Lecture Notes in Computer Science 2867
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
*Berlin*
*Heidelberg*
*New York*
*Hong Kong*
*London*
*Milan*
*Paris*
*Tokyo*

Marcus Brunner  Alexander Keller (Eds.)

# Self-Managing Distributed Systems

14th IFIP/IEEE International Workshop
on Distributed Systems: Operations and Management, DSOM 2003
Heidelberg, Germany, October 20-22, 2003
Proceedings

Springer

Volume Editors

Marcus Brunner
NEC Europe Ltd., Network Laboratories
Kurfürstenanlage 36, 69115 Heidelberg, Germany
E-mail: brunner@ccrle.nec.de

Alexander Keller
IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA
E-mail: alexk@us.ibm.com

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): C.2, K.6, D.1.3, D.4.4, K.4.4

ISSN 0302-9743
ISBN 3-540-20314-1 Springer-Verlag Berlin Heidelberg New York



Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de



Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein
Printed on acid-free paper SPIN: 10966747 06/3142 5 4 3 2 1 0

# Orchestrating Self-Managing Systems for Autonomic Computing: The Role of Standards

Thomas W. Studwell

IBM Corporation, Autonomic Computing Architecture and Technology Group
3039 Cornwallis Road, Research Triangle Park, NC 27709-2195, USA
studwell@us.ibm.com

Computing technology has progressed rapidly over the last several decades with implementations and applications that were unthinkable a decade ago now commonplace. The rate of progress, however, has brought its own cost. As large IT infrastructures grow more complex the cost of managing these systems has increased rapidly. As a result a greater percentage of the IT budget is going toward maintenance of the infrastructure rather than improving its benefit to the business.

One doesn't have to dig too deeply to understand that this increasing maintenance cost is directly related to the increase in complexity of computing technologies that have become so advanced that traditional manual management techniques are equally as apt to harm systems rather than enhance them. As each new technology strives to add enhancements for manageability, the very controls that were viewed as improving capability are actually weakening systems because the IT systems become unwieldy from the sheer number of adjustments whose interactions are unknown and virtually unknowable. Couple this increased complexity with the trend to heterogeneous distributed systems, and the complexity of the computing environments increases dramatically.

While, at first glance, the solution may seem to be a move back to very simple IT infrastructures, this is clearly not a viable alternative. The promise of e-business, distributed computing, and, eventually, Utility computing and Grid computing is just too great to ignore. For the IT Industry to bring these complex, unmanageable sytems under control it is necessary to move to self-managing systems where technology itself is used to manage technology. Autonomic systems in the human body provide the appropriate model for this progression. In this keynote, we will discuss how self-managing systems, similar to biological processes found in every living being will not only help solve today's problem of increasing complexity but, will, in fact, finally allow IT infrastructures to directly serve the corporation's business needs rather than being a static and poorly implemented attempt to automate yesterday's business processes.

This last point is vital – the IT industry must think in terms of solving business solutions rather than each supplier dwelling on the piece parts they supply to the infrastructure. This can only be accomplished through active participation in open standards where issues of interoperability and decomposition of function for self-managing systems can be properly resolved. In this keynote we will delineate the opportunities for, and challenges of, standardization of

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 1–2, 2003.

self-managing technologies. The IEEE will play an important role in these standardization efforts but coordination of work within the IEEE and other key computing standards bodies, such as DMTF, OASIS, GGF, and IETF, will be essential to the success of this effort. It is difficult to overstate the importance for standardization collaboration. It is not enough for localized domains to be self-managing; every component within the IT infrastructure must participate in an orchestrated way. Just as the human body has many discrete autonomic systems but all work toward a single goal, so must all the self-managing systems in the future IT infrastructure work in a uniform and predictable way.

In this keynote we will describe IBM's vision for autonomic computing and how we plan to apply the model of autonomic systems to self-managing systems with particular focus on problem determination, configuration, and optimization. At each point we will highlight the key standardization efforts underway with special emphasis on areas where standardization effort is necessary but not yet begun.

Ironically, the most prominent gap in standardization exists, not because it is perceived as unimportant, but precisely because it is recognized as essential to self-managing systems. The key area is Policy and the rules that govern self-managing systems. While each standards domain has a reasonable understanding of its own technology, policy related to the self-management of each domain is, to a large extent, determined by needs beyond the scope of the domain. Hence, without coordination of standards across domains the industry is deadlocked into uncertain or inadequate policy implementations. Breaking this logjam is critical to the advancement of self-managing system's standards. Further, breaking this logjam will allow the fundamental goals of the business to be converted into the standardized policies that govern the distributed IT domains throughout the enterprise.

As to the future, as self-managing systems are widely deployed, initial systems will have very rudimentary control functions. This will create new opportunities for research and product development in the area of analysis and control necessary to improve efficiency and accuracy of autonomic control. We will discuss these as well hoping to spark the interest of the DSOM community to join the call to advance the technologies of self-managing systems.

## Biography

Thomas Studwell is a Senior Technical Staff Member in IBM's Autonomic Computing Architecture and Technology team. In 1975, Tom joined IBM's Thomas J. Watson Research Center and has worked for a number of IBM divisions including Research, Microelectronics, Networking Systems, and Personal Computing Devices. Tom joined IBM's Autonomic Computing Architecture and Technology group in December, 2002, and is responsible for promoting autonomic computing technologies in open standards. Tom holds several patents, is an IEEE member, and has represented IBM on IEEE, PCI SIG, and other industry consortia workgroups.

# Generic Online Optimization of Multiple Configuration Parameters with Application to a Database Server

Yixin Diao[1], Frank Eskesen[1], Steven Froehlich[1], Joseph L. Hellerstein[1], Lisa F. Spainhower[2], and Maheswaran Surendra[1]

[1] IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
[2] IBM Server Group, 2455 South Rd., Poughkeepsie, NY 12601
{diao,eskesen,stevefro,hellers,lisa,suren}@us.ibm.com

**Abstract.** Optimizing configuration parameters is time-consuming and skills-intensive. This paper proposes a generic approach to automating this task. By generic, we mean that the approach is relatively independent of the target system for which the optimization is done. Our approach uses online adjustment of configuration parameters to *discover* the system's performance characteristics. Doing so creates two challenges: (1) handling interdependencies between configuration parameters and (2) minimizing the deleterious effects on production workload while the optimization is underway. Our approach addresses (1) by including in the architecture a rule-based component that handles interdependencies between configuration parameters. For (2), we use a feedback mechanism for online optimization that searches the parameter space in a way that generally avoids poor performance at intermediate steps. Our studies of a DB2 Universal Database Server under an e-commerce workload indicate that our approach can be effective in practice.

## 1 Introduction

The advent of e-Commerce has created a need for responsive and cost-effective information technology (IT) services. However, the increasing complexity of computing systems has resulted in a correspondingly larger human effort for system configuration, especially the optimization of configuration parameters. This paper describes a generic approach to automating configuration optimization. The approach is generic in that it is relatively independent of the target system for which the optimization is done.

Enterprise level software, especially middleware, has tens to hundreds of configuration parameters. For example, IBM's DB2 Universal Database Server has approximately 100 to 200 configuration parameters (e.g., buffer pool sizes, time delay for writing commit records, maximum number of database applications). The challenges here are well recognized, as evidenced by efforts such as IBM's autonomic computing initiative to develop self-managing systems [1]. In particular,

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 3–15, 2003.

addressing self-configuration and self-optimization often depends on subtleties in the workload and system configuration. This motivates the need for a generic approach that discovers the performance impact of configuration parameters by interacting with the target system. However, such an approach creates two challenges: (1) handling interdependencies between configuration parameters and (2) minimizing the deleterious effects on production workload while the optimization is underway (e.g., minimize oscillations, avoid large response times).

There are several efforts related to our work. Some researchers have studied the regulation of system resources to achieve policy objectives, especially using control theory [2]. Examples here include controlling buffer length in Internet routers [3] and response times for web service differentiation [4]. The problem we address differs from these efforts in that we seek to optimize a service level metric (e.g., response time) rather than regulate it, which typically means that a search is required. Thus, closer to our current work is [5], who describe a system that performs on-line optimization of a web server by using hill climbing techniques. However, the approach taken requires a detailed knowledge of the system being optimized in order to construct the queueing models. A similar concern arises with the approach in [6] who consider how to maximize profits based on queueing-theoretic formulas. [7] proposes a fuzzy control approach to minimize response time using a combination of feedback control system and qualitative insights into the effect of tuning parameters on QoS. However, only one configuration parameter is considered. [8] presents case studies of using randomly generated configuration settings for application servers. This approach is simple and generates good results for off-line parameter optimization, but the absence of a guided search may turn out to be problematic for online optimization (as discussed in Section 4). A further concern with all of the foregoing is that none of the approaches consider the architecture necessary to support online optimization, especially handling interdependencies between configuration parameters.

This paper describes a generic, online approach to optimizing configuration parameters. The approach builds on recent work in the area of metric discovery, especially the use of the Common Information Model (CIM) to discover metrics and configuration parameters and the architecture necessary to support this [9]. The problem of interdependencies between configuration parameters is addressed by architecting rule-based components on the target system that handle such interdependencies. The challenge of online search is addressed by employing an existing optimization technique that generally avoids poor performance at intermediate steps. We apply our architecture to IBM's DB2 Universal Database Server since automated configuration of databases is a pressing issue [10]. While there have been many efforts in this area (e.g., [11, 12]), our approach differs in that it requires no prior knowledge of the target system being optimized, although we do require access to the sensors (e.g., response time measurements) and effectors (e.g., buffer pool sizes).

The remainder of the paper is organized as follows. Section 2 details our control architecture, and Section 3 describes how we optimize the setting of

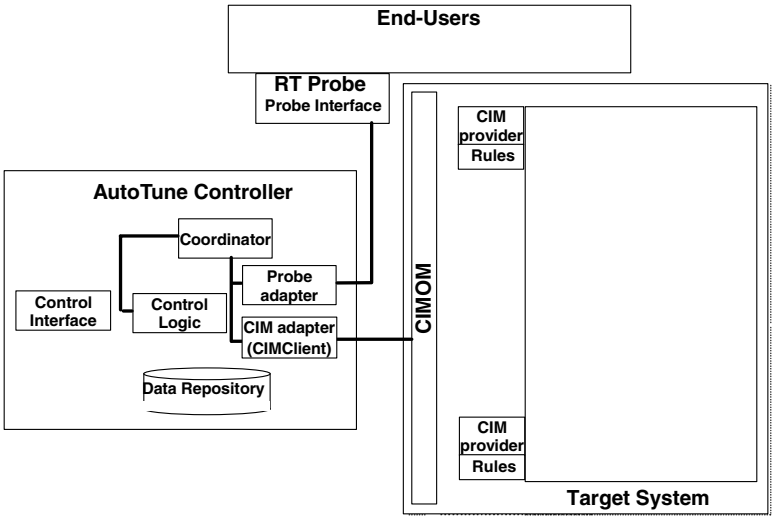



**Fig. 1.** Diagram of system architecture

configuration parameters. Section 4 presents the testbed setup and experimental results. Our conclusions are contained in Section 5.

## 2 System Architecture

This section describes the architecture used to support online optimization of configuration parameters.

Figure 1 depicts our architecture. The main elements are the target system being controlled, the AutoTune Controller that dynamically adjusts configuration parameters, and the RT Probe[1] that collects response times from the end-users (the workload). We use the Common Information Model (CIM) [13] to describe the target system (e.g., a database with tables and tablespaces) and the response time probe, especially their metrics (e.g., response times, rows read, sort times) and configuration parameters (e.g., buffer pool sizes). The architecture that we present extends our prior work with metric discovery [9]. Indeed, we discover configuration parameters in the same manner as is done with discovering resource metrics.

