Proceedings of the 1st IEEE Latin American Network Operations and Management Symposium (LANOMS'99). December 3-5, 1999, Rio de Janeiro, RJ, Brazil. ISBN: 85-900382-3-8.

Algorithms and Contracts for Network and Systems Management

Paulo Pereira
Instituto Superior Técnico,
Universidade Técnica de Lisboa.
INESC, R. Alves Redol, 9.
1000 Lisboa, Portugal.
Voice: +351-213100345. Fax: +351-213145843.
prbp@inesc.pt

Paulo Pinto
Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa.
UNINOVA, Quinta da Torre.
2825 Monte de Caparica, Portugal.
Voice: +351-212948545. Fax: +351-212948532.
pfp@uninova.pt

Abstract

Most current management applications provide a rather low-level view of the managed systems, have limited operations automation, usually forcing the manager to work at MIB variable level. To solve these problems, an approach based on a hierarchy of management algorithms for the different abstraction layers is used in this paper to manage a company's business processes. An example is described to illustrate how the top-level management requirements are refined down to the equipment level, formalized in contracts enforced by the algorithms and monitored by management QoS parameters offered by the algorithms.

1. Introduction

The main task of a network manager is offering the end users the best possible service from the network with the minimum cost. As networks grow in number, dimension, complexity, dynamism and equipment variety, network management and control is becoming a non-trivial task, and the amount of effort dedicated to manage a network and its systems is substantial.

The network manager's task can be simplified by automating simpler tasks, or by providing him operations powerful enough to implement management goals, with decisions taken at a very high level.

This paper describes an approach based on management algorithms and contracts to attain these objectives. It starts with an overview of the different management abstraction levels, and how the management requirements can be refined from the business process goals down to the equipment level.

The management framework is based on a hierarchy of algorithms mapped onto the different abstraction levels. Each algorithm automates the management operations related to a certain management aspect, as specified on a contract made with its user. Contracts are general-purpose configuration statements specifying service guarantees to enforce management goals. Management quality of service (QoS) parameters allow monitoring the guarantees' fulfillment, providing a measure of how well the algorithms are doing their job.

The internal operation of the algorithms is then described, with an example to illustrate the generic algorithm operation models and the improvements obtained by using the algorithms. The following sections list the benefits of this approach, compare it with other related work, draw some conclusions and list other issues for further research.

2. Management Abstraction Levels

The concept of *business processes* is used in modern enterprises to aggregate the tasks needed to accomplish a certain objective (accounting, sales, product manufacturing, etc.). As shown in figure 1, these business processes are supported on *services*, which nowadays are mostly network and computer based. These services are implemented as distributed applications over a distributed services layer that provides location independence, transparent remote invocation, and event notification. Each service is implemented through a few *processes* running on the company's computers. The processes depend on *equipment* like computers, peripherals, and network equipment. Each of the layers identified (business process, service, process and equipment) corresponds to a different management abstraction level. However, this is not a strict division, as the layers can be further subdivided into sub-layers to ease the implementation.

The business process requirements – usually a prose text – are successively refined to requirements that make sense at each level (availability, performance, security, configuration and accounting specifications for each service, process, and equipment). In addition to the refinement, some requirements specific to the implementation of each level have to be added by the network manager to configure that level. For instance, the number of servers needed and the network capacity have to be specified, even if not present in the topmost requirements. All requirements are formalized as management contracts to the management algorithms that enforce them.

