A Scalable Approach to Automated Service Dependency Modeling in Heterogeneous Environments

Christian Ensel*
Munich Network Management Team
University of Munich, Dept. of CS
Oettingenstr. 67, 80538 Munich, Germany
phone: +49-89-2178-2171, fax: -2262
email: ensel@informatik.uni-muenchen.de

Abstract

This paper presents a new methodology to automatically generate service dependency models. It strives to enable more comprehensive IT–management by providing an always up-to-date information basis about inter–dependencies of IT–services and other management objects like services applications and network components. The approach specially aims for heterogeneous environments as found in large enterprises and outsourcing scenarios. It comes with an agent based implementation architecture allowing its smooth integration into existing IT–infrastructures.

Keywords

dependency models, automated model creation, IT-management, agent architecture, neural networks

1. Introduction

Without doubt, IT–management has made enormous progress over the last years. It evolved from its early focus on the network and systems layers to the management of complex services that takes into account special requirements coming from enterprise wide business processes and service outsourcing to external partners. Current management architectures and their realizations in platforms and tools provide standardized remote access to a wide range of objects in the managed environment [1]. But still there exists a number of management tools based on proprietary resource interfaces. And worse, important management information is often only available through those non–standardized interfaces or even not at all. This is especially true for information about dependencies between those managed objects, although this would be required for the aforementioned higher level management. The knowledge about dependencies may still look simple on the level of contracts, but as soon as IT–infrastructures of different organization overlap—which is a pre-requisite for outsourcing scenarios—the situation becomes much more complex.

*The author's work was sponsored by Siemens AG, Zentralabteilung Technik, Fachzentrum Neuroinformatik in the context of the project "LEONET" of the German Ministry for Education and Research (BMBF).

Often, each administrator can only concentrate on one part whereas the 'big picture' of the overall dependencies and interconnections gets lost.

Figure 1 shows a small scenario with two domains (`L` and `R`). One hosts a web server `WS` and a database server `DB` providing content for the web pages. The other contains the Domain Name System `DNS` responsible to resolve the name of both web– and database server. Thus, `WS` is said to depend on `DNS`. Further, there are two users who typically access information on the web server via web clients. They depend on `WS` and—if they want to type normal URLs instead of IP addresses—also on `DNS`.

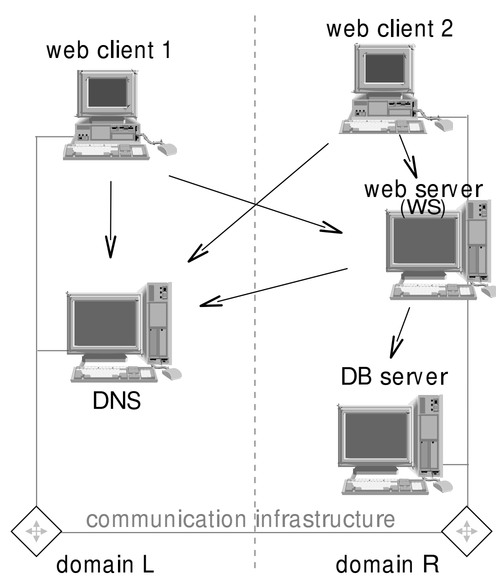

Figure 1. Simple multi service scenario

The figure also depicts the mentioned dependencies between its major objects. For simplicity, it neglects all dependencies to the communication infrastructure and further sub–services. One can see that although several objects of the example depend on `DNS`, none of their existing implementations explicitly tells to do so and cannot be queried by management applications for this fact.

As there is no support to obtain information about dependencies by nowadays management tools, conventional approaches generally struggle with evaluation problems, e.g., of configuration files (for an overview see section 3.2). The problem becomes worse if, e.g., the format of those files changes with software updates etc. In heterogeneous environments such modeling systems typically must further be restricted to a very limited set of resource types or vendors. The major alternative to dependency detection at runtime would be an a priori description of the environment with its components and dependencies, similar to what is achieved in software engineering by the help of Architecture Description Languages (ADL) and Module Interconnection Languages (MIL), respectively. However, so far similar dependency description languages have not been commonly agreed on in the IT–network and service provisioning world.

As a consequence, dependency models are not generally used in today's management world—although their benefits are commonly known (section 2 explains existing applications). This fact leads to a lack of overview for the IT-administrators and prevents the use of powerful management tools like event correlators that are based on dependency models [2]. More applications are described in section 2 together with an overview of existing types of dependency models.

Typical enterprise scenarios will of course be much larger than the example above. In such cases it would be hard to keep the overview even with the help of dependency models. To reduce the number of elements, respectively to restrict the model to currently interesting parts, the concept of domains is used to provide a basic means of structuring models. This allows to provide an overview on higher, e.g., business process oriented levels, and enhances visualization and understandability for the IT–managers working with them—without abandoning details on lower levels needed for proper fault diagnosis. Graphical user interfaces of management tools will typically be capable of folding and unfolding domains to navigate into sub–models, thus handling scalability of their visualization. However, this concept also needs support from the underlying modeling architecture.

To overcome the two main problems of automated modeling—the lack of direct dependency information as well as the scalability issues—this paper presents a new approach to gain management relevant dependency information. Unlike conventional approaches it is designed to obtain useful results independent of the heterogeneity of the managed environment. It is based on two key parts. The first (covered by section 4.1) are the underlying concepts of dependency determination that are carried out with the help of neural networks. However, this paper does not aim at details about artificial intelligence like, e.g., the training methods of our neural networks, but concentrates on modeling and realization aspects relevant for IT–management. Thus, the second part (section 4.2) deals with questions of installation efforts and scalability to seamlessly integrate the modeling into real IT–environments.

Section 2 introduces the most important types of models and shows their existing applications. This is followed by an overview of the *state of the art* of approaches to model creation in section 3. They are analysed by the help of a generic modeling process which also leads to a complete set of *requirements*. Our solution to meet those requirements is presented in section 4. Section 5 draws the conclusions of the paper.

2. Dependency Information Models and Applications

Modeling dependencies of managed objects has to be seen in the context of the description of management information in general which has always been an important part in the definition of management architectures like OSI (Open Systems Interconnection) management standardized by the ISO [3]. The syntax and semantics are defined in the so called *information model*; its access in the *communication model*. The focus of management information traditionally lay on attributes and properties of single objects. Although, e.g. OSI management already defined a General Relationship Model (GRM, [4]), it was never widely used in IT–management. Only in recent years the issue regained more interest [5]—mainly with the increasing number and complexity of the inter–service, –system and –domain dependencies.