The operation of the system is depicted in Figure 2. Every control interval, the AutoTune Controller sends a request to the target system to modify a subset of the configuration parameters. The target system makes these modifications, and the RT probe provides data on the achieved performance. Based on this, the AutoTune Controller computes new values of the configuration parameters.

One point should be underscored in the foregoing. The only way that the AutoTune Controller knows the performance of a setting of configuration pa-

[1] In general, this could be a measurement source for any service level metric.

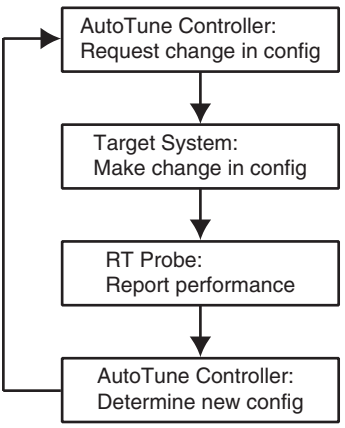



**Fig. 2.** High level flow of the operation of online optimization of multiple configuration parameters

rameters is to observe the system under those settings. Since this is taking place during normal system operation, caution must be exercised so that there is minimal impact on end-users both in terms of poor performance and the variability of performance.

Now consider the target system. Access to both metrics and configuration parameters is handled by the CIM Object Manager (CIMOM). The CIMOM provides a standard object-oriented interface to this information. The information itself is obtained by various CIM Providers. Note that CIM Providers have rules associated with them to handle conflicts that arise from interdependencies between settings of parameter values. For example, a CIM Provider for memory would handle requests for re-sizing buffer pools that result in demands that exceed memory constraints. That the rules are associated with CIM Providers is appealing in that the specifics of the constraints are known to the providers. A disadvantage is that there may be constraints that involve multiple CIM Providers, although we have not encountered these to date.

Now consider the element schemas used by the CIMOM. The CIM Providers extract data from various parts of the target system (e.g., table spaces, buffer pools) and provide this to the CIMOM. This design is done according to the principle that all descriptive and capability-related information of a managed element is modeled as properties of the class representing the resource itself, while its statistical data is put in an associated class, subclassed from `CIM_StatisticalData`. Specifically: (1) the `controllerName` property is used in multiple controller environments to specify which controller is to be used; (2) the `physicallyEnabled` indicator determines whether the provider is physically capable of setting a control (e.g., the DB2 provider is not capable of setting a control when DB2 is stopped), and (3) the `logicallyEnabled` indicator is intended to be set by the system administrator to override the controller.

The RT Probe provides response time information. Its element schema describes how to operate the probe (e.g., what synthetic transactions can be sent, the resource to which these transactions are sent) and the response times reported (e.g., by transaction type and resource).

The AutoTune Controller is an agent based system that uses the Agent Building and Learning Environment (ABLE) [14]. ABLE is a Java-based toolkit for developing and deploying hybrid intelligent agent applications. Built on top of ABLE is a general AutoTune agent framework that facilitates the construction of control agents by separating the control interface (e.g., probe adaptor, DB2 adaptor) with control logic (e.g., the direct search method described in Section 3). This general and extensible ABLE/AutoTune based controller architecture allows us to easily target our controller to a controlled element (for example, DB2) simply by adding an "adaptor" component that knows how to interface with the controlled element. The control logic typically requires a history of control actions. These data are accumulated by using the control interface to find relevant data for the element (by querying the element schema) and then subscribing to updates of element data (which are then placed in the controller's historical data repository).

## 3 Optimization Technique

From Figure 2, we see that configuration parameters are changed on an on-going basis. This places two requirements on the approach taken to optimization. First, once a "bad" setting of configuration parameters is identified, we want to move quickly back to "good" settings. Second, we want to minimize the variability in performance due to exploring parameter settings.

Many optimization methods have been studied and applied to solve real world problems (e.g., [15]). However, not all of them are suitable for online optimization. For example, gradient methods (e.g., steepest descent, conjugate gradient, Newton's [16]) are widely used. These techniques evaluate the derivative of performance with respect to configuration parameters, and change parameters in the direction in which they improve performance. However, these techniques are quite sensitive to noise, and noisy data are common in computing systems. Also, obtaining the gradients or Hessians information through approximations and using the line search algorithm to choose the step size require more evaluation samples. They are costly for online optimization. A second class of techniques that are fairly robust to noise are stochastic optimization methods (e.g., random search [8], genetic algorithms) in which randomness is used to evaluate different settings of configuration parameters. However, these approaches can have longer times to converge and more extreme variations in performance during convergence.

Our starting point is the Nelder-Mead simplex method [17, 18, 19], a robust version of a gradient method that has been used with success in many practical optimization problems. This technique is also referred to as a direct search method as it does not actually compute the gradient. Rather, it uses informa-

tion about local minima and maxima to determine the direction in which the parameter space should be searched.

Some notation is introduced for the following discussion. We denote the vector of configuration parameters by $\theta = (\theta_1, \cdots, \theta_n)$. Thus, each $\theta$ can be viewed as a point in $n$-space. Let $J(\theta)$ be the performance of the system if the setting of the configuration parameters is $\theta$. Our objective is to navigate points so that the best performance is achieved in a short time without extreme values of $J(\theta)$ at intermediate configurations. In the following, we assume that *best* means lowest value, as in minimizing response times (although the approach applies to maximizing values as well). That is, we want to find $\theta^*$ such that $\min_\theta J(\theta) = J(\theta^*)$. (Technically, what is found is only a local minimum, not a global minimum. This should be fine for most resource allocation problems where the cost function is convex and a local minimum is also a global minimum. However, if the function is not convex, random search or genetic algorithms may perform better.)

Below, we briefly summarize the steps in the direct search method. (1) Initialize: Define a simplex with $n+1$ vertices $\theta(1), \theta(2), \ldots, \theta(n+1)$; (2) Evaluate: Evaluate the performance (i.e., compute $J(\theta)$) of all vertices; (3) Navigate: Several moves should be considered including reflection (mirroring the worst vertex), contraction (converging towards the best vertex), and expansion (moving further along the good direction), which identifies a $\theta$ that replaces the worst vertex and forms a new simplex with better performance; (4) Test for completion: If the new simplex is sufficiently small, then the optimization is complete; otherwise, go to Step 2.

Figure 3 illustrates the direct search method for a simulation of two DB2 buffer pools in which response times are obtained from an analytic model. The four plots in the figure represent four iterations of the algorithm. Each contains a contour plot showing how the two configuration parameters affect response time. (The contour plot can be constructed because the example is generated by simulation.) Iteration 0 corresponds to Step 1 above. We see that the simplex is below and to the left of the region of small response times that are in the middle of the range of parameter values plotted. In iteration 1, a new simplex is constructed by applying Step 3 to move towards a region in which response times are smaller. The new simplex is constructed by using the new $\theta$ to replace the one with the largest response time. Iteration 2 results in a simplex constructed in a similar manner. In iteration 3, we see that the new point has a larger response time than the one previously encountered, but it is smaller than the vertex with the largest response time in iteration 2.

The online optimization technique that we employ handles some practical considerations from some well-known techniques [18, 19, 20]: (1) Re-evaluate $J$ at the vertices before the simplex is contracted to help convergence in the existence of stochastics, and when the simplex is converges to adapt to workload nonstationarity. (2) Limit and fix the smallest size of the simplex to better handle variations in workload and avoid unnecessary oscillations, which can be costly especially for online optimization of computer configurations. (3) Incorporate constraints on parameter interdependencies to handle boundary conditions. The

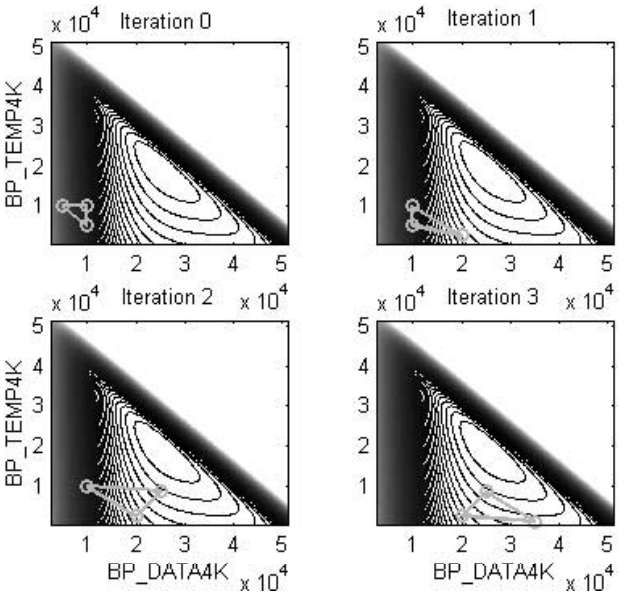



**Fig. 3.** Illustration of the direct search method for a simulation of two DB2 buffer pools. The contour plots illustrate the response times with axes for buffer pool sizes. The triangle in each plot depicts the simplex used in that iteration of the search

constraints are obtained from the CIM provider, and also enforced by it. For example, the buffer pool size cannot be negative, and the total buffer pool size cannot exceed a limit otherwise the system may crash. Thus, it is a constrained optimization problem. The projection method is used to enforce the parameter constraints.

## 4 Experimental Assessment

### 4.1 Testbed Setup

To assess the applicability of our approach to online optimization of configuration parameters, we study it in the context of a database server. IBM's DB2 version 8.1 provides a plethora of tuning parameters that can be changed programmatically in an online environment. Among them are some memory related parameters such as buffer pool size, package cache size, and sort heap size, which have drastic impact on database performance. In this paper, we consider buffer pool tuning.

Our evaluations are done using TPC-W, an industry standard e-commerce benchmark [21]. We used three buffer pools, `BP_INDEX4K` for indexing spaces, `BP_TEMP4K` for cached data, and `BP_DATA4K` for all remaining data. Generally,

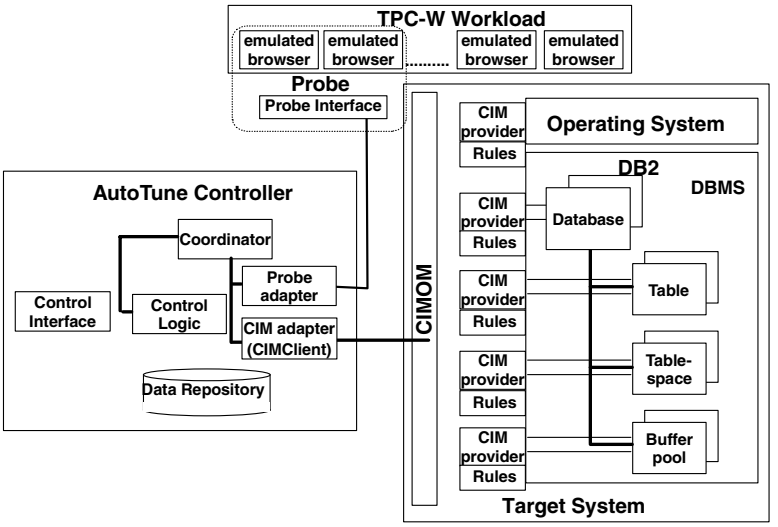



**Fig. 4.** Diagram of testbed system on which experiments were conducted

having larger buffer pool size results in smaller user response time. However, the total buffer pool size cannot go to infinity due the limited memory space.