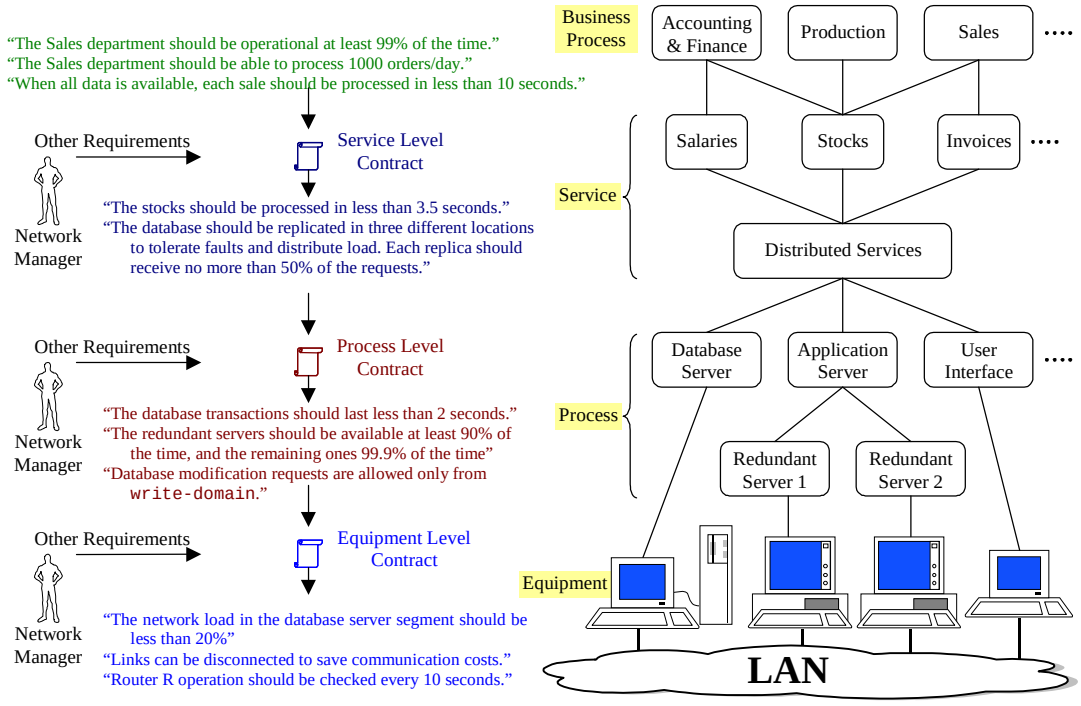

Figure 1: Management abstraction levels.

The refinement process is done by analyzing each business process dependencies and mapping the top-level requirements onto them.

Picking the case of the sales delay requirement, figure 2 shows how a sales process is decomposed. These different components are used to establish the contracts for the management algorithms at each abstraction level. The corresponding process level contract is:

"The database transactions should be performed in less than 2 seconds."
"The applications should respond in less than 2 seconds."
"The data checking delays should be less than 1 second."

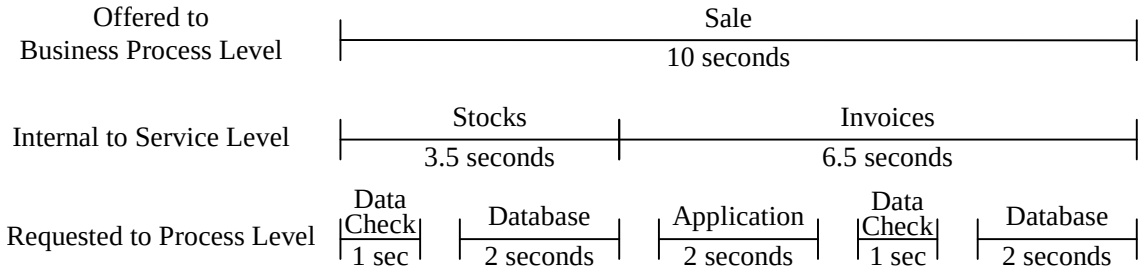

Figure 2: Refinement of the sales delay requirement.

[Alexandrov 97] proposes a similar decomposition model for performance evaluation applied to WWW services.

At the upper abstraction levels, the refinement process clearly needs human intuition to define the configuration requirements and the specific algorithm behaviour. It is envisaged that below a certain abstraction point, the refinement could be done automatically, by selecting the correct behaviour from the available algorithms at that level, establishing contracts and selecting the usual configuration requirements. In fact, middle and lower level algorithms have a parameterized generic behaviour acting on real managed objects such as servers, routers, trunks, system parameters and management instrumentation in general. In this way, algorithms act autonomously within certain border values as specified in their contracts.

A hierarchy of algorithms is then deployed, to decompose the management problem into several sub-problems at different abstraction levels and at different network locations. To keep their contracts, the algorithms use their own effort, and the lower level algorithms as specified in contracts made with each one. These contracts can be dynamically changed by the upper algorithms as the system runs, to adjust to changes in network or system conditions, and better fulfill the top-level contract.

As we go down the hierarchy, the network will be more segmented. The level immediately above a segmentation can take the segmentation in consideration as another degree of action to fulfill its contract, for instance by balancing load between the different redundant servers, as shown in figure 1.

