New Approach for Automated Generation of Service Dependency Models

Christian Ensel[1]
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

Managing services by not only looking at their own current state but with support from the "big picture" of service– and inter–service dependencies becomes increasingly important in nowadays' IT–management. This paper presents a new methodology to automatically generate such dependencies models together with an agent based implementation architecture. It strives to enable more comprehensive IT–management by providing an always up–to-date information basis about inter–dependencies of services, applications and network components. The approach specially aims for heterogeneous environments as found in large enterprises and outsourcing scenarios.

Keywords

automated dependency model creation, IT–management, agent based architecture

1 Introduction

Over the last couple of years IT–management has made enormous progress: Management architectures and their realization in platforms and tools provide standardized remote access to the managed environment [1]. But there is still a number of management tools that is based on proprietary resource interfaces. And worse, often important management information is 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 by the various management applications described in section 2.

Figure 1 shows a small scenario with two domains. One hosts a web server `WS`, the other contains the Domain Name Service `DNS` responsible to resolve the name of the web server and other subservices that are not shown in the figure, e.g., a database server providing content information for the web pages. Thus, the web server is said to depend on the DNS server. Further, there are two users who typically access information on `WS` via web clients. They depend on the web server and—if they want to type normal URLs instead of IP addresses—also on `DNS`. The figure also depicts the mentioned dependencies between the major objects. For simplicity, it neglects all dependencies to the communication infrastructure and further sub–services.

Although several objects of the example depend on DNS, none of them explicitly tells to do so and cannot be queried by a management application for their dependencies. In the example,

[1]The author's work was sponsored by Siemens AG, department CT IC 4 in the context of the project "LEONET" of the German Ministry for Education and Research (BMBF)

the dependencies are hidden in `WS`' 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 "resolve.conf". 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 "resolve.conf", because it would then be up to a third file to determine whether local resolving is carried out at all.

Conventional approaches generally struggle with evaluation problems of various configuration files as mentioned above. It gets even worse if the format of those files changes with software updates. In heterogeneous environments such modeling tools typically must further be restricted to a very limited set of resource types or vendors. The major alternative to dependency detection at runtime is 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– resp. Module Interconnection Languages. However, so far similar dependency description languages have not been commonly agreed on in the IT–network and service provisioning world.

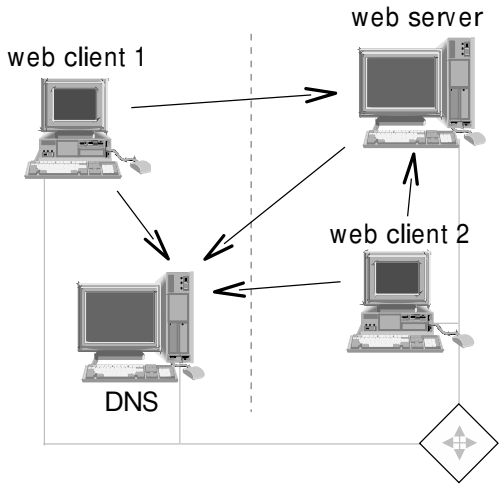

Figure 1: Simple multi service scenario

As a consequence, dependency models are not generally used in today's management world—although their benefits are commonly known. This 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.

To overcome the problems of automated modeling described above, 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 3) 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 the training methods of our neural networks etc., but concentrates on modeling and realization aspects relevant for IT–management. Thus, the second part (section 4) deals with the concepts' integration into real IT–environments and management processes, where questions of installation efforts, scalability etc. have to be taken into account. The conclusions of the paper are drawn in section 5.

2 Existing Modeling Concepts and their Applications

To provide means for the description of management information has always been an important part of the definition of management architectures. Syntax and semantics of information descriptions are defined in the so called *information model*; the access to it in the *communication model*. The focus of management information traditionally lay on attributes and properties of single objects. Although, e.g. the ISO/OSI (Open Systems Interconnection, [3]) management architecture 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 type of management information which has gained more and more importance with the growing complexity of IT-systems and networks: the dependencies between managed objects, captured by the so called *Dependency Models* In the following, 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.