Examples for typical information specific to single managed objects are variables of the Management Information Bases (MIBs, [6]) defined in the Internet Management Architecture, like ...`mib-2.system.sysLocation` that is defined to store the system's location. This kind of information is either stored at the real objects (hardware components, applications, etc.) and accessible via management agents using standardized protocols or within the corresponding object representation at the management tools.

The following subsections focus on another aspect of modeling management information: the so called *dependency models*. Various types of such models are illustrated together with their applications. Although most model types could also be used for the management of lower (communication) layers, they are analysed with regard to service dependencies, i.e., we leave aside models at the lower OSI layers, like network topologies. Of course, knowledge about the underlying communication structures is also essential, e.g., to diagnose problems or to identify bottlenecks, but this has to be (and to a satisfactory part already is) carried out by very different techniques than the ones needed on the level of end user– and supplementary services, with a much higher degree of complexity and dynamics.

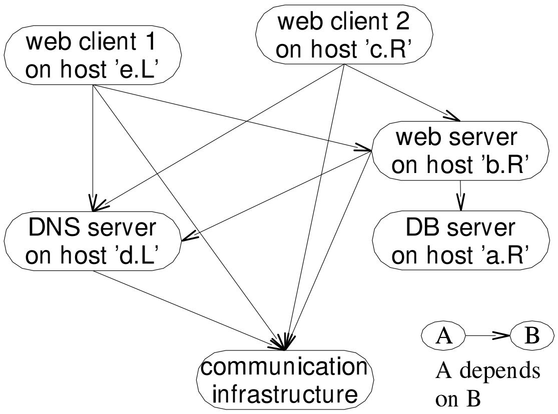

Figure 2. Simple environmental service dependency graph

2.1. Environmental Models

The description of the scenario in the introduction already comprised the objects' dependencies. These immediately lead to the dependency model of figure 2 depicting the web server with its database, both web clients, the DNS server and a generalizing object for the common communication infrastructure. In the following, such models are called *environmental models* to stress their capability to reflect information specific to real objects in the managed IT–environment (in contrast to the abstract models explained further down).

Each node represents one single or alternatively a group of real objects, which may be enriched with further management information—corresponding with traditional management variables in other information models. The group– or *collective objects* are used in various cases: They represent whole domains (e.g., organizational units like departments or responsibility areas of IT–managers) or—on a more technical level—distributed applications that are intended to appear in the model as a whole, but not in full detail. Figure 3 depicts such a domain object, by hiding the whole domain `R` under one node. It could, e.g., be the model that is displayed to the administrator of domain `L` who is not allowed to look at details beyond his own area, but nevertheless is interested in dependencies to the outside world. Domains are represented by one element in the model, which is either a terminal element or stands for (and may be expanded to) collections of one ore more:

- simple (terminal) objects also taken from the modeled environment, and/or
- underlying (sub-)domains, recursively providing more fine grained levels of detail.

[7] presents further details on the subject of domains. As they obviously are an important concept, the architecture presented in section 4.2 also includes the appropriate features.

The second type of components in the models are (directed) edges representing dependencies between the nodes. For some applications of the models, undirected graphs are sufficient. For others it is useful to attach further management relevant attributes, e.g.:

- to form groups which, e.g., express that a dependency only occurs together with others,
- to express that some dependencies must occur in a certain timely order,
- to attach values of strength or likelihood, but also
- to reflect 'internal' management information, e.g., how the dependency was detected or how much effort its reevaluation would take.

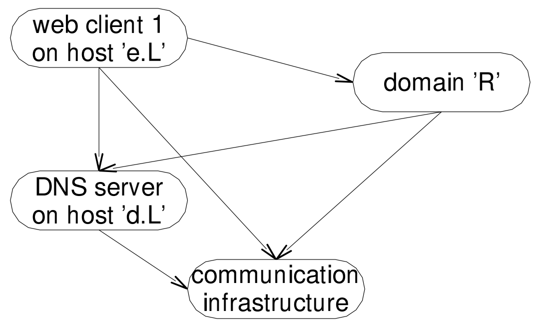

Figure 3. Management perspective from domain L

If `DNS` in our example fails, web clients in principal are still useable by typing IP addresses. This restriction in the quality of service could be denoted as an attribute of the dependency between the clients and the `DNS` server.

Applications

Utilizations of models can be found in several research projects. One application is the so called *root cause analysis*. It helps to find a common (root) cause of problems or faults detected at distinct places within an environment. It may be applied to network components reporting error conditions as well as to services, where end users detect problems. The reason for the actual need of such root cause analysis is that error conditions or problem reports brought to administrators or management systems, are just descriptions of symptoms. To be able to derive their causes, further knowledge about the dependencies among them is necessary. [8], [9] and [10] explain this subject in detail.

Similar dependency models are needed when *determining availability requirements* on sub–services (looking from a top down perspective) respectively for the *calculation of service availability* from the availability of underlying services (bottom up), as described in [11].

Knowledge of dependencies between systems may be of further use for the *prediction of impacts* on other systems due to management operations. This is of particular interest in typical maintenance scenarios, where a server has to be shut down temporarily: It is essential to know, respectively to simulate the effects on other systems beforehand. Further investigations of advantages can be found in [12] and [13]. A common result of their and others' examinations is that—assuming models do already exist—great benefits can be achieved for management tasks. For our purposes, following major advantages for the practical utilization of environmental models can be resumed: they

- are not restricted to special types of objects (e.g., hosts, applications, services and comprehensive objects like *communication infrastructure*),
- provide overview to IT-administrators on selectable levels of details,
- support for more intelligent management tools.

More applications of environmental models emerge if the algorithm used for their generation allows—like the one presented in section 4—frequent iteration of the modeling process in certain time intervals. This enables the analysis of changes of dependencies in the managed environment during that time. This is, e.g., useful for *fault prediction*, because significant changes in the overall system behavior are detected through emerging or disappearing dependencies. This often reflects errors that are already present in currently unused parts of a service which may later (under different usage conditions) effect its usability. The detected changes may also be used to point out forbidden actions or disallowed use of services. This is helpful especially for *intrusion detection* and to *recognize service misuse*.