The different abstraction levels help both the specification and the implementation of management. The number and definition of abstraction levels varies in the literature. The logical layered architecture of TMN [M.3010] proposes four levels: *business*, concerned with a total enterprise (i.e. all services and networks) and carries out an overall business coordination; *service*, concerned with services offered by one or more networks; *network*, concerned with each network; and *element*, managing individual network elements. [Moffet 93] proposes three levels: *goals*, specifying objectives; *policies*, defining rules; and *plans*, corresponding to procedures of actions. The same notation [Marriot 96] is used for all the abstraction levels. [Koch 96] also proposes three levels: *requirements*, described in prose; *goal-oriented*, using templates of attributes specifying events, constraints and actions; and an *operational* level, where a Policy Description Language and an Event Definition Language are used. [Wies 95] proposes four levels: *corporate policies*, specifying corporate goals; *task-oriented policies*, at task or process level; *functional policies* at SMF level; and *low-level policies* at managed object level.

3. Management Algorithms

Management *algorithms* are active objects with internal state and a set of operations, specialized on asserting certain conditions over a subpart of the network, by automatically solving the related problems. The conditions it takes care of are formalized in a *contract* to the entity that launched the algorithm. Contracts can be monitored by *management quality of service* (QoS) parameters. Figure 3 shows the internal structure of an algorithm for the particular case of the service level algorithm.

An algorithm offering a contract makes use of other lower level contracts. At each level, several algorithms can be running to simplify implementation, but the structure is similar for each one (contracts, configuration requirements and QoS parameters). Each algorithm monitors lower level QoS parameters, receives events and invokes actions over the objects under its control. Internally, the algorithms have a monitoring and reactive part to provide quick answers to problems, and a proactive part acting based on the events history to help in planning.

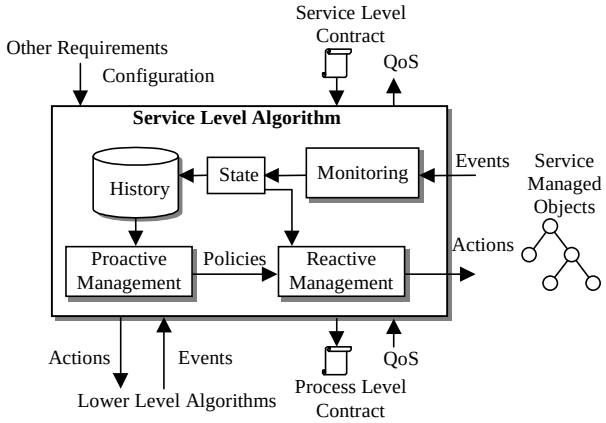

Figure 3: Service level algorithm.

The management application is thus the hierarchy of the management algorithms, built according to the refinement process. As long as the contract is held, the manager can relax on those aspects, and work at a higher level of abstraction. Contracts are built up of guarantees offered by the algorithms. Each guarantee has an associated semantics that can be:

best-effort: the algorithm does its best to keep the guarantee.

statistic: the algorithm allows a certain percentage of violations.

deterministic: the algorithm guarantees the condition at all times.

Contracts do not have a simple success-failure result. While working, the algorithms make the current status of its activity available to its user by the QoS parameters. This allows for a monitoring activity from the layer above, in order to act at a higher level, to prevent future failures, if something is not running according to the plan.

The management QoS parameters are important to police how the algorithm is performing its task. This is even more important on best-effort guarantees that do not lead to contract failures, but only provide a good measure of the distance towards the ideal conditions. These management QoS parameters can be seen as a generalization to network management of the common QoS [Aurrecoechea 98] parameters. Examples of QoS parameters are service delays, service availability, costs, server load, network load, and error rates.

Constructing the behaviour of higher level algorithms is very much like programming distributed applications. Each algorithm is active, has its own autonomous behaviour, and interacts with the other system components to fulfill its contract. As we go down the hierarchy, actions become more restrict and can fit different models controlled by parameters. Three simple examples are the following:

- A decision orthogonal multi-variable space can be built in which each coordinate has a certain weight contributing to a QoS parameter, as exemplified in figure 4. Variables can be classified as discrete (on/off, high/low/stop), or continuous (when any real value is allowed). Variables represent some entities related with the guarantees (system configuration, number of users, etc.), that can be dynamically modified within a certain range. A typical situation is specifying a QoS optimum working point and a maximum deviation. Typical algorithm reactions are operation invocations on lower level algorithms to improve QoS (e.g. increase something 10%). Usually, variables are modified one at a time.