2.1 Environmental Models

Figure 2 shows a directed graph expressing the dependencies of the objects in the scenario as described in the introduction. Its main components are the web server, both web clients, the DNS server and a generalizing object for the common communication infrastructure. The edges in the graph represent the dependencies. 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. Information attached to them reflects the content of traditional object related management variables. The (directed) edges represent 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.:

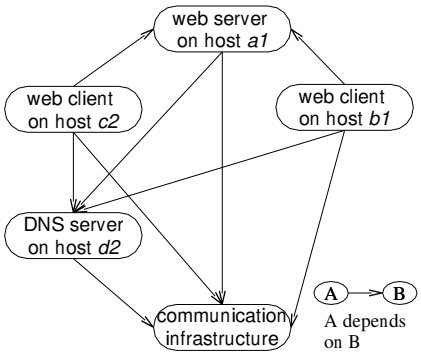

Figure 2: Simple instantiated Service Dependency Graph

- 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, or
- to attach values of strength or likelihood.

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

Several research projects have come up with utilizations of such models. One of the applications 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. [7], [8] and [9] 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 [10].

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, resp. to simulate the effects on other systems beforehand. Further investigations of advantages can be found in [11] and [12]. 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 3—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

The main elements of abstract models are classes providing an abstraction of the specialties of real environments. This contrasts the components of environmental models, which deal with objects directly mappable to the real world. The dependencies between the classes are specified on the same level of abstraction. It is easy to see that environmental models actually are instantiations of abstract models—of their objects as well as their dependencies.

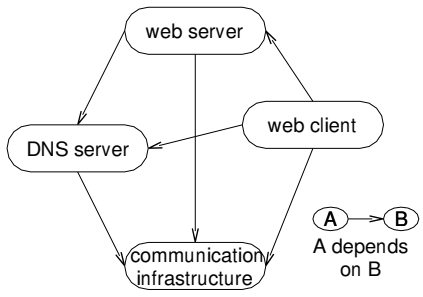

Figure 3: Simple Abstract Service Dependency Graph

Abstract models are normally generated by hand—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 3 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, resp. needed attributes.

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 more simple to handle.

However, the models' restriction to the abstract level has of course 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 an environmental model. 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, [11]) systems, since 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 have so far not been very successful with regard to their share on the market: 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 strengthen 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, [13]), 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 Oriented Approaches

To obtain a complete picture, one also has look at object oriented approaches. Standardized object oriented modeling of management information exists at least since the definition of the ISO/OSI management architecture [3]. An example for a vendor specific object oriented management model is the one used in Cabletron Spectrum [14]. It was introduced to overcome the lack of a similarly powerful information model in the Internet Management.

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 [13]. 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, resp. 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, resp. model generation CIM also needs to be supplemented with further algorithms.

2.4 Extensional Domain Concept

In complex scenarios it is hard to keep the overview even with the help of environmental models. To reduce the number of elements, resp. to restrict the model to currently interesting parts, the concept of domains is introduced as addition to the modeling concepts presented above to provide a simple means to structure models hierarchically. 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 diagnosis. Graphical user interfaces of management tools will typically be capable of unfolding domains and of generating and presenting a model including underlying objects, resp. to navigate into submodels.

Domains are represented by only one element in the model, but stand for (and may be expanded to) collections of:

- objects in the real world (selected by IT-managers) that are relevant for modeling, and/or
- other (sub-)domains.

The selection of domains depends on the management purpose. Possible criteria are:

- administrative zones,
- organizational (work-) groups,
- topological aspects regarding the underlying network or
- buildings, etc.

As domains obviously are an important concept, the architecture presented in section 4 is able to generate models that include domain elements and submodels. [15] presents further details on the subject of domains.

3 New Methodology for Automated Model Generation

It is a challenging task to automatically generate environmental models using information gathered at run-time, without disturbing the normal operation of networks and systems. A straightforward method to determine the dependencies is to choose data directly expressing this kind of information. Examples are entries of utilization in server log files. In the example of figure 2 the web server's log file would contain entries showing that both clients had connections. In other words, there is a dependency to each of them.

However, a major drawback of this approach is, that log files typically have a proprietary format or sometimes change from version to version of the application. Even worse, not all applications provide log files or similar mechanisms containing this information, or its access may be restricted for several other reasons, like security policies or limited amount of local disk space. Similar problems of analysis of configuration files at client side as well as the current lack of a standardized and commonly accepted service description format (as discussed in section 1 show the necessity of a new approach.