2.2. Abstract Models

In contrast to the environmental models dealing with objects directly mappable to real world components, the main elements of abstract models are classes providing an abstraction of the specialties of real environments. The dependencies between those classes are also specified on that level of abstraction. It is easy to see that environmental models actually are instantiations of abstract models, in regard to objects as well as their dependencies.

Abstract models are normally generated manually—either by the vendor of the corresponding objects in the real world (e.g., the developer of an application provides a model showing the dependencies to other applications and the underlying system), or by the suppliers of management tools that are based on reasoning on such models. Just like environmental models, abstract models are suitable to express knowledge about higher layers, e.g., to model services. They do not depend on environmental specifics, but only express general or principal dependencies.

The model shown in figure 4 looks similar to the previous one. However—as each node now depicts a class—both web clients are covered by a single *"web client"* element. Following object oriented principles, the nodes' and edges' attributes are now replaced by definitions for allowed, respectively needed attributes.

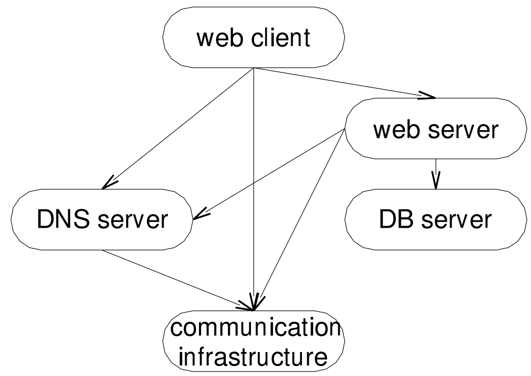

Figure 4. Simple abstract service dependency graph

Applications

In principal, the applications are similar to the ones described in the previous subsection, with the difference that abstract models are:

- partly constructible before their actual application on real environments (e.g., by the service vendors),
- much smaller and simpler to handle.

However, the models' restriction to the abstract level of course has implications on their usability. This problem is typically circumvented by one of two methods: *Virtual instantiation*, keeps lists of real objects (together with the object specific management information) for each class, but the inference engine's main work is carried out on the abstract models. *Full instantiation* maps the abstract to environmental models. In this case the former are used to add commonly known dependency information to specific scenarios, but after their instantiation they are not directly operated on by the management tools.

More direct use of abstract models has been carried out in several *Model Based Reasoning* (MBR, [12]) systems, for a number of years. These have already been able to map the results of errors in simple components to services or systems visible to end users. Another application is to diagnose possible sources of errors on lower layers, if problems on higher ones are reported. Still, management tools based on abstract models so far have not been very successful with regard to their market share. The number of available models remained too small to let the strengths of the tools become fully visible: On the one hand, companies delivering products do not want to enclose models due to the extra efforts needed and because of strategic policies enforced to protect the companies' market positions (e.g., confidentiality). On the other hand, it is also virtually impossible for the providers of the management tools to supply sufficient models themselves.

As a partial solution to this problem a standardized and widely accepted library for the most important classes could be used. In the past, efforts to do so have not been very successful. However, the endeavors of the Distributed Management Task Force (DMTF) for the Common Information Model (CIM, [14]), where esp. the Common Schemas are able to serve as a basis for further abstract models, hopefully will be more successful. The following subsection provides more information on models similar to CIM.

2.3. Object Models

To obtain a complete picture one also has to look at object oriented approaches. Standardized object oriented modeling of management information exists at least since the definition of OSI management. An example for a vendor specific object oriented management model is the one used in Cabletron Spectrum [15]. It was introduced to overcome problems of the Internet Management which lacks a similarly powerful information model.

Newer endeavors led to the aforementioned CIM specified by the DMTF, a federation of many leading companies in the areas of computer systems, software and networks. It is an information modeling and representation schema widely accepted by the industry. The CIM Specification provides a "Meta Schema", a specification language called "Managed Object Format" (MOF) and mappings to other information models. Details can be found in the CIM Specification 2.2 [14]. CIM further includes a set of pre–defined basic schemas, defining fundamental classes like *"System"* (in the "Core Schema") and classes specific to certain areas, like *"Rack"* in the Common Schema "Physical". In addition to the definition of an appropriate set of general attributes and an inheritance hierarchy it also allows the modeling of arbitrary dependencies between classes, respectively objects. Thus, CIM provides concepts for descriptions on the abstract modeling level and means to instantiate, represent and exchange environmental models. However, for dependency determination and model generation CIM also needs to be supplemented with further algorithms.

3. Principles of Dependency Model Creation

This section shows how we subdivide the overall 'lifecycle' of dependency models—from their conception to their utilization—into phases and explains the subtasks that have to be carried out in those phases, together with an overview of the state of the art of conventional modeling approaches. The structuring of the process further helps to understand the requirements on automated modeling and how our new method (described in the next section) provides solutions for them.

The main phases are:

1. object selection
2. tool installation
3. model creation
4. model based management

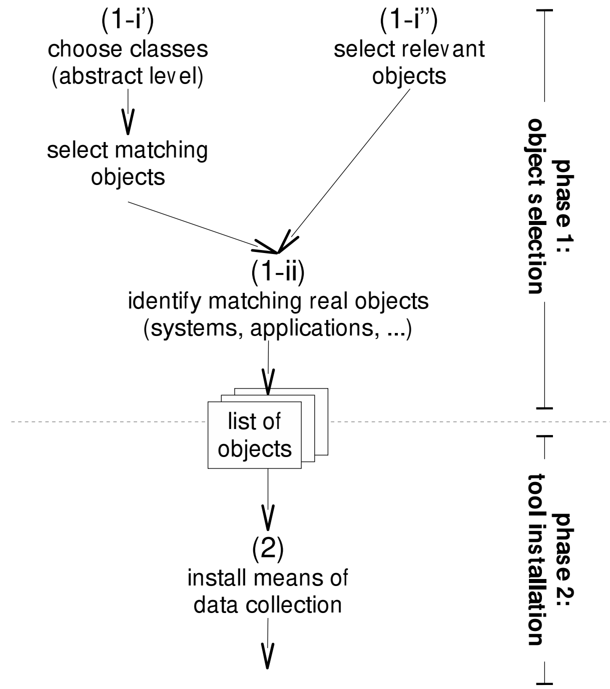

Figure 5. Two phases of preparation

The first two phases are needed as preparation for the actual automated model generation, while the last two take place at operation time.