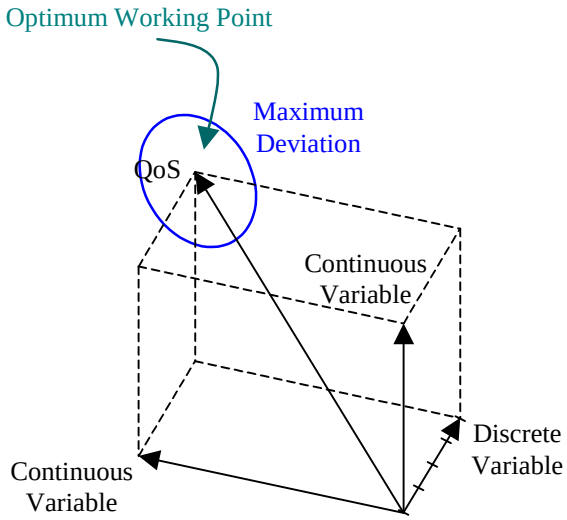

Figure 4: Internal algorithm operation.

- The previous model assumes the variables are independent. If dependencies are known, simultaneous modification on more than one variable can lead to faster convergence. This corresponds to oblique plan movements in relation to the axes.
- The previous examples assume the variable contributions are monotonic. If the dependencies between variables are too complex, and a specific formula can be identified (polynomial, average, differential, fuzzy, etc.), a parameterized specific model can be used. The monitoring of the system will provide the information for the algorithm to react.

4. Sales Example

In this section, the example of the sales delay is further analyzed. A simulation of the system was built using the Ptolemy simulator [Ptolemy]. Figures 5 and 6 show the graphs of the average and maximum sales delay versus the sales request rate, with and without the algorithms operating.

It can be concluded that when the rate of user requests increases too much, an overload situation on the servers and on the network happens, causing service degradation.

According to the contract, sales processing should always be below 10 seconds. This is an example of a deterministic guarantee, as it should never be exceeded. However, a less rigid contract could also be written with a statistic guarantee, allowing for some violations.

To keep the contract, the algorithms optimize system and network parameters, improving operation, and reducing service delays. In addition, the algorithms prevent entering the service degradation zone, by stopping new users from entering the system whenever necessary.

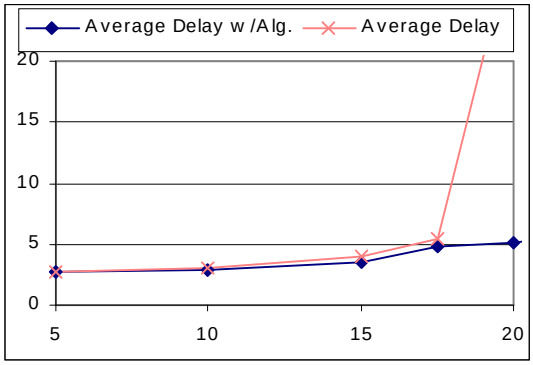

Figure 5: Average sales delay vs request rate.

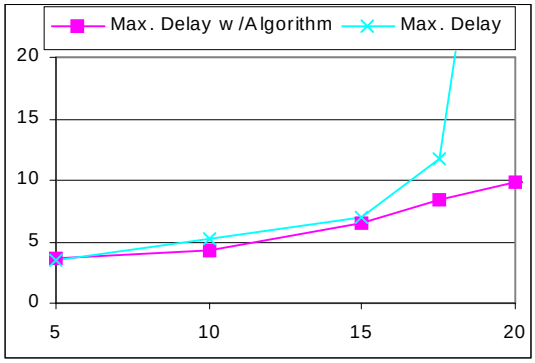

Figure 6: Maximum sales delay vs request rate.

The lower level contracts derived from this top-level contract have a statistic semantics, since infrequent violations are tolerated. When the higher layer algorithms sense (by monitoring the QoS parameters) that lower level contracts are near the edge, they adapt the conditions in the network, since they control other parameters, so as to improve the situation and prevent contract failure on both layers. An example is diverting database requests to other database replicas in different locations.

The meaning of the QoS parameters depends on the abstraction level. At the top level, they allow for the assessment of the overall quality of the service being provided to the company's clients, providing a measure of service time, service throughput and availability (although the last ones correspond to different contracts).

At the process abstraction level, QoS parameters provide a measure of server load, server throughput, response times, availability and number of users.

At the equipment abstraction level, QoS parameters provide a measure of error rates, network traffic in each segment, collisions, cost of the external connections, equipment load and availability.

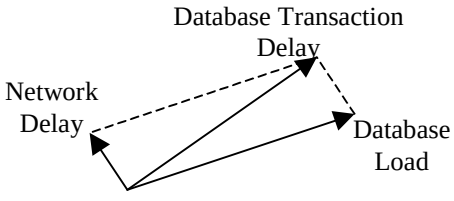

Figure 7: Contributions to the database delay.