Figure 4 displays how we instantiated the architecture in Figure 1. The testbed itself consists of three machines for the database server, the database client, and the AutoTune controller. The database server machine is an IBM RS/6000 model 7044-170 with 768 megabytes of RAM storage. This machine uses the AIX operating system and contained IBM DB2 V8.1 server and the CIM server code including DB2 CIM providers. The database client machine, an IBM RS/6000 model 7044-270 with 1 gigabyte of RAM storage, also uses the AIX operating system. This machine drives the client application using DB2 V8.1 client support. The client application is an emulated browser (EB), an online ordering load generator using TPC-W to emulate database access characteristics of a web e-commerce environment. The AutoTune controller runs on an IBM model T20 ThinkPad with Windows 2000.

Our experiments proceed as follows. A set of 50 emulated browsers (EBs) were running which executed transactions against the database server according to the TPC-W benchmark specifications. A subset (5) of these EBs were also instrumented to provide client side response time for the AutoTune controller. The AutoTune controller used the direct search method to determine the buffer pool sizes and the desired values were sent to the database for real time adjustment. In this environment, while the DB2 client and controller machines were lightly loaded, the DB2 server was always driven at 100% CPU utilization. The control interval was set as 20 minutes. The buffer pool sizes were changed at the start of the interval but the client side response time was measured 5 minutes later (for 15 minutes) to avoid the transient effect of buffer pool resizing.

**Table 1.** Results of the first six iterations of direct search on the testbed

| Sample | BP_DATA4K (4K pages) | BP_INDEX4K (4K pages) | BP_TEMP4K (4K pages) | RT (sec.) |
|---|---|---|---|---|
| 1 | 29199 | 23359 | 2919 | 15.8 |
| 2 | 14599 | 23359 | 2919 | 17.8 |
| 3 | 29199 | 11679 | 2919 | 16.7 |
| 4 | 29199 | 23359 | 1459 | 19.2 |
| 5 | 19465 | 15572 | 4379 | 15.0 |
| 6 | 14598 | 11679 | 5839 | 18.4 |

### 4.2 Experimental Results

Table 1 displays some data obtained from applying direct search to our testbed system. Intuitively, the direct search method operates by evaluating different buffer pool settings within a small region, and moving towards the direction that reduces the response time. Since we use three buffer pools, the initial simplex was composed of 4 vertices and 4 sample buffer pool combinations were selected and evaluated accordingly (as shown in the first four lines in Table 1). The first sample takes the default buffer pool size, and the reset three are defined by halving the size, one at each sample. The constraint is that the total size cannot go beyond 150% of the default total size. Among the four samples, the first sample had minimum response time (15.8 seconds) and the fourth sample had maximum response time (19.2 seconds). Next, we evaluate the reflection point as shown in the fifth sample, which is defined as the mirror point of the maximum point through the centroid. Since its response time (15.0 seconds) is even smaller than the minimum response time we get in the first sample, we evaluate the expansion point as shown in the sixth sample, which was expanded further along this direction. However, the expansion results in higher response time (18.4 seconds). Therefore, according to the direct search algorithm, we complete the first iteration and use the vertices at sample 1, 2, 3, and 5 to compose the simplex for the next iteration.

Figure 5 displays the results of applying direct search until convergence is achieved. In the upper plot, the dashed line indicates the buffer pool size for BP_DATA4K, the dotted line indicates BP_INDEX4K, the dashed-dotted line indicates BP_TEMP4K, and the solid line indicates the total buffer pool size. Note that as the optimizing continues, BP_DATA4K and BP_TEMP4K increase, and BP_INDEX4K decreases. This indicates that indexing space (BP_INDEX4K) need not be so large, but more space is required for cached data (BP_TEMP4K) and the remaining data (BP_DATA4K). During the search there is one instance of a large response time (e.g., the 16th sample) since not all the moves in navigation are heading towards the correct direction. However, the wrong moves will not be continued. Generally, the algorithm convergence time is up to the applications, e.g., the number of configuration parameters, the initial parameter values, the performance function shape, and the convergence criteria. Our experiments indicate a reasonable convergence requires roughly 10*(number of parameters) samples.

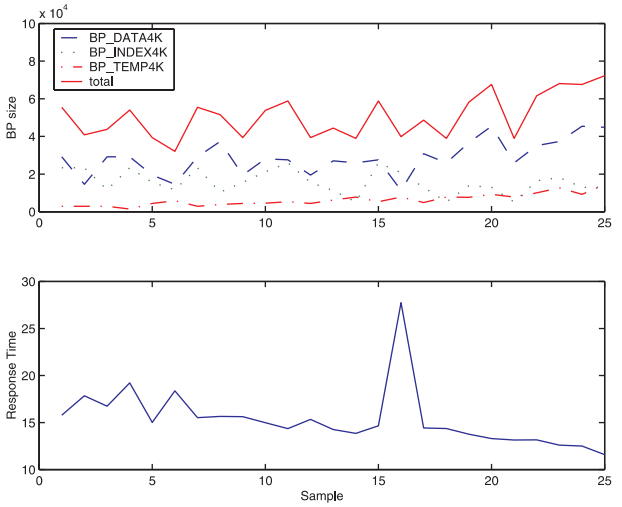



**Fig. 5.** Experimental results of DB2 buffer pool tuning with the direct search method

It is also interesting to compare direct search with the techniques used in [8] for off-line parameter optimization (but we used it here in an on-line scenario). We conducted experiments in which the total buffer pool size is chosen from a uniform distribution over the range 300,000 to 900,000 4K pages, and then the selected total pool size is randomly sub-divided into three buffer pools. Some of the experimental results are shown in Figure 6. Note that this random approach achieves performance comparable to direct search. However, there is much more variability in performance during the optimization, and considerably longer response times at intermediate values. This comes no surprise as the direct search method chooses the navigation directions based on the sample values that are just evaluated, but the above method goes randomly without guidance.

## 5 Conclusions

Optimizing configuration parameters to improve performance is time-consuming and skills-intensive. We have proposed an architecture and an algorithm for automating this process through the simultaneous optimization of multiple configuration parameters in a manner that minimizes prior knowledge of the target system (the system being optimized).

Central to our approach is minimizing knowledge of the target system. We accomplish this by *discovering* the effect of configuration parameters on the performance of the target system by making changes to configuration parameters and observing the effects of these changes. Doing so has two implications. First, we must handle interdependencies between configuration parameters (e.g., limits

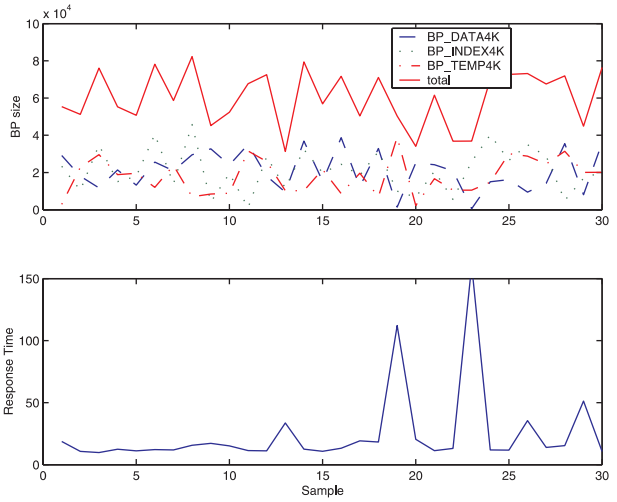



**Fig. 6.** Experimental results of DB2 buffer pool tuning with the stochastic optimization method

on the total size of a buffer pool). Second, the search for better configurations should be done in a manner that is unlikely to degrade performance excessively and restores performance quickly when it is degraded. Our approach to the first incorporates rules into the CIM Providers of the target system so that changes in configuration parameters are properly constrained. The second is addressed by using the direct search algorithm, a feedback mechanism that exploits the performance function to optimize performance with guidance. This generic approach is expected to be applicable for a wide class of online optimization problems such as configuring server parameters to increase system utilization, differentiating services from different customers, and balancing workload among multi-tiered systems.

In this paper we have applied our architecture and approach to IBM's DB2 Universal Database Server to optimize the setting of buffer pool sizes. This results in approximately a 25% response time reduction in an e-commerce benchmarking environment. We compare these results with an alternative - randomly selecting configuration settings. While the random approach achieves almost the same reduction in response time, its intermediate settings of configuration parameters result in highly variable performance and some very large response times.

## Acknowledgements

We would like to thank Dr. Rajarshi Das for many valuable discussions on applying the direct search method for a related Apache Web server optimization problem.

## References


[1] IBM, "Autonomic computing: IBM's perspective on the state of information technology," *available at http://www.research.ibm.com/autonomic/*, 2001. 3

[2] G. F. Franklin, J. D. Powell, and A. Emani-Naeini, *Feedback Control of Dynamic Systems*. Reading, Massachusetts: Addison-Wesley, third ed., 1994. 4

[3] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *INFOCOM*, 2001. 4

[4] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. Son, and M. Marley, "Performance specifications and metrics for adaptive real time systems," in *Proceedings 21st IEEE Real Time Systems Symposium*, pp. 13–24, Nov. 2000. 4

[5] D. Menasce, D. Barbara, and R. Dodge, "Preserving QoS of e-commerce sites through self-tuning: A performance model approach," in *Proceedings of 2001 ACM Conference on E-commerce*, 2001. 4

[6] Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *Proceedings of the ACM Conference on Electronic Commerce*, 2001. 4

[7] Y. Diao, J. L. Hellerstein, and S. Parekh, "Optimizing quality of service using fuzzy control," in *Proceedings of Distributed Systems Operations and Management*, 2002. 4

[8] M. Raghavachari, D. Reimer, and R. Johnson, "The deployer's problem: Configuring application servers for performance and reliability," in *Proceedings of the International Conference on Software Engineering, Portland, OR*, 2003. 4, 7, 12

[9] Y. Diao, F. Eskesen, S. Froehlich, J. L. Hellerstein, A. Keller, L. Spainhower, and M. Surendra, "Generic on-line discovery of quantitative models for service level management," in *Proceedings of IEEE/IFIP Symposium on Integrated Network Management*, 2003. 4, 5

[10] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback, "Self-tuning database technology and information services: from wishful thinking to viable engineering," in *International Conference on Very Large Data Bases*, 2002. 4

[11] G. M. Lohman and S. S. Lightstone, "Smart: Making db2 (more) autonomic," in *Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China*, 2002. 4

[12] J. Rao, C. Zhang, G. M. Lohman, and N. Megiddo, "Automating physical database design in a parallel database," in *SIGMOD*, 2002. 4

[13] "Common Information Model (CIM) Core Model, Version 2.4," white paper, Aug. 2000. http://www.dmtf.org/var/release/Whitepapers/DSP0111.pdf 5

[14] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao, "ABLE: A toolkit for building multiagent autonomic systems," *IBM Systems Journal*, vol. 41, no. 3, 2002. 7

[15] D. G. Luenberger, *Linear and nonlinear programming*. Addison-Wesley, Reading, MA, 1984. 7

[16] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh, "Online response time optimization of apache web server," in *Proceedings of the 11th International Workshop on Quality of Service*, pp. 461–478, 2003. 7

[17] J. A. Nelder and R. Mead, "A simplex method for function minimizatioin," *Computer Journal*, 1965. 7

[18] F. H. Walters, J. L. R. Parker, S. L. Morgan, and S. N. Deming, *Sequential Simplex Optimization: A technique for improving quality and productivity in research, development, and manufacturing*. CRC Press, 1991. 7, 8

[19] C. H. Brooks, "An introduction to amoeba," *available at http://nexus.cs.usfca.edu/~brooks/papers/amoeba.pdf* 7, 8

[20] J. O. Kephart, R. Das, and J. K. MacKie-Mason, "Two-sided learning in an agent economy for information bundles," in *AmEC IJCAI*, 1999. 8

[21] W. D. Smith, "TPC-W: Benchmarking an ecommerce solution," in *http://www.tpc.org/tpcw* 9

# Eos: An Approach of Using Behavior Implications for Policy-Based Self-Management

Sandeep Uttamchandani[1], Carolyn Talcott[2], and David Pease[1]

[1] IBM Almaden Research Center, San Jose CA
{sandeepu,dpease}@us.ibm.com
[2] SRI International, Menlo Park CA
clt@csl.sri.com

**Abstract.** Systems are becoming exceedingly complex to manage. As such, there is an increasing trend towards developing systems that are self-managing. Policy-based infrastructures have been used to provide a limited degree of automation, by associating actions to system-events. In the context of self-managing systems, the existing policy-specification model fails to capture the following: a) The impact of a rule on system behavior (behavior implications). This is required for automated decision-making. b) Learning mechanisms for refining the invocation heuristics by monitoring the impact of rules.