Determining Dependencies with Neural Networks

The suggested alternative solution chosen by this project is to concentrate on information relatively easy to collect and available for most types of services, hosts, etc. and to decide for each relevant pair of objects, whether a dependency exists and (if required) the type and attributes of that dependency. This is done by a Neural Network fed with time series of the objects activities to judge whether they "have something to do with each other" or not. Of course, values of activity do not show the dependencies explicitly. The fact that two services show activity at the same time does not yet allow to say that they are dependent, but after observing this behavior several times (within a certain period of time), such a conclusion is plausible.

Examples for values of activities measured per object are:

- CPU activity of a host (mainly useful, if the selected host is an interesting object by itself, or hosting 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, and
- sum (or other appropriate function) of activities of sub-components (if the activity of an object is not directly measurable, e.g., of a distributed application)

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

For the project, we constructed and trained neural networks. After normalization and preselection of relevant intervals in the activity data they are capable of deciding for a pair of objects whether there is a relationship or not.

Neural networks were chosen because of advantages, like:

- dealing with uncertain information,
- robustness to noise in the input data

and others, described in more detail in [16]. 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. The complex training process of the neural networks needed to achieve the necessary robustness and flexibility cannot be presented in the brevity of this paper. The interested reader will find more details about it in [17].

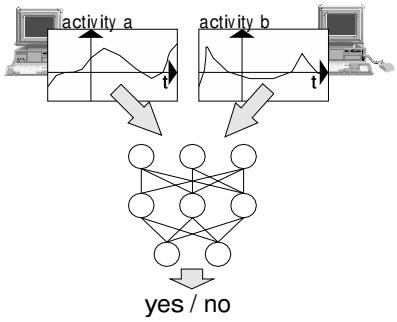

Figure 4: Neural Network decides per Pair of Objects

A possible disadvantage of pairwise decision over dependencies between all objects 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 what 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 introduced in section 2.4. Smaller models are generated per interesting domain. Additionally, the activity of the whole domain is condensed into one single "domain activity" (e.g., by summing up activity values of important objects) allowing to calculate the dependencies between domains and also between one single object in one domain and (other) 'outside' domains.

4 Architecture for Automated Model Generation

The purpose of the architecture is to enable an implementation that generates the previously described models of environmental dependencies—not just for a pair of objects, but on a larger scale. To support the understanding of the architecture's design (section 4.2) and the purpose of its components, the overall modeling process is explained step by step in the subsection below. It starts from selecting the models objects and covers the process up to the automated model generation and utilization in management tools.

4.1 Overall Generation Process

The following explanation of the process assumes that the implementation is already installed in the managed environment. Figure 5 depicts all steps to be carried out by the IT-managers and the involved algorithms during the most important parts of the model's 'lifecycle'.

In the first step the administrator has to select all services, components, departments etc. that should be part of the model. These will later appear as the objects in the model. Of course, this selection highly depends on the management tasks the model will be used for. There are two variants, on which level the selection process can take place: Either (i') 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. Or (i'') directly on the level of objects in the real world, e.g., *web server on host* `a`, *web client on host* `b`, etc. Note, that domains are represented as objects, too.

In step ii the matching components, applications etc. in the real world must be chosen for each object. This requires no special actions for simple objects, but only 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*.

Step iii, where the appropriate probes to meter the objects' activity (as explained in section 4.2) must be installed, is the last step of the installation phase. 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 same channels as the user data. Therefore, the architecture presented in the following section supports different