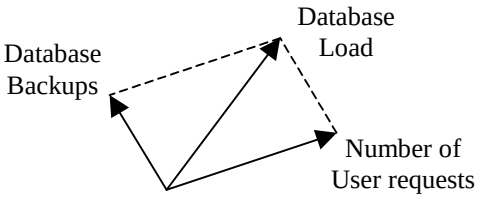

Figure 8: Contributions to the database load.

At the **service abstraction level**, the algorithm analyzes the different service times, trying to compensate process variations by restricting or loosening the conditions imposed on the process

level contracts. For instance, if the application delays consistently increase, and the database delay can be tightened, the algorithm does the corresponding process level contract adjustments, still respecting the overall 10 seconds requirement. In addition, if the sales delay reaches 8 seconds, this algorithm sets all lower level algorithms to maximum optimization, which includes reconfiguring the system to better distribute service requests through the servers. If the sales delay reaches 9 seconds, new users are refused from entering the system. These two conditions prevent the system from entering the degradation zone. For this reason, the graphs of figures 5 and 6 stop at 20 request/s, since higher request rates only happen when the system load is low.

At the **process abstraction level**, the algorithms try to optimize each server performance. Considering the case of the database server, a very simple orthogonal multi-variable model with only two of the variables contributing to the database delay is shown in figure 7. The network delay specification is passed down to the equipment level algorithm. The database load is further decomposed (figure 8) into a component due to the database backups, and a component due to user requests that are controlled at the service level algorithm.

The algorithm acts by suspending database backups when the delays exceed 1 second, and by turning them on again when the delays fall below 0.8 seconds.

At the **equipment abstraction level**, the algorithms optimize parameters related to network and equipment operation, according to the corresponding models. The model for the average network delay is shown in figure 9. The distributed system uses caches to optimize remote operations. Management acts on them too. When the cache sizes are increased, both the network traffic and average delays are reduced, but the processing delays on that host are slightly increased due to reduced processing resources (both CPU and RAM).

In addition, the routing algorithm can minimize communication costs, or network delays, at a management option, both corresponding to best-effort guarantees. To minimize costs, links between company's branches are disconnected when the traffic is less than 5% of the link capacity for 5 minutes, provided that connectivity is maintained. The links are restored when the load in any of the other links exceeds 80%. To minimize delays, all links are established, and traffic is routed through the fastest route.

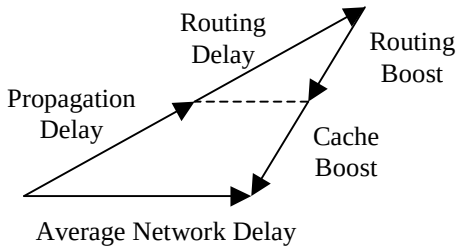

Figure 9: Contributions to the network delay.

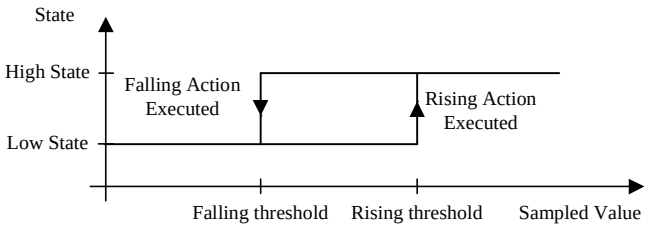

Figure 10: Threshold Operation.

Figure 11 shows the experimental results for a single network segment operation for a 90 seconds period, with a 20 request/s load. The instants where the algorithms acted are marked, according to the algorithm operation previously described. Additionally, during this period, the application cache is turned on when the application delay exceeds 1.4 seconds, and turned off when the delay drops below 1.1 seconds.

It should be noted that although the application delay exceeds the contracted limit of 2 seconds for a while, the top-level contract is still kept.

Most of the algorithm actions are converted to thresholds with hysteresis that when crossed trigger the execution of some action, as shown in figure 10. These thresholds are derived from the contracts established, and dynamically adjusted during the algorithm operation.