3.1. Preparation

The first phase needs to be carried out per dependency model. It selects the objects that are finally going to be represented in the model. Most basic choices are:

- the desired level within the management hierarchy: e.g., high level services, applications and their components, network systems, or any other level,
- the object type: e.g., all available services vs. a certain selection of classes,
- the model's main range: restricts the model, e.g., to objects from one certain department or area of responsibility,
- the models border area: in addition to the main range the model may also enclose collective objects representing things outside that range, like *internet* or the *domain R* in figure 3 which are represented in the model just like any other object.

Obviously the selection heavily depends on the purpose that the model will be used for. Figure 5 further shows two choices on which level the selection process of step 1 may take place:

i') Either on the abstract (class-) level, e.g., *"web servers"* and *"web clients"* followed by (possibly platform supported, automated) instantiation, to select the real objects belonging to the chosen classes.

ii'') Or directly on the level of objects in the real world, e.g., *web server on host b.R*, *web client on host e.L* etc.

So far, the selection of objects followed a top–down approach. However, one also has to consider restrictions coming from the modeled environment and the modeling tools used in the later phases, because these may only accept restricted types of input data that is not available for all objects. Especially collective objects for domains are usually not supported by current automated modeling tools. Section 4 will therefore also show how our approach deals with this problem.

In step 1-ii the chosen objects must be mapped onto the matching components, applications etc. in the real environment. This requires no special actions for simple objects but for those where the realization is dispersed over distinct real objects, e.g., distributed applications and also the previously mentioned domains. In our example, two main routers could be selected to represent the object *communication infrastructure*.

One can see that this phase is only semi–automatable through the help of systems management or workflow utilities which are able to provide lists of available services, domains and departments etc. The assessment whether those objects should take part in the model must be done by human administrators.

In phase 2 the appropriate probes to meter the objects' activities (as explained in section 4.2) must be installed. As for all measurements in distributed environments special care has to be taken on where to place the means of collection and the model generating algorithms; esp. in the TCP/IP world, where the management data is transferred 'inband' through the same channels as the user data.

In some cases it is necessary to create multiple models for the same environment, e.g., due to the application of distinct management tools or overlapping modeling of different organizational areas. Therefore, it is an important requirement on the means of information collection to be able to install them independently from a single model creation process, thus making them and their collected data reusable.

3.2. Generation and Management

During normal operation of the systems and networks, activity data is collected (step 3-i, figure 6). After a certain period of time the information is transferred to places where the models are generated. The type of information gathered in this step totally depends on the method used in step 3-ii which is the most challenging task of the process: the actual automated generation of models. Several methods have evolved to support or replace manual generation which mainly suffers from the enormous effort it imposes on the administrators during actual model creation and also to keep them up-to-date. Alternative automated approaches are:

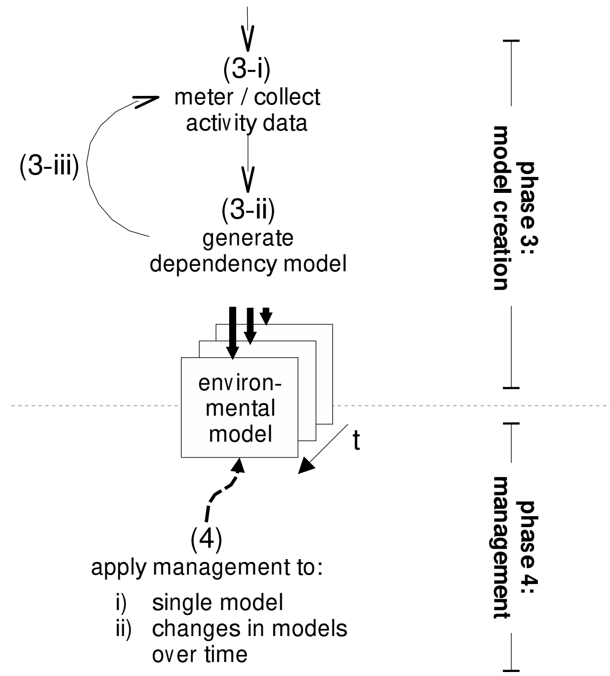

Figure 6. Phases for model generation and management

Service instrumentation: The modeling process would be most easy if the modeled objects provided appropriate instrumentation by themselves. The problem is that—although CIM and other information models theoretically have appropriate description mechanisms—none of today's applications is able to provide this kind of information at an acceptable granularity. Most do not even allow to be queried for any dependency information at all.

Evaluation of files at dependents' side: A straightforward method to determine the dependencies is to look for configuration data at the dependents (i.e., the objects playing the client role in a client–server relationship) directly expressing on what services they depend. E.g., `WS`' dependencies (in the introductions' example) cannot be queried directly, but are hidden in its configuration file that mentions the database server by name instead of by IP address, plus the fact that this host name is *not* listed in local name resolving files like "/etc/hosts" on typical Unix systems. One can already imagine how hard an automated detection of dependencies by looking at configuration files would be. The case would become even more complicated if host name and IP address indeed *are* listed in "/etc/hosts", because it would then be up to a third file to determine whether local resolving is carried out at all. Similar problems exist for the following case.

Evaluation of files at antecedents' side: Similar to the previous one this modeling method analyses files. However, it looks for indications of service utilization (by clients) at service side, e.g., in a server's log files. In the example of figure 1 the web server's log file would contain entries showing that both clients had connections. In other words, both have a dependency on `WS`.

However, a major drawback of all approaches that analyse log or configuration files is that these typically have a proprietary format or sometimes change with software updates. Even worse, not all applications provide log files or similar mechanisms containing this information; also access to files may be restricted for several other reasons, like security policies.

Exploiting software repositories: A new extension to these approaches is described in [16]. Additionally to the analysis of application configurations it makes use of information stored in software installation repositories of operation systems to obtain intra–system relationships.

In every case the result of this step is one dependency model reflecting the current stage of the environment. Step 3-iii indicates the reiteration of the previous two steps, which leads to a sequence of models being generated over time. The appropriate duration of the time period between the models also depends on the modeling approach and application. Looking at configuration files might be needed only about once a day, while our approach allows (if needed) to reflect the actual service utilization in arbitrary periods of time, e.g., for every hour.