This paper proposes *Eos*; An approach to enhance the existing policy-based model with behavior implications. The paper gives details of the following aspects:

- Expressing behavior implications.
- Using behavior implications of a rule for learning and automated decision-making.
- Enhancing existing policy-based infrastructures to support self-management using Eos.

The paper also describes an example of using Eos for self-management within a distributed file-system.

## 1 Motivation

Systems are becoming extremely complex to manage. The cost of administration is becoming a significant percentage (75-90%) of the Total Cost of Ownership (TCO) [6,16]. Jim Gray in his Turing award speech "What next? - A dozen IT research goals" [9] emphasized the need for buildings systems that are self-managing. IBM's initiative on autonomic computing aims to build self-managing systems, reducing the demand on system administrators.

System management in the real world is done by administrators. Their primary task is to ensure that the behavior goals specified by Service Level Agreements (SLA) are met. As such, they employ the following action loop: monitoring → analyzing

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 16-27, 2003.

required changes to system behavior → tuning system parameters and invoking system-services.

A self-managing system can be defined as one in which the system by itself decides the configuration parameters to be set and system-services to be invoked, in response to a specific system state. The aim of this adaptation is to meet the specified goals. Another important aspect of a self-managing system is its ability to evolve and learn from its actions i.e. self-learning

Currently, policy-based infrastructures have been used to provide a limited degree of automation [15]. In simple words, a policy is defined as a set of rules that are based on ECA i.e. Event → if (Condition) → then (Action). These rules map system states to setting of tunable parameters and invocation of system services [5].

There are multiple approaches for specifying policies. They can be specified as a programming language that is processed and interpreted as a piece of software [8,10] or in terms of a formal specification language [17,19] or the simplest approach is to express policies as a sequence of rules. The IETF has chosen rule-based policy representation in its specifications [1].

## 2 Problem Statement

Existing rule-based policy specifications lack the capability to express semantics required for automated decision-making and self-learning. There is no systematic approach to define the following:

- The impact of the rule on system behavior. This mapping is the essence for automated decision-making that the system uses to decide the rule(s) to be invoked.
- Refining the invocation heuristics of the rules i.e. self-learning. Each time a rule is invoked, its impact of system can be recorded to refine future decision-making.

Eos is an approach that extends the existing policy-based infrastructures for providing self-management semantics. The key contributions of this paper are:

- Extending existing rule-based semantics for self-management specifications.
- Using the extended semantics for automated decision-making and self-learning.
- Describing the modules to be added to existing policy-based infrastructures to support the self-management semantics.

The paper is organized as follows. Section 3 enumerates the terminology. Section 4 gives a bird's eye-view of Eos. Section 5 formalizes the Eos concepts using a vector-space model. Section 6 describes a real-world example of self-management within a distributed file-system. Section 7 describes implementation details namely specification template, strategies for self-learning and decision-making and the Eos framework. Section 8 discusses the related work followed by the conclusion.

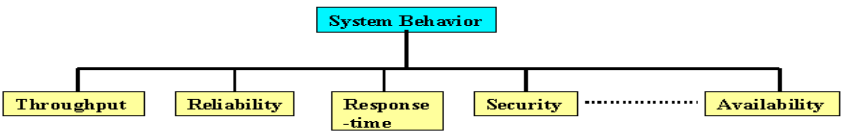



**Fig. 1.** Dimensions of behavior

## 3 Terminology

***Dimensions of Behavior***

The term "behavior" is generally used loosely to describe the observable characteristics of the system. These characteristics can be specified using abstractions such as QoS goals, transaction-properties [3], etc. In each of these abstractions, behavior is a composition of multiple dimensions. Figure 1 represents system behavior to be composed of dimensions such as throughput, latency, reliability, security, availability and so on.

***Behavior Implications***

It is the impact of a rule on system behavior. It is expressed in terms of dimensions of behavior.

***Management-Knob***

Broadly classified, administrators have two sets of controls for managing the behavior of the system. First, there are configuration parameters that are either application-specific or system variables such as buffer-size, number of concurrent threads, etc. Second, there are system services that can be invoked in certain scenarios. For example, in a distributed file system, there are services such as backup, data-migration, and replication. These parameters and services are together referred as "management knobs."

***Low-Level System-State***

It represents details of the system such as resource utilization and system events. Resource utilization is expressed in terms of cpu, i/o and network bandwidth being used. Events can specify system conditions such as disk is 95% full or errors such as network failures, or disk failures.

***Workload Characteristics***

It captures the properties of the application request-stream. For example, in a file-system, workload characteristics include read-write ratio, sequential/random, etc. Workload characteristics play a significant role in deciding the impact of the management-knob on system behavior. For example, increasing the Prefetch-knob makes sense only when the access pattern is sequential.

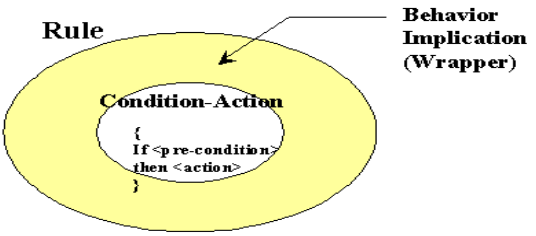



**Fig. 2.** Extending existing specification model with Behavior implications

## 4 Bird's Eye-View of Eos

In the existing policy-specification model, rules are defined as condition-action pair expressed using if-then semantics. Eos extends this specification by defining a wrapper around the existing rule (Figure 2). The wrapper represents the behavior implications of the rule and also the workload characteristics on which it is dependent.

In simple words, the working of Eos can be described as follows: When the assigned goals are not met, a trigger is generated. The decision-making module scans through the repository using behavior implications, low-level pre-conditions, and workload characteristics. Based on this analysis, it decides the rule(s) that should be invoked. Each time a rule is invoked, its impact is monitored and used to refine the behavior implications.

## 5 Eos Concepts

To formalize the Eos model, we represent the concepts using an n-dimensional vector space. Vector space models have also been used in other areas of research such as information retrieval [22]. To make the discussion concrete, we consider the example of invoking the data-replication knob within a distributed system. A more elaborate example is covered in the next section

### 5.1 Behavior Implications

Let $t_1, t_2, \ldots t_n$ be the terms used to describe the dimensions of system behavior. For each term there is a corresponding vector $t_i$ in a vector space. This is shown in Figure 3. This vector space is referred to as the behavior space. At any given time, the state of the system is represented as a point within the behavior space.

$$\text{Current-state} = (a_1 t_1, a_2 t_2, \ldots, a_n t_n)$$

where $a_i$ is the current value along the dimension $t_i$.

The behavior implication of a rule B(r) is represented as a difference vector between the new state $(b_1t_1, b_2t_2, \ldots, b_n t_n)$ and the previous state $(a_1t_1, a_2t_2, \ldots, a_nt_n)$

before the rule is invoked. This vector is a sparse matrix with the diagonal representing the values of the dimensions it affects (assuming the dimensions are independent). A compact representation is represented as the following summation:

$$B(r) = \sum_{i=1,n} (b_i - a_i)\, t_i$$

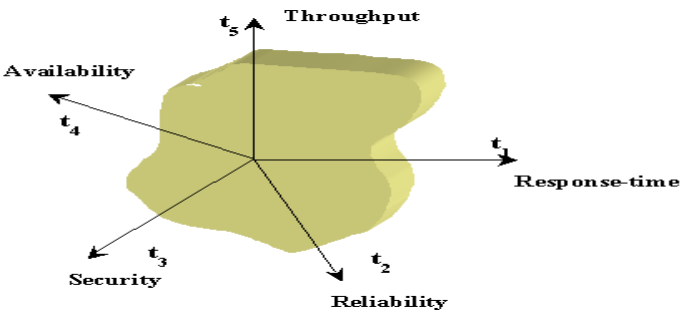



**Fig. 3.** Vector space to represent system behavior

As an example, the behavior implication of the data-replication rule is a vector along the dimensions of throughput, latency and availability. It is represented as:

$$B(\text{data-replication}) = [(0.3)\text{Throughput} - (0.1)\text{Latency} + (0.2)\ \text{Availability}]$$

where invoking replication improves throughput and availability by 30% and 20% respectively, and degrades latency by 10%.

### 5.2 Self-Learning

The behavior implication of a knob is not a constant vector. For example, in the case of data-replication knob, it is a function (*g*) of the workload characteristics (read/write ratio), the degree change of the knob-value (number of replicas) and the current value of the knob (going from 1 replica to 2 replicas has a different impact on behavior than going from 5 to 10 replicas).

The behavior implication vector is a key component for automated decision-making. Hence, the aim of self-learning is to refine the behavior implication vector by learning the dependency function (*g*). Each time a rule is invoked, the changes to system behavior are monitored. The following feedback information is recorded:

- Current behavior value and percentage change in value by invoking the knob ($\beta$).
- Workload characteristics when the knob was invoked ($\gamma$).
- Current value of the knob ($\eta$).

Self-learning refines the behavior implication vector and is represented by:

$$S[B(r)] = \sum_{j=1,n} [g(\beta, \gamma, \eta)]_j\ t_j$$

where the composite function (g) is learnt by using machine learning approaches such as neural networks.

### 5.3 Automated Decision-Making

This is a 3-step process. The first step is to analyze the current state and determine the goals that are not met, the workload characteristics and the low-level system state. Next, a list of candidate rules is generated. This is done by matching the workload characteristics and pre-conditions of the rules to the current system-state.

The final step is to decide the combination of rule(s) to be invoked from amongst the list of candidate rules. One of the strategies for combining the behavior implication vectors is using the following recursive algorithm (Figure 4):

- Generate the target vector starting from the current-state to the desired-state.
- At each stage, select the unit vector whose cosine angle with the target vector is greatest. The step size of the vector is k, where 'k' signifies the degree of instability of the system and is less than the target vector.

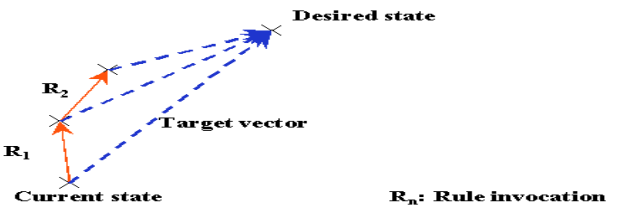



**Fig. 4.** Strategy for combining rules

## 6 Example: A Self-Managing Distributed File-System

Consider the example of managing a distributed file system within a data-center. Let database and multimedia be the two primary applications running on top of this file-system. The database is serving a complex workload consisting of OLTP and decision-support while the multimedia application is serving a Video-on-demand (VOD) service. The database and multimedia applications are tuned assuming the underlying file system meets goals specified in terms of throughput, latency, reliability, and availability.

