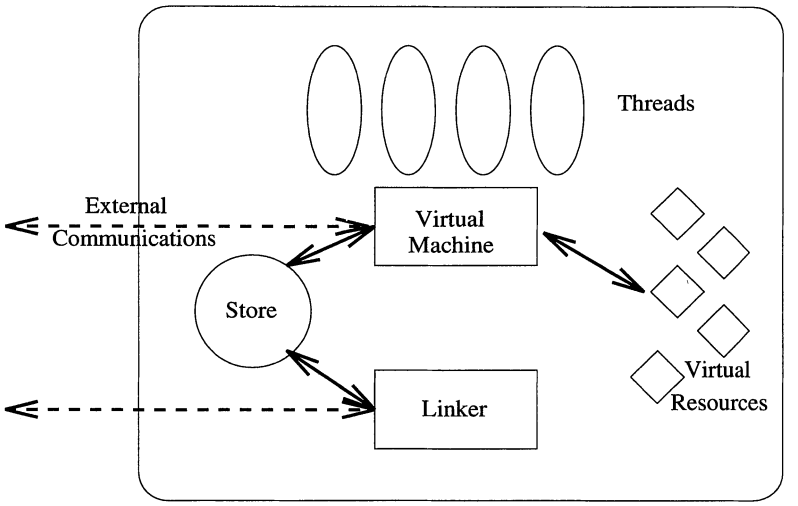

Figure 1: The elements of the management environment

machine stores the libraries which give access to the managed objects abstractions, with which threads have to be linked. Only the interfaces definitions for these libraries need to be available to compile a thread on a manager's site.

Each thread can be activated with specific information, in the form of run-time arguments. Each thread has a "log" channel to recover error information. Another channel recovers normal output. These channels are set up dynamically for each thread and, similarly, each thread can report to a different manager.

Figure 1 illustrates the general structure of the management environment.

We use this environment to remotely manage our distributed heterogeneous workstation environment. We have built an interface to the Unix kernel for system monitoring and we use Unix commands to carry operations. We also have integrated a SNMP access mechanism. Worms have been used to track users, implement load balancing and experiment with several distributed algorithms. We have also replaced the distributed configuration environment of our workstations by a local control managed by delegation. Several forms of resources management are also executed locally in this environment. In this context, distributed management follows the "think globally, act locally" philosophy.

5 Discussion

In our vision of distributed management, all NE's should support remote execution. Local interaction would come second on our list. The resources of the devices could limit the number of resident and active programs, with the possible effect of increasing latency in the

response to some operations.

Our perspective, however, is that not all NE's need to support the whole set of mechanisms. By keeping them orthogonal, we can limit the impact of their combination on performance, memory and CPU requirements. It is even possible to cross compile for a machine which doesn't have enough memory for a linker, or enough storage for libraries. There are in fact a wide range of quality of service factors which can be tuned *independently of our conceptual mechanisms.* Different transport protocols can be used, storage can be offloaded to another machine; it is even possible to fully compile a (then) static application where performance can be critical.

One interesting feature of our mechanisms is that they can be used recursively across hierarchical administration domains. The remote execution and interaction mechanisms make no provision on the peer to peer or master/servant nature of the communication relationship. It is straightforward for an agent station to become a server provided it can store the code for the tasks under its control. The remote execution mechanism is directly accessible to the threads.

In our experience, the overhead of the transmission of threads is low. For simple operations, a thread fits in a single packet. The main cost of the execution of threads on the virtual machine is in memory management. We have had to tune a garbage collector to optimize the management of the memory of on-shot threads vs recurring threads. In fact, specific algorithms can be used depending on the degree of sophistication of the operations performed by the agent (i.e. hub vs workstation vs management platform).

Further mechanisms such as file transfer would be left out of the virtual machine. In that respect, it is worth pointing out the fact that distributed network applications coexist with distributed management. Typical examples are distributed network reconfiguration and, more generally routing. Although such mechanisms can be implemented using our mechanisms, they tend to be either integrated in a low level protocol, or are realized with dedicated links. What is at stake here is a tradeoff between flexibility and efficiency. Dedicated mechanisms potentially avoid information extraction and conversion overhead, at the cost of flexibility. Traditionally, real-time applications (e.g. routing) have used dedicated mechanisms whereas less time critical applications (e.g. on-line diagnostic) have used more generic mechanisms, and also potentially more computationally intensive (e.g. AI search techniques) techniques.

6 Comparison with other work

The techniques we have described here have been pursued in various guises, but, to our knowledge, never in a similar integrated context of a toolkit of complementary mechanisms, supported by a programmable environment.