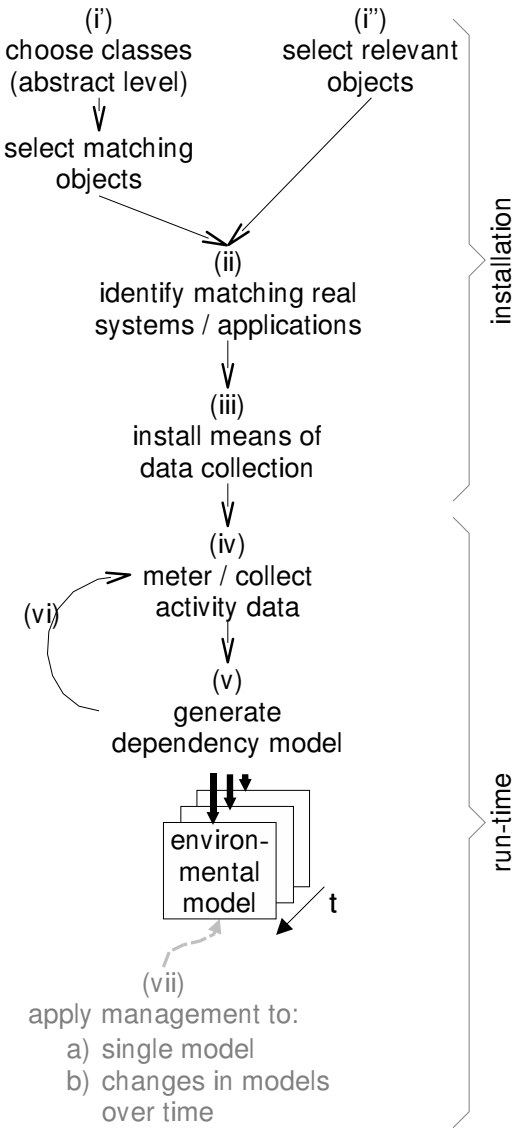

Figure 5: Steps of the Modeling Process

kinds of collectors, assuring its suitability for various environments.

During normal operation of the systems and networks, activity data is collected (step iv). After a certain period of time the information is transferred to places where a model can be generated (step v) using the method described in section 3. From hereon management tools can start to use the generated model (vii, a)). Special management functions (vii, b))—as described at the end of section 2.1—may be applied on sequentially generated models (step vi). For example, pointing out the differences to the administrator or generating alarms if heavy (or certain) changes occur.

4.2 Agents Architecture

This section describes the actual architecture based on a management agent system. Reasons for that choice are the following features, that are provided in an easy to use way (see also [18]):

- interfaces to resources (managed objects),
- 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 independent of the underlying system to a great extent. The inter–agent communication is based on the Common Object Request Broker Architecture 2.0 (CORBA, [20]).

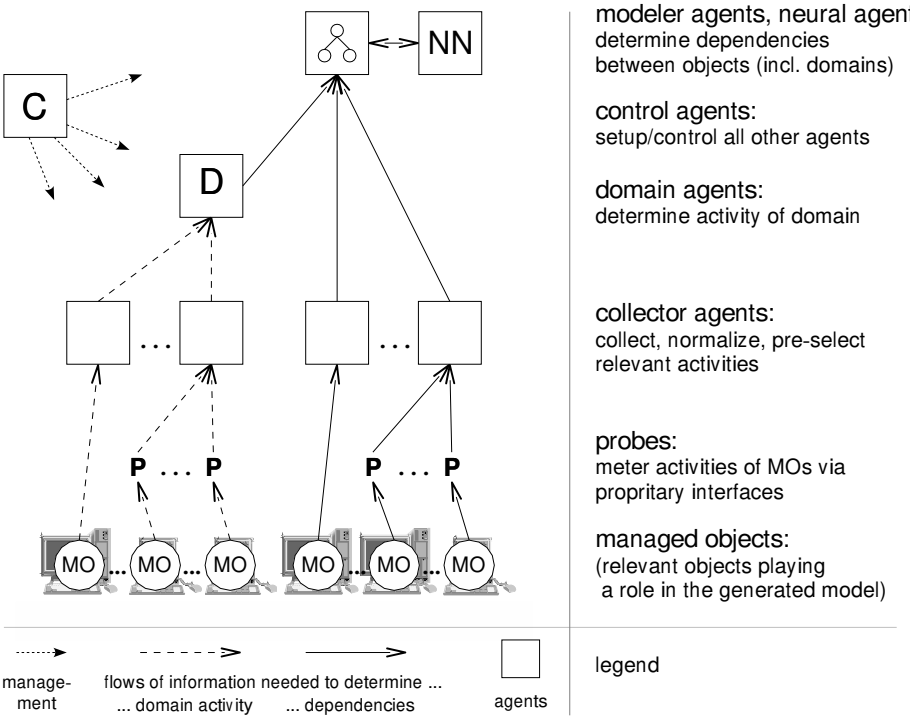

Figure 6: Architecture's hierarchy of probes and agents

Regarding the steps of the overall process, the architecture first becomes involved in step iii. The input at that step is a list of objects that play a role in the desired model and lists (for each object) of corresponding real managed objects and measurable activity values. With this information, one or more *control agents* (see figure 6) start the required data collection and domain agents in the managed environment.