For the interaction points between this phase and the next it is crucial to define a common model exchange format. Several of the formats mentioned in section 2 (like MOF, etc.) suite this purpose. In [17] we developed an XML/RDF–based format specially suited for dependency description in distributed environments. However, for the purpose of this paper the actual representation is of minor interest.

Once the model is generated the last phase starts. Management tools use the generated models directly (step 4-i) or calculate and use comparisons of sequentially generated models (step 4-ii), e.g., to point out emerging differences (which indicate changes in the modeled environment) to the administrator or to generate alarms to management platforms if dependencies to certain important services change.

Special management cases may also require to derive an abstract service model from the environmental one. This helps to find unusual cases of unexpected dependencies either indicating inconsistencies in the model or errors in the network or service design.

3.3. Summary of Requirements

The following list of requirements summarizes the investigations of the overall modeling process and the analysis of the current approaches. Both clearly indicate the need for a new automated modeling approach that:

1. is able to sufficiently close the lack of dependency information;
2. handles the scalability issues required by large environments, regarding both model visualization and efforts to create the models;
3. supports domain concepts (actually as a consequence of the previous requirement);
4. allows to create models reflecting the actual system behavior in a certain time interval to enable management applications to evaluate short–term changes in the environment as described above;
5. cleanly decouples data collection (step 3-i) from the actual model creation (step 3-ii) to be able to reuse collected data for (overlapping) models created for different places, e.g., for different departments;
6. is based on data available for most types of managed resources, but still
7. needs very little effort to be installed and used in heterogeneous environments (this concerns both the heterogeneity of the probed objects as well as of the runtime–environments for the probes themselves);
8. has little impact on the performance of the managed systems and networks.

Further concerns like security issues are not covered by this paper; in our prototype security features like authentication of agents and encrypted communication are automatically provided by the agent system we use (the agent based architecture is presented in subsection 4.2).

4. New Approach to Automated Modeling

This section presents our new solution to the key part (step 3-ii) of the overall modeling process together with an agent based architecture allowing its clean integration into real IT–environments.

4.1. Determining Dependencies with Neural Networks

Our new approach is based on neural networks and thus very different from traditional ones explained further above. As input data for the neural networks we use any kind of values that express an object's activity within small time periods (of about 1 to 10 seconds each). We further restrict our selection on values that are relatively easy to collect and available for most types of services, hosts, etc. Examples for such values are, but are not restricted to:

- CPU activity of hosts (mainly useful, if the selected hosts are interesting objects by themselves or carry only one main service);
- CPU usage of an application, compared to the CPU power available over a certain period of time (useful in various cases measuring applications in scenarios different from above);
- communication bandwidth used by a system during each of the short time intervals;

• sum (or other appropriate function) of activities of subcomponents (if the activity of an object is not directly measurable as one value, like for distributed applications or domain objects).

Generally speaking, this is data taken from lower layers like the operating system, middleware or transport system.

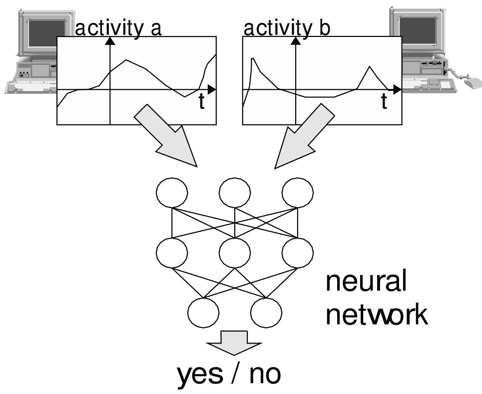

Figure 7. Neural network decides per pair of objects

As depicted by figure 7 two streams (time series) of activity data are fed into a pre-trained neural network for each relevant pair of objects. The neural network decides whether a dependency exists or not and (if required) retrieves further information about that dependency like the assumed dependency strength. Of course, the values of activity do not explicitly show a dependency. But simplifying the process within the neural network one can imagine that peaks of activity often occurring in both input streams with similarly repeating patterns over time (as depicted in the example below) allow the conclusion of a dependency.

Neural networks were chosen because of advantages, like:

• dealing with uncertain information, and
• robustness to noise in the input data.

These advantages are necessary to overcome the lack of explicitly useful information in the simple input values and problems like small timely displacements of values at certain managed objects (e.g., due to not well synchronized clocks). The second point is especially important, because—depending on the kind of values that express activity—there potentially is a lot of "internal" activity, meaning that actions are performed which are completely unrelated to other objects outside.

In our project we constructed and trained neural networks with data collected from real environments for which the results (whether dependencies between the objects exist or not) where known. For a proper decision quality the training set had to contain data from at least two or more distinct types of service implementations and different sources of activity data (to obtain samples of positive training cases) as well as pairs of non–related services (negative cases). Each of them was observed under various usage conditions and during times of high and low service utilization. Using data from real environments led to the problem of noisy training data, but with the neural networks quality to perform generalization on the input data our requirements could still be met. As a positive consequence the design and installation of a special test field was not necessary.

In later tests we positively verified the results in different environments without retraining the neural network. However, we do not exclude that it may be necessary to improve the neural networks in other cases, e.g., with the help of special reinforcement learning techniques that can be applied even in parallel to the networks utilization. Further studies on the robustness and general reusability are currently in progress.

Figure 8 shows two example plots of data collected from two hosts during the same period of time with a sampling rate of 5 seconds. The values shown represent the intensity of the hosts' IP–communications with others during time intervals of five seconds. It is of course just a very small excerpt of the real time series fed into the neural network.