To meet the desired goals, the administrator tunes the file-system using the management-knobs, enumerated in table 2. The policy specification of these knobs consists of two parts. First, the low-level pre-conditions for invoking the knob. Second, the wrapper that extends rules with behavior implications.

**Table 1.** Illustrating current system state

| | Goals achieved | % Change required [% Change Tolerated] |
|---|---|---|
| Throughput | × | 15[-] |
| Response-time | ☗ | 0 [2] |
| Availability | × | 8[-] |
| Security | ☗ | 0 [Authentication removable] |
| Reliability | ☗ | 0 [35] |

Table 1 shows the current values of the assigned goals. Each of the goals is quantified by parameters that can be monitored. For example reliability can be quantified by MTBF, Time-to-repair (TTR), Number of Failures, type of Failures.

As shown in table 1, the throughput and availability goals are not being met. Based on the low-level system-state, assume that the following management-knobs from table 2 qualify the pre-condition: Pre-fetch size, Data replication service and Volume migration service. Decision-making involves analyzing the behavior implications of each of the management-knobs:

- Pre-fetch size: Will improve throughput, but does not have an impact on availability.
- Replication: Will help throughput and replication, but will have a negative impact on latency, due to consistency requirements of the replicas.
- Volume migration: Has a positive impact on throughput, availability and response-time.

As shown in Table 1, the value of response-time cannot be changed by more than 2%. Thus, based on the above analysis, the volume migration service is invoked. Similarly, there can be scenarios where more than one rule is invoked, using the vector-addition strategy described in Section 5.3.

**Table 2.** Information specified by the Administrator

*↑ Positive Impact ↓ Negative Impact ↑+ Positive Impact ↓- Negative Impact ↔ Unspecified Impact*

| **Capability** | **Low-level system-state (Pre-conditions) and workload dependencies** | **Behavior Implication** | | | | |
|---|---|---|---|---|---|---|
| | | **Throughput** | **Latency** | **Availability** | **Security** | **Reliability** |
| *Configuration Parameters* | | | | | | |
| Clean-delay | Memory available && high write/read ratio | ↔ | ↔ | | | ↔ |
| Pre-fetch size | Sequential access-pattern | ↑+ | ↔ | | | |
| Data Integrity Check | Application imposed requirement | ↓- | ↓- | | ↑ | ↑ |
| *System Services* | | | | | | |
| Load balancing | Resources not uniformly utilized | ↑ | ↑ | | | |
| Data replication | Access pattern read-intensive | ↑ | ↓ | ↑+ | | |
| Volume migration | Non-uniform utilization of disks | ↑ | ↑ | ↑ | | |
| Data Backup | Low system-load OR system errors | ↓ | ↓ | | | ↑ |

After volume migration is invoked, its impact on the behavior is recorded, along with the workload and low-level system state. This information is used to re-fine the implication vector. Assume that in the steady-state, the invocation of volume migration actually degraded throughput. The implication vector is updated as:

B(volume migration) = -(0.15) Throughput + (0.04) Latency + (0.2) Availability

# 7 Implementation Details

## 7.1 Specification Template for Behavior Implications

The behavior implication template is specified as a wrapper around the existing rule. The specification template is shown in Figure 5. The specifications are treated as initial guidelines and are refined via self-learning. For the specification of behavior dimensions, the administrator specifies the impact using an intuitive description space. For example the degree of impact is described using terms such as positive, negative, positive++, negative-- and unspecified.

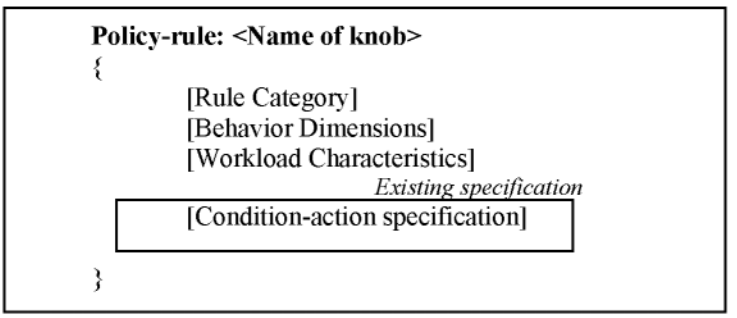



**Fig. 5.** Specification template

The specification grammar is enumerated in Appendix. As an example, the data-replication service is represented as follows:

**Rule: Data-replication {**
**Rule Category**
Type = Service; Range = Boolean; Resource = Storage;
**Behavior Dimensions**
Throughput Positive Always; Latency Negative;
Availability Positive Depends;
**Workload characteristics**
Primary = Read/write ratio
**Condition-Action** *(Existing rule)*
{
**If** (num_reads/num_writes > 0.9)
**Then** replication = ON
}
}

### 7.2 Implementation of Self-Learning

When a rule is invoked, its impact on behavior depends on the following:

1. The current value of the knob.
2. The current behavior state.
3. The workload characteristics.

In a simplified case (assuming a single variable for behavior and workload characteristics), these factors create a 3-dimensional learning space. This space is divided into sub-spaces, referred as "zones."

Each time a rule is invoked, the change in the behavior is recorded as a function of the percentage change in knob value. This function could be linear, polynomial, quadratic, exponential, etc. Similarly, the fact that the administrator invoked the knob can also be recorded within the learning-space (Figure 6).

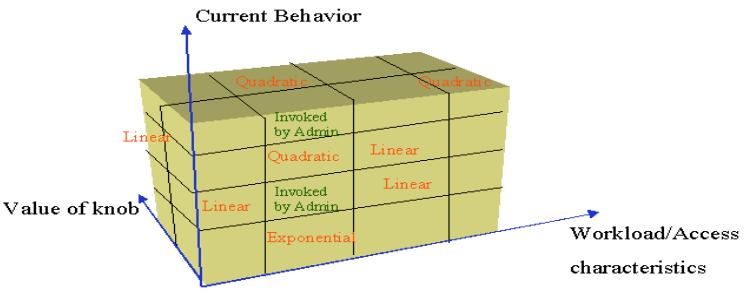



**Fig. 6.** Self-learning by dividing space into "zones"

### 7.3 The Eos Framework

Existing policy-based infrastructures consist of 3 key entities: A repository, Policy Enforcement Point (PEP) and Policy Decision Point (PDP). The PDP acts as a rule-filter i.e. based on the system-events, it determines the rules in the repository that are applicable and directs them to the PEP.

Figure 7 illustrates the Eos Framework. It working can be defined as a sequence of three stages:

***1. Rule Filter: Pre-qualification of Management-Knobs***

The rule-filter analyzes the low-level system state and determines the configuration-knobs that can be invoked.

***2. Capability Broker: Decision-Making for Selecting Knob***

As shown in the figure, the Capability Broker compares the specified goals and their current-values. It decides the knobs to be invoked.

### *3. Self-Learning*

After the rule is invoked, its impact on system-behavior is monitored and recorded in the rule repository.

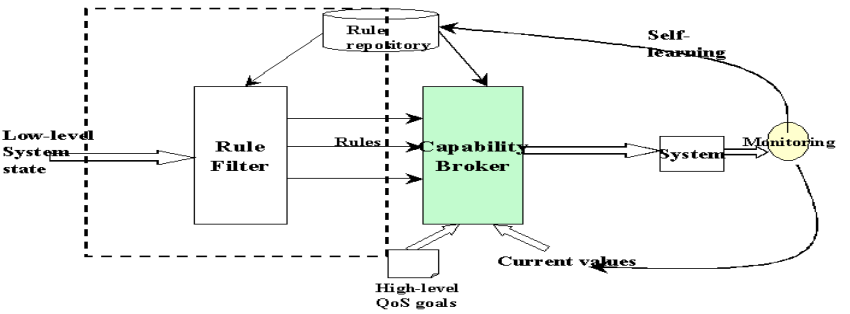



**Fig. 7.** Components of the Eos Architecture

The existing policy-based infrastructure supports the mapping of low-level system events to the invocation of management-knobs. The dotted line in the figure 7 illustrates this. Adding one more reasoning layer i.e. the Capability Broker to the existing infrastructure, allows for higher-order operations on rules namely automated decision-making and self-learning.

## 8 Related Work

Mark et al [13] propose an approach to separate the goal from the base rule specification. In other words, they create a mapping between the rule and user-requirements, making it easy for valiadation and usage. The Eos approach is in a similar direction, but aims to encodes goals for automated decision-making and refinement.

Zinky et al. [7] present a general framework, called QuO, to implement QoS-enabled distributed object systems. The QoS adaptation is achieved by having multiple implementations. Each implementation is mapped to an environment and a QoS region. This approach is static as it does not implement semantics for reasoning about the various possible configurations.

[4] describes an approach to build self-tuning systems using genetic algorithms. It relies on the fact that each system parameter is tuned by an individual algorithm and the genetic approach decides the best combination. This approach does not allow refinement of the decision-making based on self-learning.

GridWeaver [2] and other projects [18] aim for configuration of large scale computation fabrics such as the grid. Their primary concern is with the initial system configuration. The goals of Eos are complementary to this effort and aims for dynamic QoS management.

## 9 Conclusion

This paper is aimed as a starting-point in describing a systematic approach to build self-managing systems, by extending the existing rule-based management model. The key points of the Eos approach are: First, it defines behavior implications to capture the mapping between the rule and its impact on system-behavior. Second, it describes how these behavior implications can be used for automated decision-making and self-learning i.e. adding information to the rules based on the feedback from previous decisions.

## References

[1] The IETF Policy Framework Working Group. http://www.ietf.org/html.charters/policy-charter.html
[2] The GridWeaver Project. http://www.gridweaver.org/
[3] B. Sabata, S. Chatterjee, M. Davis, J. Sydir, T. Lawrence. Taxonomy for QoS Specifications. Workshop on Object-oriented Real-time Dependable Systems (WORDS ), 1997.
[4] D. Feitelson, Michael Naaman. Self-Tuning Systems IEEE Software 16(2), pp. 52-60, 1999.
[5] D. Verma. Simplifying Network Administration using Policy based Management. IEEE Network Magazine, March 2002.
[6] E. Lamb. Hardware Spending Matters. *Red Herring*, pages 32–22, June 2001.
[7] J. A. Zinky, D. E. Bakken, and R. D. Schantz. Architectural Support for Quality-of-Service for CORBA objects. Theory and Practice of Object Systems, Vol. 3(1), 1997.
[8] J. Fritz Barnes and Raju Pandey. ``CacheL: Language Support for Customizable Caching Policies. In Proc of Web Caching Workshop (WCW ), March 1999.
[9] J. Gray "What Next? A Dozen Information-Technology Research Goals," ACM Turing Award Lecture, June 1999, MS-TR-99-50.
[10] J. Hoagland, "Specifying and Implementing Security Policies Using LaSCO, the Language for Security Constraints on objects". Ph.D. Dissertation, UC Davis, March 2000.
[11] J. Matthews, D. Roselli, A. Costello, R. Wang, and T. Anderson. Improving the performance of log-structured file systems with adaptive methods. In *Proc. of ACM SOSP*, 1997.
[12] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. A*CM TOCS*, pages 108–136, Feb. 1996.
[13] M. Bearden, S. Garg, W. Lee: Integrating Goal Specification in Policy-Based Management. POLICY 2001: 153-170.
[14] M. Seltzer and C. Small. Self-monitoring and self-adapting operating systems. In *Proc. of HOTOS Conf.*, pages 124–129, May 1997.
[15] M. Sloman, E. Lupu. Security and management policy specification. IEEE Network, pp. 10-19, March-April 2002.