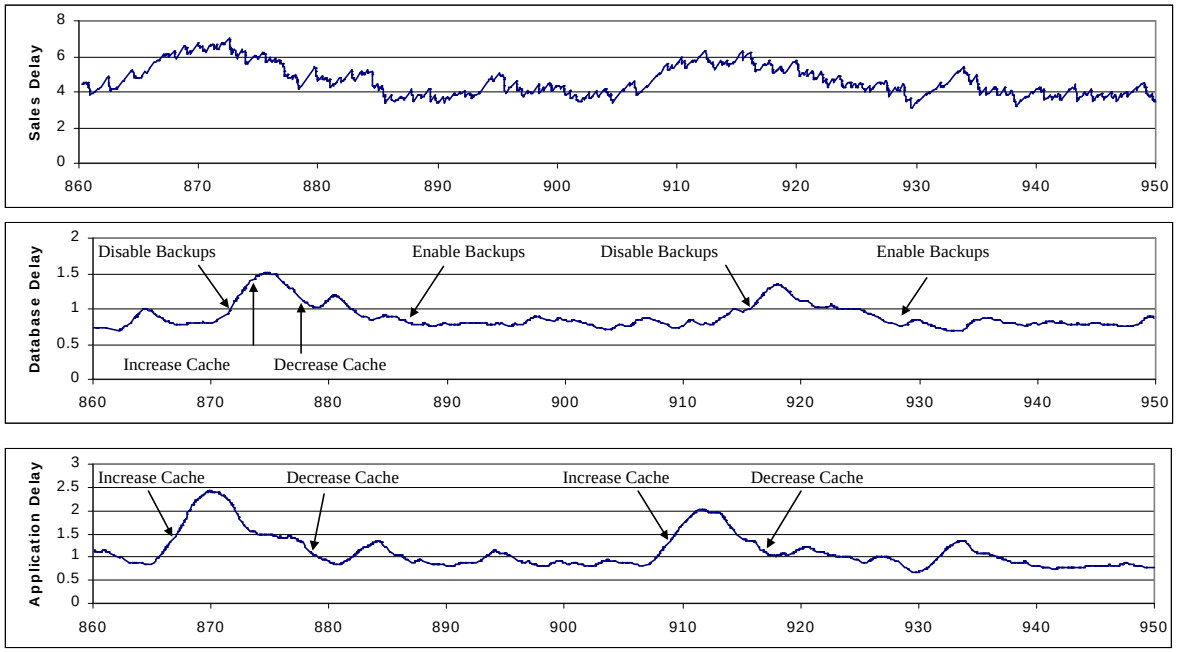

Figure 11: Experimental results from algorithm operation.

To avoid unpredictable algorithm actions caused by temporary fluctuations on the sampled variables, a moving average of the samples is used in most cases, instead of the samples themselves. This moving average is calculated so that each sample's weight is reduced to half for each half-life period, which in this example is 3 seconds. To avoid polling the variables, the algorithms just set callbacks with the necessary thresholds.

The planning algorithms at the service abstraction level analyze the service performance, faults, accounting, configuration and security information, helping to optimize the company's operation. In this example, when the thresholds are successively crossed on all database servers, corresponding to a history of service degradation, the algorithm warns the manager that a new database server should be deployed, the best location for it, and reassigns the regions to each server. This information can be determined based on the requests traffic matrix obtained from an RMON2 [RFC 2021] probe.

The planning algorithms at the process abstraction level start a database optimization before the established limit is exceeded.

The planning algorithms at the equipment abstraction level analyze the routes taken by the network traffic, defining virtual private network settings like traffic priorities, and reserving network resources, according to higher-level directives to optimize some services performance. These algorithms also suggest network modifications like splitting network segments, or moving machines between segments, to help enforcing network load and delay specifications.

This simple example shows that by controlling a few key variables, the algorithms can improve system performance, and additionally avoid entering the degradation zone.

5. Benefits of the Contract Approach

There are four aspects where the contract approach brings advantages: the first is the transparency towards certain aspects of system operation, having the possibility of just monitoring the QoS being obtained. Each entity with management responsibilities does its best to fulfill the contract and, as long as the contract holds, the other entities do not need to know how this is done. This approach can be combined with management by delegation [Yemini 91]: algorithms can be placed nearer the resources, reducing the overall management traffic on the network. The delegation of management algorithms decentralizes the management, adding scalability, flexibility, and fault tolerance.

The second advantage is the simplicity of the environment. The use of formal languages to specify the behaviour allows an automatic reuse of components to build an application. At each level, it is necessary to understand the descriptions of the available lower-level algorithms (in order to select and validate the choice) and to dynamically deploy, configure and activate the algorithms. The next step is monitoring and controlling their operation. Current mechanisms for defining management information (like SNMP MIBs) have much of the critical information specified as human-readable prose that is not suitable for automatic exploitation by generic management applications.