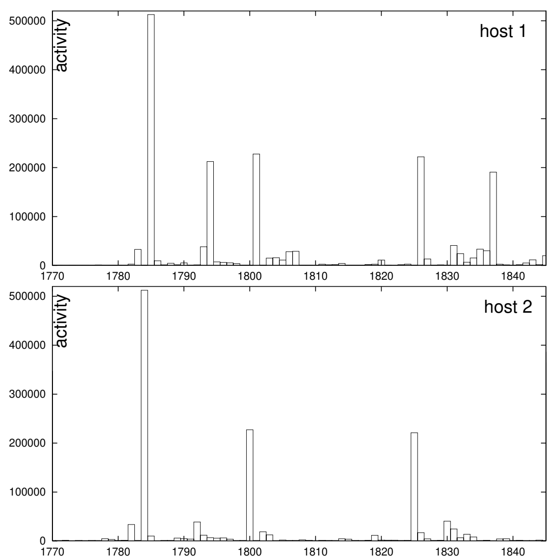

Figure 8. Time series of two hosts' activity values

The high spikes within the plots are of special interest. At three time intervals (numbered with 1785, 1801 and 1826 for the first host, respectively 1784, 1800 and 1825 for the second) both hosts show an activity of nearly the same intensity and shape indicating a possible relationship. The plot of host 1 additionally shows further significant activities (e.g., at 1794 and 1837) which turn out to be noise for the investigation of the two hosts' relationship.

The fact that two services show activity at the same time does of course not yet allow to say that they are dependent, but after observing this behavior several times with similar

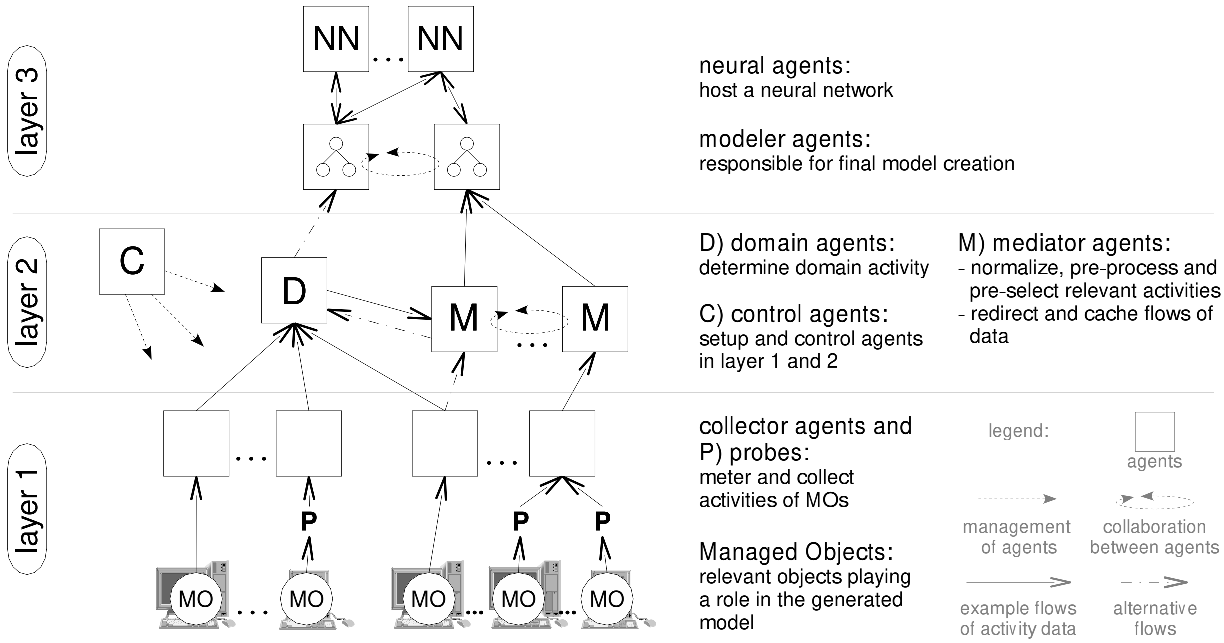

Figure 9. Architecture's hierarchy of probes and agents

peak patterns, a decision becomes plausible. Further algorithms are used to find groups of dependencies (occurring together or in a row, respectively) that belong to one type of transaction and to distinguish two timely unrelated dependencies involving common objects ($A \rightarrow B$; $B \rightarrow C$) from real transient ones ($A \rightarrow B \rightarrow C$).

A possible disadvantage of pairwise calculations is that it needs $O(n^2)$ time for n elements. For large numbers of n special techniques must be applied: One simple possibility is to pre-exclude pairs that are either not of interest, or where dependencies are not possible anyway. In the web server scenario one could omit all calculations for pairs of web clients. This usually makes up a significant percentage, comparing the huge number of clients against a smaller number of servers. Further reduction comes from applying the domain concept. Smaller models are generated per interesting domain. Additionally, the activity of the whole domain is condensed into one single domain activity allowing to calculate the dependencies between domains and also between one single object in one domain and (other) 'outside' domains.

At the same time the argument of complexity also supports the use of neural networks, as they—once trained—are able to calculate the results faster than traditional correlation analyses.

Looking back at the requirements listed at the beginning of this section, a decision method about dependencies as such is obviously not enough for comprehensive dependency modeling. It has to be supplemented with an architecture allowing for a clean integration into real IT–environments. Such an architecture is presented in the following.

4.2. Architecture for Automated Model Generation

The architecture explained in this section covers all steps of phase 3 of the modeling process, including data collection, the evaluation in neural networks and a well defined interface to the management tools of phase 4. It also contains means to simplify the installation process of the agents in the managed environment (in correspondence to phase 2), however the focus of this section lies on the operational aspects.

Besides some low–level probes the whole architecture is based on agents and agent systems, respectively, but its principles are not restricted to a special implementation of a management agent system. Reasons for the choice to use an agent based architecture are that the following features are provided by most agent systems in an easy to use way (see also [18]):

- interfaces to resources (managed objects),
- a communication infrastructure, and
- support for flexible balancing of duties.

However, it is not assumed that agent systems have to be hosted on all machines. As explained in the following, the architecture contains means to cleanly embrace other sources for activity measurement via proprietary or standardized management protocols. Along with the architecture, supplementary information about the prototypical implementation developed in the project is presented. The agent platform chosen is the Mobile Agent Systems Architecture (MASA, [19]) implemented and developed at our research group for general management purposes. The platform and agents are written in Java, making them to a great extent independent of the underlying system. The inter–agent communication is based on the Common Object Request Broker Architecture 2.0 (CORBA, [20]).

Our agent architecture is structured in three layers (as depicted by figure 9):

1. The lowest layer hosts all means of data collection. It contains implementations to measure data via standardized or proprietary management interfaces. It further provides a homogeneous interface for objects' activities to the next layer, thus making their heterogeneous nature fully transparent to the rest of the architecture.
2. The middle layer filters and pre-processes the activity data. It is further used to channel the data flows in a way that the impact on the performance of managed systems is kept at an acceptable level through load balancing and caching mechanisms.
3. The last layer hosts the components for model generation including the neural networks.