Means of collection 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, the implementation supports access to SNMP agents, currently used to meter CPU activity of hosts and amount of network traffic on IP interfaces. Further implemented are probes of the third type, metering CPU utilization of applications by reading from the 'proc filesystem' (as provided by SUN's 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 is the preferred choice.

There is no difference, whether (in step 4) the information is gathered to calculate a collective domain activity or directly for the later model generation. Figure 6 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 the *modeler agent*.

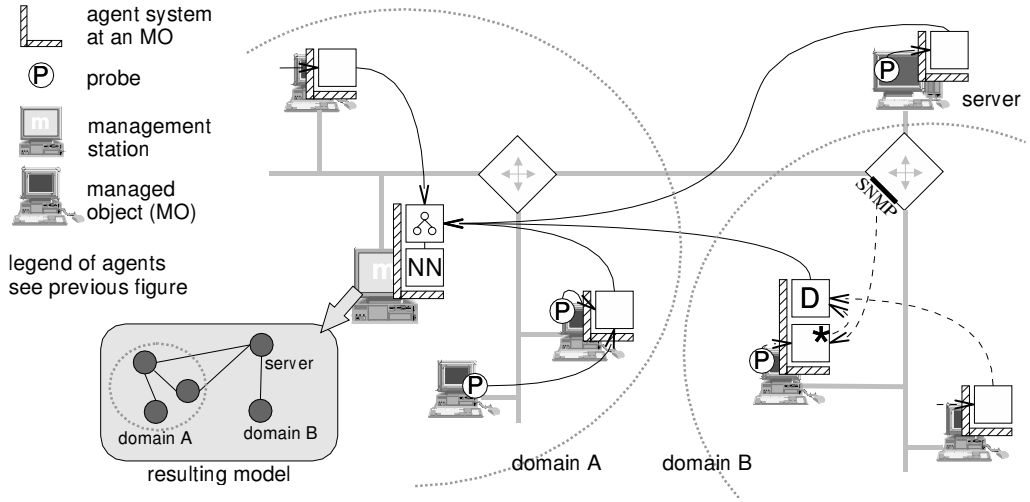

Figure 7: Deployment of probes and agents

The *collector agents* help to concentrate the flows of data. In figure 7 the collector agent in domain B (like all depicted by white squares without inner symbol, but marked with '*') uses queries to an SNMP management agent on the router and collects data from a probe on its own host. The agent on the other system in the domain directly accesses its host. Further tasks assigned to collector agents are the pre–selection of time intervals containing significant patterns of activity and the normalization of values. These tasks are combined in one agent

to reduce the required communication bandwidth. In the example, 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 modeler the agent behaves just like collector agents. Therefore, the whole domain appears as just one object in the model.

Modeler agents are responsible for the generation of the complete resulting models. The task to decide for each pair whether a dependency exists (in the way it is explained in section 3) is performed by one or more *neural agents* containing the pre-trained neural network. An example model for the scenario is included in figure 7. All systems except the one hosting the modeler agent and one router are part of the model, however, the systems in domain B are only represented indirectly by one single element. In cases where both modeler and domain agent are present in the same domain (to generate a detailed model and calculate the overall activity for other models), the data collection still must take place only once. The modeler simply queries all information from the domain agent which does not only aggregate the data, but also caches the individual activity time series.

5 Conclusions and future work

This paper presented a methodology that enables the creation of powerful dependency models for various use cases, in a—to a considerable extent—automated way. Thus, it solves the worst problems of existing applications of dependency models by overcoming the lack of up-to-date models. Due to the nature of the new approach it is even able to do so in heterogeneous environments. This paper further described an agent based architecture enabling the modeling of large scale scenarios as they are typically found in nowadays' IT-management-world. This is an important advantage, as those scenarios disallow manual model creation simply due to the huge number of managed objects involved.

For future work of the project, we consider to have a closer look at 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 case.

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, "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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

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

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

[16] Denise W. Gürer, Irfan Khan, and Richard Ogier, "An Artifical Intelligence Approch to Network Fault Management," California, USA.

[17] C. Ensel, "Bericht zum Arbeitspaket KM 2.1 der Siemens Kooperation (LEONET)," Kooperationsbericht, July 1999.

[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] B. Gruschke, S. Heilbronner, and H. Reiser, "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.