[16] N. Allen. Don't Waste Your Storage Dollars. Research Report, Gartner Group, March 2001.
[17] N. Damianou, N. Dulay, E. Lupu, and M Sloman, "Ponder: A Language for Specifying Security and Management Policies for Distributed Systems", Imperial College, UK, Research Report DoC 2001, Jan. 2000.
[18] P. Anderson and A. Scobie. Large scale Linux configuration with LCFG. In *Proceedings of the Atlanta Linux Showcase*, pages 363–372, Berkeley, CA, 2000. Usenix.
[19] R. Darimont, E. Dalor, P. Massonet and A. Van Lamsweerde. GRAIL/KAOS: An Environment for Goal Driven Requirements Engineering. In Proc. of International Conference on Software Engineering, pp. 58-62, 1998.
[20] S. Chaudhuri and V. Narasayya. AutoAdmin "what-if" index analysis utility. In *Proc. of ACM SIGMOD Conf.*, pages 367–378, June 1998.
[21] S. Mullender, Distributed Systems. Addison-Wesley 1993.
[22] Salton, G. and McGill, M. J. *Introduction to Modern Information Retrieval*. McGraw Hill, New York, 1983.

## Appendix: Specification Grammar

**[Rule Category]**
<Parameter-type>:= Tunable-parameter | System-service
<Parameter-range>:= Integer | Boolean| Floating-point
<Resource-type>: CPU | Memory | Network| Storage

**[Behavior Dimensions]**
[<Dimension><Impact-Degree><Impact-probability>]*

<Dimension>:= Throughput | Response-time| Reliability | Availability | Security | Error-recovery
<Impact-Degree>:= Positive | Negative | Positive++ | Negative-- | Unspecified
<Impact-probability>:= Always | Mostly | Never | Depends | Unspecified

**[Workload Characteristics]**
<Primary parameter>:= <Parameter>
<Secondary parameters>:= [<Parameter>]*

<Parameter>:= Read/Write ratio | sequential/random ratio | Request-size | Request-rate | Burst-interval | think-time

**[Condition-Action Specification]**
Rule:= Specified using existing rule-based languages

# On the Algebraic Structure of Convergence

Alva Couch and Yizhan Sun

Tufts University, Computer Science
Medford MA 02155, USA
{couch,ysun}@cs.tufts.edu
http://www.cs.tufts.edu

**Abstract.** Current self-healing systems are built from "convergent" actions that only make repairs when necessary. Using an algebraic model of system administration, we challenge the traditional notion of "convergence" and propose a stronger definition with improved algebraic properties. Under the new definition, the structure of traditional configuration management systems is a natural emergent property of the algebraic model. We discuss the impact of the new definition, as well as the changes required in current convergent tools in order to conform to the new definition.

## 1 Introduction

There are at present three main approaches to automated configuration management. The "imperative" approach of ISconf and its relatives[16, 25, 26] is closest to human practice; it expresses configuration actions as a series of commands applied to a system that possesses a known initial state. The "convergent" approach of Cfengine and its relatives[3, 4, 5, 6] expresses configuration as a set of known states to assure in an otherwise unknown system. The "generative" approach of LCFG and its relatives[1, 2, 14, 15, 18, 19, 24] controls state via regeneration of individual configurations from an overarching ruleset, rather than by making incremental changes to existing configurations.

The three configuration management strategies differ in their ability to handle reordering of configuration tasks in time. Imperative commands can branch based upon existing conditions created by other configuration management commands; the order of operations always matters in any imperative scheme[16, 26]. Meanwhile, Cfengine's "convergence" proponents argue that not only is order irrelevant, but novel self-healing configuration management strategies can be based upon *random* ordering of convergent actions[12, 17]. Which is the proper strategy: deterministic ordering, random convergent ordering, or generative reconstruction? We think that part of the key to this question lies in *algebraic* properties of the primitive operations for the three kinds of methods.

One problem with current "convergent" strategies is that a composition of individual convergent actions need not be convergent as a composite action. We propose a very simple algebraic model that justifies a new definition of convergence, in which a composition of convergent actions remains convergent.

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 28–40, 2003.

We argue that this definition is "more correct" because the definition allows us to derive an appropriate algebraic structure for an effective self-healing system.

## 2 Semigroups of Actions

We first develop an algebraic model of configuration management. This model adopts some curious conventions in order to make a minimal set of structural assumptions about configuration management. We then allow algebraic structure to arise from the axioms we choose for the system. In this way, we can analyze the axioms based upon the system properties that they imply when used.

Configuration actions are commands or subroutines that change the state of the managed system. We make no other assumptions about their nature. Let $P$ represent a set of actions. Each action $p \in P$ acts upon a system $X$ to produce a change in configuration. Configuration actions might include things like:

- replace `/etc/inetd.conf` with the one at `foo:/bar/repo/inetd.conf`
- replace `/etc/inetd.conf` with the one at `cat:/dog/repo/inetd.conf`
- delete all udp protocol lines in `/etc/services`.
- cd into `/usr/src/ssh` and run `make`.

etc. One can compose actions $p, q \in P$ in the natural way, by performing one after the other from right to left, e.g., $p \circ q \circ X = p(q(X))$. We also use composition notation for the final step $q(X) = q \circ X$ because a system itself is already composed of actions, even those such as "buy system from vendor Y". The system itself is some kind of action (e.g., a recipe for building it) but that action may not be in the set of actions $P$ that potentially *configure* $X$.

To simplify notation, we ignore parameters for actions. Similar actions with different parameters are treated as differing actions. For example, "run tftp with source directory `/home/tftp`" and "run tftp with source directory `/tftpboot`" are separate and distinct actions for our purposes.

## 3 Sequences of Actions

Given $P$ a set of actions, we form the set of all sequences of actions $\mathcal{F}(P)$ given by

$$\mathcal{F}(P) = \{\overline{p} = p_1 \circ p_2 \circ \cdots \circ p_{k-1} \circ p_k \mid k > 0, p_i \in P, 1 \leq i \leq k\} \cup \{\varepsilon\} \qquad (1)$$

where $\varepsilon$ is the *identity action* that consists of doing nothing at all. This is the *free semigroup with identity* over the alphabet $P$. Clearly $\mathcal{F}(P)$ is an infinite set, even though $P$ is finite.

The free semigroup $\mathcal{F}(P)$ consists of all possible sequences of actions. Many beliefs about the nature of actions in $P$ can be expressed as equivalence relations $\approx_i$ on $\mathcal{F}(P)$, where $\approx_i$ is a set of pairs $(\overline{p}, \overline{q})$ where $\overline{p}, \overline{q} \in \mathcal{F}(P)$. As usual, equivalence relations are reflexive ($(\overline{x}, \overline{x}) \in \approx_i$ for all $\overline{x} \in \mathcal{F}(P)$), symmetric ($(\overline{x}, \overline{y}) \in \approx_i \rightarrow (\overline{y}, \overline{x}) \in \approx_i$), and transitive ($(\overline{x}, \overline{y}), (\overline{y}, \overline{z}) \in \approx_i \rightarrow (\overline{x}, \overline{z}) \in \approx_i$ ).

In our physical interpretation of actions acting upon systems, two sequences of actions are equivalent if they accomplish the exact same result.

**Definition 1.** *Two configuration actions are equivalent if they achieve the same effect and are interchangeable in all contexts.*

In abstract terms, this means that equivalent actions are interchangeable within a sequence of actions:

**Definition 2.** *An equivalence relation* $\approx$ *is an* action equivalence *on* $\mathcal{F}(P)$ *if for all* $\overline{x}, \overline{y}, \overline{a}, \overline{b} \in \mathcal{F}(P)$, $\overline{a} \approx \overline{b} \leftrightarrow \overline{x} \circ \overline{a} \circ \overline{y} \approx \overline{x} \circ \overline{b} \circ \overline{y}$.

In other words, the substitution principle works. For the rest of this paper, we assume that all equivalences on sequences of actions are action equivalences according to this definition.

## 4 Convergent Actions

One concept of equivalence is that the actions are all convergent under the accepted definition of convergence[3, 10, 12]. An action $p$ is *idempotent* with respect to an action equivalence $\approx$ if repeating the action has the exact same effect as doing it once, i.e., $p \circ p \approx p$. A set of actions $P$ is *idempotent* if each individual action $p \in P$ is idempotent.

**Definition 3.** *We say a set of actions* $P$ *is* weakly convergent *if* $P$ *is idempotent as a set.*

This is the definition of convergence that controls actions in many common configuration management tools. In physical terms, an action $p$ is idempotent if repeating the action has no effect. Our algebraic definition of convergence does not include discussion of other attributes of convergence, such as non-intrusiveness. Non-intrusiveness is not an algebraic property, just an implementation detail.

In this paper, we will commonly study classes of actions by constructing an action equivalence relation that describes their properties.

**Definition 4.** *A weakly convergent set of actions* $P$ *(Definition 3) determines an action equivalence relation (Definition 2)* $\approx_1$ *on elements* $\overline{p}, \overline{q} \in \mathcal{F}(P)$.

According to this definition, $\overline{p} \approx_1 \overline{q}$ whenever $\overline{p}$ and $\overline{q}$ are sequences of actions that differ only in the number of times they repeat specific elements in order. For $p_1, p_2, ..., p_k \in P$ and a set of integer powers $\{n_i \mid n_i \geq 1, 1 \leq i \leq k\}$, the composition $p_1^{n_1} \circ p_2^{n_2} \circ \cdots \circ p_k^{n_k} \in \mathcal{F}(P)$ is equivalent under $\approx_1$ to any other sequence with differing choices of powers $n_i > 0$.

Unfortunately, the definition of weak convergence in Definition 3 has an undesirable property. The composition of idempotent actions need not be idempotent itself:

**Proposition 1.** *There are idempotent actions* $p$ *and* $q$ *where* $q \circ p$ *is not idempotent as a composite action.*

*Proof.* Idempotence of an action means that there is no difference in the *resulting state* of the system after repeating the action. As a very simple counterexample, suppose that there are two configuration state parameters $x$ and $y$, where the initial or "baseline" state of the configuration is that $x = y = 0$. Let $p$ be the action "if ($x$==1) then $y$=2", and let $q$ be "$x$=1". Clearly repeating either $p$ or $q$ has no effect, so they are idempotent in isolation. Also, $q \circ p$ just sets $x$ to one (remembering that actions are applied from right to left), but $(q \circ p) \circ (q \circ p)$ sets $y$ to 2 as well. Clearly $q \circ p$ is not idempotent as an action if we start from the baseline state $x = y = 0$. If, e.g., one sets $x$ and $y$ initially to 1, rather than 0, $q \circ p$ *is* idempotent.

This proof illustrates the general principle that idempotence and convergence of operations is only meaningful relative to a choice of *baseline state* of a system: the state of the system before configuration actions begin. In this paper, we will presume that the baseline state is appropriately limited so that idempotence is present, and defer study of the structure of the baseline state for later work.

The reason that a composition of actions is not idempotent is that in the proof above, one action $p$ is *stateful*, so that *when* it is called determines what it does. There is a stronger characterization of what it means to be truly convergent that assures that compositions of actions are always idempotent.