The third advantage is the framework provided by such environment. It provides simple guidelines to incorporate vendor dependent features to specific problems. Currently, most of the new vendor features are performed as private MIBs, which is a rather low-level and unstructured solution. By standardizing an algorithm environment, manufacturers are stimulated to provide management functions as management algorithms driven by the search for a solution to a problem and not so much by the provision of general-purpose management tools.

Finally, complex systems and networks are becoming distributed environments. Distributed processing aspects must be taken into consideration, as the managed system evolves autonomously due to its dynamic nature (e.g. routing algorithms). Algorithms can be designed to work with dynamic functional requirements providing rules to an otherwise potentially free space of intervention. In addition, different abstraction levels must be provided, as a manager expects to work at the highest possible abstraction level.

6. Related Work

Low abstraction level management paradigms are based on quality or health parameters derived from the managed information provided by the managed objects. [Goldszmidt 93] describes how to produce health functions. [Valimaa 95] proposes a method to produce quality factors from SNMP MIB variables using fuzzy logic. [Stadler 96] introduces a language for deriving management information for abstract models from the information provided by agents. [Anerousis 99] describes how to generate computed views of management information.

A few high-level paradigms for management have been proposed by the scientific community. They are usually based on artificial intelligence, distributed artificial intelligence techniques, rules or policies. The architecture proposed by the Imperial College's Department of Computing [Sloman 93] organizes the management requirements in policies and roles. Policies are rules specifying what can, or what must be done. Roles [Lupu 97] identify rights, duties, functions and interactions associated with a position in the company, resulting in a definition of a set of policies for each role. [Wies 94] proposes a policy hierarchy and a formal definition of policies to raise the abstraction level seen by the operators. [Koch 96] also proposes a policy hierarchy, using different language constructs for each abstraction level. Although policies represent a high-level

abstraction, at a level a human manager would expect, they usually restrict the network behaviour, instead of improving its operation. In addition, the refinement process has to be done for each high-level policy, limiting its reuse and requiring significant expertise. Algorithms, on the other hand, are active objects that check and improve network operation. Moreover, some algorithms are generic, so that they can be reused for several similar problems, just by changing the variables they control. The algorithms share the advantages of policies: indeed, some algorithms could be programmed with sets of policies.

Service level management [Lewis 98][Collins 96] is a comprehensive high-level method of specifying the management of business processes, the supporting services, and establishing contracts between service providers and users. However, if monitoring the service level agreement is a well understood process, real-time automatic enforcement of this contract is not, especially when the contract has to cover application-level parameters not directly related to network- and transport-level QoS parameters. The algorithms offer a generic solution for this problem.

The Open Distributed Processing (ODP) reference model [ISO 10746-2] defines a contract as an agreement governing part of the collective behaviour of a set of objects. A contract specifies obligations, permissions and prohibitions for the objects involved. The specification of a contract may include the roles and interfaces of the objects involved, QoS attributes, duration or periods of validity, indications of behaviour that invalidates the contract, and liveness and safety conditions. The working draft on QoS [ISO 10746-6] from ODP, extends the ODP concepts and architecture to allow the description of QoS in ODP systems, focusing mainly on modelling and specifying QoS. The concepts of ODP are general concepts that can also be applied to management. Although not originally based on ODP, our approach can be seen as an extension and adaptation to network management of these concepts. We define algorithms as objects that enforce contracted guarantees, which may include obligation, permission and prohibitions. Furthermore, a hierarchy of algorithms is used to refine the guarantees to multiple abstraction levels.

7. Conclusions and Further Work

The management requirements are expressed in four abstraction levels. The management application is built using a hierarchy of algorithms focused on solving specific management problems.

The management algorithms offer management guarantees over the managed systems, by automating management operations and hiding equipment dependencies. The conditions the algorithms should enforce are specified by contracts that configure each algorithm, and their fulfillment is monitored by QoS parameters. The algorithms are deployed into a dynamic management architecture using delegation.

The proposed system provides a high-level view of the management problems to the manager, by handling the lower-level ones.

The first experimental results show that the ideas proposed are adequate to network and systems management, accomplishing the proposed objectives. However, other examples have to be studied and implemented.

A few issues have been left for further study. Conflicts may arise between different contracts or different guarantees, resulting in opposing management operations that should be detected and resolved before enforcing the contracts. This task may have to be done in a top-level manager if there are several mid-level managers managing the same domain.

A method to determine if a contract can be fulfilled should also be developed. This might involve baselining the network operation to check if the imposed conditions are realistic. A contract editor should be developed to assist in writing and checking new contracts.