Delegation of duties has been studied both from a operations and an administrative point of view [14]. One application of delegation in a standard's framework is the definition and implementation of higher level managed objects, which compute some chosen function based on the values of other objects. Programmable area managers are another example of delegation of operations. An area manager is responsible for a small network of - say - SNMP managed devices. The area manager is programmable and can perform tasks delegated from

a higher level manager, through a suitable, but different protocol. MINERVA [10] is such an environment, where local changes of interest, monitored through SNMP, are reflected into events which in turn trigger the execution of *scripts*, written in a custom language. *Empirical Tools and Technologies*[6] is a commercial company which sells a manager which can execute SCHEME programs which can be remotely downloaded. Let us notice here that SCHEME is not as safe a language as CAML and the risk of occurrence of run-time errors is significantly higher.

The AI notion of *agents* is also similar to our concepts of recursive remote execution, as used by worms. The use of AI agents for distributed network management has been suggested recently by different researchers. In that work, the use of agents to study and improve routing is described. Although such work is usually done by more efficient mechanisms, worms could be programmed to realize such a task.

In spite of the similarities in concepts however, we haven't found anywhere an attempt at providing scalable mechanisms, and to provide a uniform view and uniform support for the manager/NE universe.

7 Conclusions

We have described a perspective of enabling management through a set of simple conceptual mechanisms, rather than a single high level one, and described a management environment based on remote execution and interaction. These mechanisms support a number of paradigms well suited for network and distributed application management. These mechanisms have been implemented in a programming language-based environment. Complementary mechanisms such as file transfer can be done efficiently using a dedicated protocol outside of this environment.

In practice, there seem to already exist a few commercial tools which follow our philosophy of combining several mechanisms, including a form of remote execution, in their management environment. However, they all tend to support a single layer in the management hierarchy and do not share our vision of the recursive application of similar concepts with tradeoffs with regard to the quality of service.

The major benefit that we see in using remote execution as opposed to a database mechanism is the integration of a computational and a data models which allow us to uniformly manipulate the data as well as retrieving it.

Since our focus was on low level enabling mechanisms, there are a large number of concerns that we haven't covered in this short presentation, such as higher level of management coordination, domains and policies, etc. We are currently studying the requirements of the management platform with these considerations in mind.

Acknowledgments.
The development of the distributed platform has been done by F. Gagnon.
N. Greene and F. Gagnon have provided helpful feedback on various drafts of this paper.

[6] this information is based on an exchange with K. Auerbach

References

[1] CCITT Recommendation X.700— ISO/IEC 7498-4: 1992, *Information Technology - Open Systems Interconnection - Management Framework for Open System Interconnection.*

[2] CCITT Recommendation X.711— ISO/IEC 9596-1: 1992, *Information Technology - Open Systems Interconnection - Common Management Information Protocol, part 1: Specification.*

[3] CCITT Recommendation X.720— ISO/IEC 10165-1: 1992, *Information Technology - Open Systems Interconnection - Structure of management information, part 1: Management information model.*

[4] CCITT Recommendation X.722— ISO/IEC 10165-4: 1992, *Information Technology - Open Systems Interconnection - Structure of management information, part 5: Guidelines for the definition of managed objects.*

[5] CCITT Recommendation X.901—ISO/IEC 10746-1 *Basic Reference Model for Open Distributed Processing- Part 1: Overview and guide to use*, 1993

[6] O. Babaoglu and K. Marzullo, *Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms*, in "Distributed Systems", E. Müllender, Ed., 2nd Edition, Addison Wesley, 1993.

[7] J-Ch. Grégoire, *Delegation: Uniformity in Heterogeneous Distributed Administration*, LISA VII, Monterey, California, 1993.

[8] J-Ch. Grégoire, *Management with Delegation*, IFIP'93, AIPs Techniques for LAN and MAN Management, Paris, France, 1993.

[9] J-Ch. Grégoire, F. Gagnon, *Implementation of Delegation in Distributed Network Administration*, Canadian Conference on Electrical and Computer Engineering, Vancouver, Canada, 1993.

[10] D.J. Hughes, Z.D. Wu, *Minerva: An Event Based Model for Extensible Network Management*, Proceedings of INET'93, pp. CEC-1—CEC-6.

[11] Internet RFC 1157, *A Simple Network Management Protocol (SNMP)*, 1990.

[12] X. Leroy, "The Caml Light system documentation and user's manual", version 0.6, INRIA, 1993.

[13] M. Mansouri-Samani, M. Sloman, *Monitoring Distributed Systems*, Chap. 12 in *Network and Distributed Systems Management* M. Sloman, Ed., Addison Wesley, 1994.

[14] J.D. Moffett, M.S. Sloman, *Delegation of Authority*, I. Krishnan & W. Zimmer (eds), *Integrated Network Management II*, North Holland (1991), pp 595-606.

[15] Object Management Group, *Common Object Request Broker*, 1992.

[16] Open Software Foundation, *Distributed Management Environment*, 1991.

[17] Y. Yemini, G. Goldszmidt and S. Yemini, *Network management by delegation*, Integrated Network Management II, Elsevier Science Publishers, pp. 95–107, 1991.