**Definition 5.** *A set of actions $P$ is* pairwise stateless *with respect to an action equivalence relation $\approx_i$ if for any two actions $p, q \in P$, $p \circ q \circ p \approx_i p \circ q$.*

In other words, repeating $p$ *before* $p \circ q$ (remembering that we are evaluating actions from right to left) has no effect.

**Definition 6.** *A set of actions $P$ is* stateless *with respect to an action equivalence relation $\approx_i$ if for any action $p \in P$ and any sequence of actions $\overline{q} \in \mathcal{F}(P)$, $p \circ \overline{q} \circ p \approx_i p \circ \overline{q}$.*

**Lemma 1.** *A set of actions $P$ is stateless with respect to an action equivalence relation $\approx$ iff it is pairwise stateless with respect to the same relation.*

*Proof.* If $P$ is stateless, it is clearly pairwise stateless. So assume that $P$ is instead pairwise stateless and proceed by induction on the length of $\overline{q} \in \mathcal{F}(P)$. If $q$ has length 1, then it consists of one element $x \in P$ and

$$\begin{aligned} p \circ \overline{q} \circ p &= p \circ x \circ p && \text{hypothesis} \\ &\approx_i p \circ x && \text{pairwise statelessness} \\ &= p \circ \overline{q} && \text{hypothesis} \end{aligned} \tag{2}$$

If we assume that the lemma is true for compositions $\overline{q}$ containing $n$ components, then for $\overline{q} = q_1 \circ \cdots \circ q_n \circ q_{n+1}$ containing $n+1$ components,

$$\begin{array}{rcll} p \circ \overline{q} \circ p & = & p \circ q_1 \circ \cdots \circ q_n \circ q_{n+1} \circ p & \text{definition} \\ & = & (p \circ q_1 \circ \cdots \circ q_n) \circ q_{n+1} \circ p & \text{associativity} \\ & \approx_i & (p \circ q_1 \circ \cdots \circ q_n \circ p) \circ q_{n+1} \circ p & \text{hypothesis} \\ & = & p \circ q_1 \circ \cdots \circ q_n \circ (p \circ q_{n+1} \circ p) & \text{associativity} \\ & \approx_i & p \circ q_1 \circ \cdots \circ q_n \circ (p \circ q_{n+1}) & \text{pairwise statelessness} \\ & = & (p \circ q_1 \circ \cdots \circ q_n \circ p) \circ q_{n+1} & \text{associativity} \\ & \approx_i & (p \circ q_1 \circ \cdots \circ q_n) \circ q_{n+1} & \text{hypothesis} \\ & = & p \circ q_1 \circ \cdots \circ q_n \circ q_{n+1} & \text{associativity} \\ & = & p \circ \overline{q} & \text{definition} \end{array} \tag{3}$$

so that $p \circ \overline{q} \circ p \approx_i p \circ \overline{q}$ and we are done.

**Definition 7.** *A set of actions $P$ is* convergent *with respect to an action equivalence relation $\approx$ if it is both idempotent and stateless with respect to that relation.*

Note that if $P$ contains the empty (identity) action $\varepsilon$, then statelessness straightforwardly implies idempotence: $p \circ p = p \circ \varepsilon \circ p \approx p \circ \varepsilon = p$.

Statelessness can be an elusive property in practice. The conditional `chmod` command "`chmod u+X file`" is convergent but not stateless. The "`u+X`" flag sets the owner execute flag if any execute flag is set for owner, group, or all. Interspersing a command that changes other execute flags, e.g., "`chmod g+x file`" or "`chmod a-x file`", can change the effect of a second instance of "`u+X`". The command "`chmod 755 file`" is by contrast both convergent and stateless.

## 5 Equivalent Actions

We can now define what it means for stateless actions to be equivalent.

**Definition 8.** *For any set of actions $P$, let $\approx_2$ represent the set of equivalences required by Definitions 2 and 7.*

In particular, $\overline{p} \approx_2 \overline{q}$ if there is a sequence of substitutions according to the definition that will transform $\overline{p}$ into $\overline{q}$.

Definition 7 makes the situation in Proposition 1 impossible. For $p, q \in P$, $q \circ p \circ q \circ p \approx_2 (q \circ p \circ q) \circ p \approx_2 (q \circ p) \circ p \approx_2 q \circ (p \circ p) \approx_2 q \circ p$.

For any set $S$ and equivalence relation $\approx$, the *factor set* $S/\approx$ of the set and the relation is a partition of the set into pairwise disjoint subsets $S_i$, each of whose elements are equivalent according to the equivalence relation. For a set of actions $P$, consider the factor set $\mathcal{F}(P)/\approx_2$. This is the set of *distinct actions* represented by $\mathcal{F}(P)$ under the equivalence $\approx_2$.

**Lemma 2.** *Every element in $\mathcal{F}(P)$ is equivalent under $\approx_2$ to one containing only the leftmost instance of each duplicated action. Thus $\mathcal{F}(P)/\approx_2$ consists of a finite number of subsets.*

*Proof.* Let $x \in \mathcal{F}(P)$. The goal is to express the word $x$ as another equivalent word $x'$ possessing only the leftmost instance of each distinct action from $x$. If $x$ consists of one action $p \in P$, let $x' = x$ and we are done. So presume that the lemma is true for strings of $n$ objects and suppose that $x$ consists of $n+1$ objects $p_1 \circ \cdots \circ p_{n+1}$. Then $y = p_1 \circ \cdots \circ p_n$ consists of $n$ actions and the lemma applies, so that there is a sequence $y'$ equivalent with $y$ and containing only the leftmost instance of each action in $y$. Then $x \approx_2 y \circ q_{n+1} \approx_2 y' \circ q_{n+1}$. Now if $q_{n+1}$ is not identical to some element of $y'$, $y' \circ q_{n+1}$ consists of unique elements and is an appropriate choice for $x'$. If it is identical to some $q_i$ that is a term of $y'$, then by Definition 7, $x \approx_2 y' \circ q_{n+1} \approx_2 y'$ and $y'$ is an appropriate $x'$.

The purport of this proof is that even though the free semigroup $\mathcal{F}(P)$ is infinite, we only "care about" a finite factor set of that infinite set.

**Lemma 3.** *Every element $\overline{p}$ of $\mathcal{F}(P)$ is* idempotent *with respect to $\circ$ and $\approx_2$, i.e., $\forall \overline{p} \in \mathcal{F}(P), \overline{p} \circ \overline{p} \approx_2 \overline{p}$.*

*Proof.* Given $\overline{p} \in \mathcal{F}(P)$, apply Lemma 2 to create an equivalent composition $\overline{p}' \approx_2 \overline{p}$ that contains no duplicates. Then

$$\begin{aligned} \overline{p} \circ \overline{p} &\approx_2 \overline{p}' \circ \overline{p}' && \text{Definition 1} \\ &\approx_2 \overline{p}' && \text{Lemma 2} \\ &\approx_2 \overline{p} && \text{construction} \end{aligned} \tag{4}$$

and $\overline{p}$ is idempotent with respect to $\approx_2$.

**Lemma 4.** *If we define for $\widetilde{p}, \widetilde{q} \in \mathcal{F}(P)/\approx_2$, $\widetilde{p} \mathbin{\widetilde{\circ}} \widetilde{q}$ as the unique $\widetilde{r} \in \mathcal{F}(P)/\approx_2$ such that*

$$\{\overline{a} \circ \overline{b} \mid \overline{a} \in \widetilde{p}, \overline{b} \in \widetilde{q}\} \subseteq \widetilde{r} \tag{5}$$

*then $\widetilde{\circ}$ is well defined and $(\mathcal{F}(P)/\approx_2, \widetilde{\circ})$ is a semigroup.*

*Proof.* The nature of equivalence is agreement on order of first appearance of each action. By Lemma 2, $\widetilde{p}$ and $\widetilde{q}$ have unique elements $\overline{p} \in \widetilde{p}$ and $\overline{q} \in \widetilde{q}$ with minimal length, and the sequence $\overline{p} \circ \overline{q}$ can be transformed by Lemma 2 to a similar representative element $\overline{r}$ of $\widetilde{r}$. The choice of $\overline{r}$ uniquely determines the contents of $\widetilde{r}$ by Lemma 2 and $\widetilde{\circ}$ is thus well-defined.

Note that for $\overline{p} \in \widetilde{p} \in \mathcal{F}(P)/\approx_2$ and $\overline{q} \in \widetilde{q} \in \mathcal{F}(P) \approx_2$, $\overline{p} \approx_2 \overline{q}$ exactly when $\widetilde{p} = \widetilde{q}$.

A semigroup whose elements are all idempotents is called a *band*. We have thus shown that:

**Theorem 1.** *If $P$ is a finite set of convergent actions obeying the equivalence $\approx_2$ and $\widetilde{\circ}$ is the operation defined in Equation 5, $(\mathcal{F}(P)/\approx_2, \widetilde{\circ})$ is a finite band.*

Note that this is not true in general if equivalence is expressed instead according to Definition 3. We have already shown that there are primitive actions $p, q$ for which $p \circ q$ is not idempotent. Not only is $\mathcal{F}(P)/\approx_1$ not idempotent; *it may not be a semigroup at all.* It is only a semigroup if one can show that for some operation $\overline{\circ}$ and every $\widetilde{p}, \widetilde{q} \in \mathcal{F}(P)/\approx_1$, $\widetilde{p} \mathbin{\widetilde{\circ}} \widetilde{q} \subseteq \widetilde{r}$ for some uniquely determined $\widetilde{r} \in \mathcal{F}(P)/\approx_1$. This is possible but depends on the nature of particular actions in $P$.

## 6 Value of Idempotence

The properties and possible structures of idempotent semigroups have been very extensively studied[20, 21, 22]. As a summary, we quote the following well-known results without proof.

A *subsemigroup* $(R, \circ)$ of a semigroup $(S, \circ)$ is a subset $R \subseteq S$ closed under $\circ$, i.e., for $r_1, r_2 \in R$, $r_1 \circ r_2 \in R$. A *commutative band* $(S, \cdot)$ is one in which all actions commute, i.e., for $x, y \in S$, $x \cdot y = y \cdot x$. A band $Q$ is a *matrix band* if it is isomorphic to some band $(\Gamma \times \Delta, \cdot)$ where $\Gamma$ and $\Delta$ are (arbitrary) disjoint alphabets, and for $(x, y), (z, w) \in \Gamma \times \Delta$, $(x, y) \cdot (z, w) = (x, w)$. In other words, a matrix band is a particularly simple kind of band. Note in particular that elements of a matrix band never commute except with themselves, because $(x, y) \cdot (z, w) = (x, w)$ while $(z, w) \cdot (x, y) = (z, y)$.

**Theorem 2.** *A semigroup all of whose elements are idempotents is a commutative band of matrix bands of unit groups ([22], Corollary 2.14, page 320).*

The proof of this theorem is too long to include, but the meaning for configuration management tools is straightforward. A configuration management tool implements a set of actions $P$, where we have shown that under the definition of convergence we consider correct, $\mathcal{F}(P)/\approx_2$ is a finite band. Its actions can be partitioned into a set of disjoint non-commutative subsets $\mathcal{B}_1, \ldots, \mathcal{B}_n$ that, considered as elements of a larger semigroup, commute. In this semigroup, the product of two elements $\mathcal{B}_i$ and $\mathcal{B}_j$ is that unique semigroup $\mathcal{B}_k$ containing all products of pairs $b_i \in \mathcal{B}_i$ and $b_j \in \mathcal{B}_j$. Each member $\mathcal{B}_i$ of the larger commutative band represents all appropriate values for one parameter, and can be partitioned further into subsets that form a semigroup of non-commutative members $\mathcal{B}_{i1}, \ldots, \mathcal{B}_{ik}$ representing particular values for a parameter.