The basic types for the means of collection in layer 1 are:

1. management agents (in the sense used in management architectures like OSI or Internet management) already in place or to be installed,
2. proprietary measurement tools (delivered with applications, etc),
3. special probes, developed and deployed for the purpose of gathering information for this modeling,
4. or agents of the agent architecture directly capable of measuring through interfaces of the agent system.

As representatives of the first type our prototype supports access to SNMP agents, currently used to meter CPU activity of hosts and amount of network traffic on IP interfaces. Further we implemented probes of the third type, metering CPU utilization of applications by reading from the 'proc filesystem' (as provided by SUN Solaris, Linux and others). The means of collection should be installed close to the objects that have to be monitored to avoid unnecessary traffic. On the other hand, not all endsystems are capable of hosting an agent system or are not allowed to—for security or other reasons. In these cases remote monitoring (as in our case of data access via SNMP) is the preferred choice.

There is no difference, whether data is gathered (in step 3-i) to calculate a collective domain activity or for single objects directly represented in the generated model. Figure 9 shows the same flow of information for both cases. On the left hand side domain activity is calculated, while on the right hand side the information goes directly to mediator agents.

The homogeneous interfaces to the upper layers are provided by so called *collector agents*. If their agent system provides appropriate interfaces they are able to directly collect data from their host system. Otherwise they send queries to externally implemented probes.

The mentioned interface is divided into two parts. One is mainly used by the *configuration agents* to initialize and configure the agents while the second part is used for the data queries at run–time. The same interface is also used and provided by *mediator agents* in the second layer. This allows to cascade them in larger scenarios or leave them out in small ones. These agents may further implement automated load balancing by traveling to hosts with unused resources or to places with higher available communication bandwidth. There are two possibilities of how these agents may collaborate: Either they use the configuration part of another agent's interface to suggest the delegation of a task like to apply filters on data, or via the query interface, e.g., by rejecting queries to probes for which it had previously been responsible. In this case the agent may specify another agent that should be responsible from now on. For now, our prototypical implementation of the mediator agents concentrates on caching and simple delegation tasks. We do not yet make active use of mobility aspects and complex collaboration algorithms. Further tasks assigned to mediator agents are:

- pre–selection of time intervals containing significant patterns of activity;
- normalization of values;
- correction of timestamps from data of systems with long time lags.

In adaption to the available resources these tasks may either be combined in one agent for all data, respectively single data streams, or be distributed over multiple cascaded agents.

The *domain agents* basically behave just like mediator agents. However, they implement special processing function to combine various streams of input data to one single stream for the collective domain (or distributed application) activity. Their query interface also allows to acquire the underlying data of single objects to reduce the communication bandwidth in cases where detailed models are constructed within the domain, but the collective domain activity is needed for other models, too.

The last layer contains the process of model creation. It is distributed onto two kinds of agents. The *modeler agents* organize the modeling. They query the other agents for pre-processed data, initiate the modeling process and finally implement the dependency model interface used by management tools. In our prototype it also contains an applet based user interface allowing to supervise and control the modeling. Modeler agents may collaborate with others by sharing ready evaluated parts of the models. Thus, an enterprise–wide modeler agent might eventually only calculate the inter–domain dependencies and query the underlying structures from local modeler agents. The second type

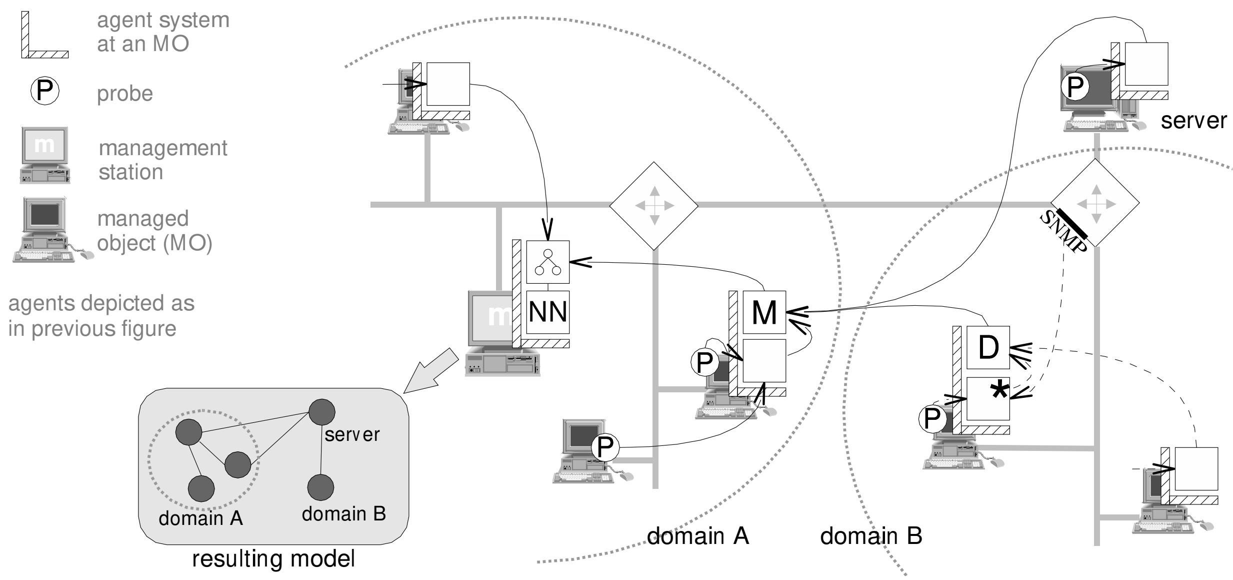

Figure 10. Deployment of probes and agents

on this layer are the *neural agents* which implement the neural networks. It is possible to install a pool of these agents and use them from the modelers as required. However, to reduce overhead it is recommended to place neural agents close to the modelers or even on the same agent system.

Figure 10 shows the agents deployment in an IT-environment with two domains. The goal is to construct a dependency model (depicted by the gray box) for the administrator of domain A, who is interested in the details of his own domain as well as the connections to the central server and the second domain B.