A high-level, management specific, language to program algorithms should be developed.

Acknowledgement

This research has been partially supported by PRAXIS XXI, under contract 2/2.1/TIT/1633/95.

References

[Alexandrov 97] Maxim Alexandrov, Alexander Hagin, Nikolai Hagin, Yuri Karpov, Vladislav Voinov. *Quality of Service management of the World Wide Web*. 4th Workshop of the HP OpenView University Association, OVUA'97, April 1997, Madrid, Spain.

[M.3010] *Principles for a Telecommunications Management Network.* ITU-T Recommendation M.3010. May 1996.

[Moffett 93] Jonathan D. Moffett, Morris S. Sloman. *Policy Hierarchies for Distributed Systems Management*. IEEE Journal on Selected Areas in Communications, Special Issue on Network Management, **11**(9):1404-1414, December 1993.

[Marriott 96] Damian A. Marriott, Morris S. Sloman. *Management Policy Service for Distributed Systems*. IEEE 3rd International Workshop on Services in Distributed and Networked Environments (SDNE'96), Macau, June 1996, IEEE Computer Society Press.

[Koch 96] Thomas Koch, Christoph Krell, Bernd Krämer. *Policy Definition Language for Automated Management of Distributed Systems*. 2nd International Workshop on Systems Management, IEEE Computer Society, 19-21 June 1996, Canada.

[Wies 95] René Wies. *Using a Classification of Management Policies for Policy Specification and Policy Transformation*. IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, California, USA. 1-5th May 1995.

[Aurrecoechea 98] Cristina Aurrecoechea, Andrew T. Campbell, Linda Hauw. *A Survey of QoS Architectures*. ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture, **6**(3):138-151, May 1998.

[Yemini 91] Y. Yemini, G. Goldszmidt, S. Yemini. *Network Management by Delegation*. International Symposium on Integrated Network Management, April 1991, pp. 95-107.

[Ptolemy] *Ptolemy Home Page*. http://ptolemy.eecs.berkeley.edu/

[RFC 2021] S. Waldbusser. *Remote Network Monitoring Management Information Base Version 2 using SMIv2*. IETF RFC 2021. January 1997.

[Goldszmidt 93] Germán Goldszmidt, Yechiam Yemini. *Evaluating Management Decisions via Delegation*. IFIP International Symposium on Network Management, April 1993.

[Valimaa 95] Harri Valimaa, Tapani Honkanen. *Producing Quality Factors of LAN Interconnection Services*. Proceedings INET'95.

[Stadler 96] Michael Stadler. *Mapping Management Information to Service Interfaces Supporting High-Level Network Management Applications*. DSOM'96 7th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquila, Italy, October 28-30, 1996.

[Anerousis 99] Nikolaos Anerousis. *An Architecture for Building Scalable, Web-based Management Services*. Journal of Network and Systems Management, **7**(1):73-104, Plenum Press, March 1999.

[Sloman 93] M. Sloman, J. Magee, K. Twidle, J. Kramer. *An Architecture for Managing Distributed Systems*. 4th IEEE Workshop on Future Trends of Distributed Computing Systems, 22-24 September 1993, Lisbon, Portugal, IEEE Computer Society Press, pp. 40-46.

[Lupu 97] Emil Lupu, Morris Sloman. *A Policy Based Role Object Model*. 1st Enterprise Distributed Object Computing Workshop (EDOC'97), Gold Coast, Australia, IEEE Press, October 1997.

[Wies 94] René Wies. *Policies in Network and Systems Management – Formal Definition and Architecture*. Journal of Network and Systems Management, Plenum Publishing Corp., **2**(1):63-83, March 1994.

[Lewis 98] Lundy Lewis. *Spectrum Service Level Management Definition, Offerings, and Strategy*. Technical Note lml-ctron-98-02, Cabletron Systems. March 30, 1998. `http://www.cabletron.com/white-papers/spectrum/slm.pdf`

[Collins 96] Terry Collins, Greg Edson, Timothy Lee. *Service Level Management Strategy*. The Enterprise Network Management and Troubleshooting Symposium. Hewlett Packard, October 1996.

[ISO 10746-2] *Open Distributed Processing - Reference Model: Foundations*. International Organization for Standardization, International Standard 10746-2 (ITU-T X.902), 1995.

[ISO 10746-6] *Working Draft for Open Distributed Processing - Reference Model: Quality of Service*. International Organization for Standardization, Document ISO/IEC JTC 1/SC 33 N 145 for Draft International Standard 10746-2 (ITU-T X.906), 1998.