The data flows begin at the probes or the collector agents, respectively. As an example, one agent in domain B (like the others depicted by a white square without inner symbol, but marked with an asterisk '*') uses queries to an SNMP management agent on the router and additionally collects data from an external probe on its own host. The agent on the other system in the domain directly accesses its host via management interfaces provided by the agent system. Both agents' data is then forwarded to the domain agent that calculates the resulting domain activity by joining the time intervals and summing up the values in case of overlaps. On the interface towards the mediator it behaves just like any collector agents. Therefore, the whole domain appears as just one object in the model. The mediator agent carries out pre-processing on the data that did not already take place and forwards it to the modeler agent, which generates the complete resulting models with the help of a neural agent.

5. Conclusions and future work

Our work showed that environmental models provide numerous advantages for various management tasks. Over the past years several powerful applications and algorithms based on models have been developed and successfully tested. However, there are reasons why they are not widely used. The most significant is the lack of generally available models and the huge effort needed to generate such models by hand. In this paper, we presented a methodology that helps to overcome this lack by enabling the creation of such models for various use cases, in a—to a considerable extent—automated way, thus solving the worst problems. Further, the paper provided an agent based architecture enabling the model creation to be used in large scale scenarios.

For future work of the project, we consider to have a closer look at further scalability issues, e.g., to determine the number of objects our approach is able to handle in a single domain, esp. taking into account that bandwidth and other resources should be used for management only in a very careful and restricted way. For extreme scenarios the project will investigate how far the use of resources can be reduced, while still being able to generate models of satisfactory quality. Or in other words, can the neural networks be trained better so they are able to cope with much less grained data?

For the part of the neural networks we consider to work on improvements allowing to distinguish between different types of dependencies. A second point is that—additional to the way it is implemented now, where the IT–administrator is not at all involved in the training process of neural networks—a feedback mechanism from the GUI to the neural agents could help to improve the neural networks and thus the modeling results. However, the pre-trained neural network currently used in our prototype already reliably works for various use cases.

Acknowledgment

The author wishes to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of the paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Its webserver is located at `http://wwwmnmteam.informatik.uni-muenchen.de`.

References

[1] H.-G. Hegering, S. Abeck, and B. Neumair, *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*, Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999, 651 p.

[2] R. Gopal, "Layered Model for Supporting Fault Isolation and Recovery," In Hong and Weihmayer [21], pp. 729–742.

[3] "Information Technology – Open Systems Interconnection – Structure of Management Information," IS 10165-X, International Organization for Standardization and International Electrotechnical Committee.

[4] "Information Technology – Open Systems Interconnection – Structure of Management Information – Part 7: General Relationship Model," IS 10165-7, International Organization for Standardization and International Electrotechnical Committee, 1997.

[5] Manuel Rodriguez, "Modeling Object Relationships in TMN / OSI Management Systems with SDL-92 (one page poster)," In Hong and Weihmayer [21].

[6] K. McCloghrie and M. T. Rose, "RFC 1213: Management information base for network management of TCP/IP-based internets:MIB-II," RFC, IETF, Mar. 1991.

[7] B. Gruschke, S. Heilbronner, and N. Wienold, "Managing Groups in Dynamic Networks," In Sloman et al. [22].

[8] B. Gruschke, "Integrated Event Management: Event Correlation using Dependency Graphs," in *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, DE, USA, Oct. 1998.

[9] M. Hasan, B. Sugla, and Viswanathan R., "A Conceptual Framework for Network Management Event Correlation and Filtering Systems," In Sloman et al. [22], pp. 233–246.

[10] S. Kätker and M. Paterok, "Fault Isolation and Event Correlation for Integrated Fault Management," in *Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management (IM 97)*, A. Lazar, R. Saracco, and R. Stadler, Eds., San Diego, USA, May 1997, pp. 583–596, Chapman & Hall.

[11] T. Kaiser, *Methodik zur Bestimmung der Verfügbarkeit von verteilten anwendungsorientierten Diensten*, Ph.D. thesis, Technische Universität München, 1999.

[12] A. Pell, K. Eshghi, J. Moreau, and S. Towers, "Managing in a distributed world," in *Proceedings of 4th International Symposium on Integrated Network Management*, Yves Raynaud and Adarshpal Sethi, Eds. IFIP, May 1995, Chapman & Hall.

[13] A. Clemm, *Modellierung und Handhabung von Beziehungen zwischen Managementobjekten im OSI-Netzmanagement*, Dissertation, Ludwig-Maximilians-Universität München, June 1994.

[14] "Common Information Model (CIM) Version 2.2," Specification, Distributed Management Task Force, June 1999.

[15] CabletronSystems, "Spectrum enterprise manager 5.0 rev 1," `http://www.spectrummgmt.com/support/manuals/501admin.html`, 1999.

[16] G. Kar, A. Keller, and S.B. Calo, "Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis," In Hong and Weihmayer [21], pp. 61–75.

[17] C. Ensel and A. Keller, "Managing Application Service Dependencies with XML and the Resource Description Framework," in *Proceedings of the 7th International IFIP/IEEE Symposium on Integrated Management (IM 2001)*, G. Pavlou, N. Anerousis, and A. Liotta, Eds., Seattle, Washington, USA, May 2001, IEEE Publishing.

[18] R. Pinheiro, A. Poylisher, and H. Caldwell, "Mobile Agents for Aggregation of Network Management Data," in *First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents (ASA/MA 99)*, Palm Springs, California, October, 3–6 1999, pp. 130–140, IEEE.

[19] H. Reiser, S. Heilbronner, and B. Gruschke, "Mobile Agent System Architecture — Eine Plattform für flexibles IT–Management," Tech. Rep. 9902, Ludwig-Maximilians-Universität München, Institut für Informatik, München, 1999.

[20] "The Common Object Request Broker: Architecture and Specification," OMG Specification Revision 2.0, Object Management Group, July 1995.

[21] J. W. Hong and R. Weihmayer, Eds., *NOMS 2000 IEEE/IFIP Network Operations and Managment Symposium — The Networked Planet: Management Beyond 2000*, Honolulu, Hawaii, USA, Apr. 2000. IEEE.

[22] M. Sloman, S. Mazumdar, and E. Lupo, Eds., *Integrated Network Management VI (IM'99)*, Boston, MA, May 1999. IEEE Publishing.