We have thus proven that configuration parameters exist in all cases, without assuming that they do. Existence of parameters is an *algebraic* property that arises from assumptions of action equivalence, idempotence, and statelessness!

## 7 Meaning of Commutativity

Commutivity or non-commutivity of idempotent actions has a rather simple meaning.

**Definition 9.** *Two idempotent actions $a, b$ are* consistent*[12] iff they commute, i.e., $a \circ b = b \circ a$.*

Consistent actions *agree in intent* on how to arrange a configuration; inconsistent actions do not. Note that *orthogonal* actions that affect differing parts of the system are automatically consistent as well.

To understand this in a simple case, consider each action to be operating on an extremely large vector of bits called "the configuration". Every action asserts some fixed values for a subset of bits of the configuration. Actions that agree

on any common values commute, including actions that act on different parts of the configuration and thus cannot conflict. Conflicting actions do not commute.

As a concrete example, suppose that configuration consists of five bits $a_1 \ldots a_5$. A configuration is then a set of values for all five, i.e., a mapping from $\{a_i\} \rightarrow \{0, 1\}$. Configuration actions are then assertions of values of particular integers. If configuration action $B = \{a_1 := 1; a_5 := 0\}$ , this commutes with $C = \{a_1 := 1; a_2 := 1\}$ but not with either $D = \{a_1 := 0; a_2 := 0\}$ or $E = \{a_5 := 1\}$.

## 8 Implications for Configuration Management Tools

We have shown so far that particularly nice properties are exhibited by any set of configuration primitives possessing statelessness as described in Definition 7. Unfortunately, few actions in current configuration management tools are stateless.

### 8.1 Convergence and File Editing

The premier tool for convergent system administration is Cfengine[3, 4]. While Cfengine's abilities to copy files, replicate links, and change protections are fully convergent in the above sense, its extensive facilities for editing files are not convergent according to the above algebraic definition. Editing operations commonly conflict in intent, thus rendering them both non-commutative and problematic.

For example, if one reorders the Cfengine editing actions:

```
editfiles:
  all::
    { /etc/services
      deleteLinesMatching "tftp"
      appendIfNotPresent "tftp 6900/udp"
    }
```

(that have obvious meanings) there will be *no* line describing tftp instead of the *new* line describing tftp. File editing is procedural, not declarative; the action of deleting an old configuration line before adding a new one is non-commutative.

### 8.2 Rethinking File Editing

The argument above suggests that the true mathematical language of convergent system administration consists of self-consistent *assertions of state*, where sets of non-conflicting and consistent assertions *commute*. These assertions are *not* similar to editing commands; they assert intent without the implicit changes of intent that plague any attempt to edit during reconfiguration.

To address this problem, we completely rethought and redesigned the process of file editing to be declarative rather than procedural. Each file is treated like a database, and editing commands become precise assertions describing contents

in the file. Rather than the above Cfengine editing script, one might write the stateless assertion:

```
in /etc/services
  where field1=tftp
    line is "tftp 6900/udp"
```

or perhaps:

```
in /etc/services
  where field1=tftp
    make field2=6900/udp
```

or even (splitting up fields and renaming):

```
in /etc/services
  where service=tftp
    make port=6900 proto=udp
```

To have the effect of deleting a line, one asserts its absence:

```
in /etc/services
  where service=tftp
    omit
```

This is clearly more trouble than a simple edit; one must translate from imperative language to declarative. The interpreter of the declaration must understand the syntax of the file.

Adapting this idea to configuration files that are not linear lists is non-trivial but feasible. One must decompose each file into regions describing particular intents, and parse each separately. As files typically follow a natural hierarchy, the assertion language naturally follows that structure. For example, to configure a virtual server, one might write

```
in /etc/httpd.conf
  with virtual server foo.bar.com
    make port=9001
```

Accomplishing this level of declarative interaction with configuration is not a single-layer process. There are several layers of interaction involved, from low to high:

1. A *syntactic* layer that describes the syntax of each configuration file, i.e., a parser.
2. A *constraint* layer that describes which configuration states are meaningful.
3. A *policy* layer that describes which configuration states are allowed by site policy.
4. An *intent* layer that converses through the constraint and syntactic layers to create a particular behavioral effect.

5. A *validation* layer that insures that configuration does indeed change behavior.

A particular declaration, such as the examples above, travels downward (backward) through these layers to achieve a particular intent.

Some layers are generic; the syntax of particular configuration files is portable among all operating systems, as is the process of validating effects. For the most part, intent is also portable, but there are two kinds of constraints: those imposed by the system being configured and those configured by humans. Most of this infrastructure is thus reusable, with the exception of the layer in the *middle* of the sandwich that describes site policies. Creating the layers is significant work, but most of it must be done once and is reusable everywhere.

### 8.3 Generative Management

A *generative configuration management tool* expresses actions in terms of the complete replacement of configuration by new configuration. A typical tool replaces all files at each configuration step, so that each configuration action expresses a state for every configuration file. The simplest generative model is "wipe the disk, re-install, and regenerate all configuration files", as in MIT's Project Athena. More complex generative models such as LCFG[1, 2], PSGCONF[24], database-driven models[18, 19], and some flavors of Arusha[14, 15] selectively replace all configuration files while leaving all else alone.

A generative configuration tool decomposes configuration into generative actions $g \in G$ each of which produces and asserts the contents of an entire configuration. For any generative actions $g$ and $h$, $h \circ g = h$, i.e., the last action wins. This means that each element of a generative semigroup is a "left zero" of the semigroup, so that the structure is "left annihilating". These $h$'s are the $b_i$'s in the above discussion.

One disadvantage of generative configuration management is the expense of building the initial model, the "problem of semantic distance"[11]. Typical configuration procedures are documented for humans in terms of scripts, not in terms of assertions. Moving that configuration from the script to a generator means turning scripts into assertions about their effects, a problem that is as difficult as proving a script to be correct. The generator incorporates distributed knowledge from multiple scripts into an expression of global knowledge, consistency, and precedences.

### 8.4 Imperative Management

By contrast, managing a system by imperative methods is the most complex method algebraically. The actions in this method are imperative scripts, so that interactions between scripts can potentially be arbitrarily complex. Unfortunately, scripts are also necessary. As a legacy, most systems are initially built through scripts. In general, however, concatenating two scripts does not guarantee a composite effect of both scripts.

Scripts do have several advantages. They are the legacy way to accomplish change and are already part of tasks such as package installation. They work well in the absence of changes. They establish a baseline within which another tool can work, and that is difficult to establish otherwise.

## 9 Conclusions

We propose a new and more strict notion of convergence that rules out common practices previously considered to be convergent. Using this definition alone, the properties of a typical configuration management system emerge from the theory of semigroups. Using the prior definition, no such properties emerge, and the resulting inelegance of the system leads to usability problems in crafting strongly convergent actions from weakly convergent primitives.

In a simple algebraic sense, imperative, generative, and convergent methods form a hierarchy in which imperative methods are closest to the machine and convergent methods closest to becoming self-healing. During initial machine bootstrapping, imperative methods are inescapable; the cost of replacing them with generative or convergent methods is too high. After initial configuration, convergent methods have the potential to inadequately cover all required options, a behavior not exhibited by typical generative systems, The initial run of a convergent system must "configure all variables" and thus has a generative flavor.

So, we conclude from this study that an ideal configuration management tool actually has attributes of each of the strategies:

1. An imperative bootstrap.
2. A generative initial configuration.
3. A convergent ongoing management process.

where the latter is convergent in the strong sense (Definition 7) rather than the weak sense exhibited in most current convergent tools, including Cfengine.

We strive for order in a disorderly world. Strongly emergent properties are the result of strong axioms and an uncompromising adherence. The cost of this adherence is sometimes high. We feel that the value exceeds the cost, and that future self-healing tools will adopt strongly convergent methods to assure the algebraic elegance that naturally results in simplicity of use.

## Acknowledgements

This work is not the product of individuals, but of a coordinated inquiry among a large and diverse intellectual community. We are grateful to the Large-Scale System Configuration mailing list (lssconf) for providing sustained discussion of the strategies for configuration management and their impact. George Leger and David Krumme both provided detailed, in-depth comments and corrections to the mathematics. Special thanks to Mark Burgess, Paul Anderson, and Steve

Traugott, for providing the main fuel for this debate with their groundbreaking work. This work was supported in part by an equipment grant from Network Appliance Corporation.

## References

[1] P. Anderson, "Towards a High-Level Machine Configuration System" *Proc. LISA-VIII*, USENIX Assoc., 1994. 28, 37

[2] P. Anderson, P. Goldsack, and J. Patterson, "SmartFrog Meets LCFG: Autonomous Reconfiguration with Central Policy Control", to appear in *Proc. LISA-XVII*, USENIX Assoc., San Diego, CA, 2003. 28, 37

[3] M. Burgess, "A Site Configuration Engine", *Computing Systems* **8**, 1995. 28, 30, 35

[4] M. Burgess and R. Ralston, "Distributed Resource Administration Using Cfengine", *Software: practice and experience* **27**, 1997. 28, 35

[5] M. Burgess, "Computer Immunology", *Proc. LISA-XII*, Boston MA, USENIX Assoc., 1998. 28

[6] M. Burgess, "Theoretical System Administration", *Proc. LISA-XIV*, New Orleans LA, USENIX Assoc., 2000. 28

[7] Lionel Cons and Piotr Poznanski, "Pan: A High-Level Configuration Language", *Proc. LISA-XVI*, USENIX Assoc., Philadelphia, PA, 2002.

[8] A. Couch, "SLINK: simple, effective filesystem maintenance abstractions for community-based administration", *Proc. Lisa-X*, USENIX Assoc, 1996.

[9] A. Couch, "Chaos out of order: a simple, scalable file distribution facility for 'intentionally heterogeneous' networks", *Proc. LISA-XI*, USENIX Assoc., 1997.

[10] A. Couch and M. Gilfix, "It's elementary, dear Watson: applying logic programming to convergent system management processes", *Proc. Lisa-XIII*, USENIX Assoc., 1999. 30

[11] Alva L. Couch, "An expectant chat about script maturity", *Proc. LISA-XIV*, USENIX Assoc., 2000. 37

[12] Alva L. Couch and Noah Daniels, "The maelstrom: network service debugging via 'ineffective procedures' ", *Proc. LISA-XV*, USENIX Assoc., 2001. 28, 30, 34

[13] A. Couch, J. Hart, E. Greenlee, and D. Kallas, "Seeking Closure in an Open World: A Behavioral Agent Approach to Configuration Management", to appear in *Proc. LISA XVII*, USENIX Assoc., San Diego CA, 2003.

[14] Matt Holgate and Will Partain, "The Arusha Project: A framework for collaborative Unix system administration", *Proc. LISA XV*, USENIX Assoc., San Diego CA, 2001. 28, 37

[15] Matt Holgate, Will Partain, et al, "The Arusha Project Web Site", http://ark.sourceforge.net 28, 37

[16] L. Kanies, "Practical and Theoretical Experience with ISconf and Cfengine", to appear in *Proc. LISA XVII*, USENIX Assoc., San Diego CA, 2003. 28

[17] Frode Eika Sandnes, "Scheduling partially ordered events in a randomised framework - empirical results and implications for automatic configuration management", *Proc. LISA XV*, USENIX Assoc., San Diego CA, 2001. 28

[18] Jon Finke, "An improved approach for generating configuration files from a database", *Proc. LISA-XIV*, USENIX Assoc., 2000. 28, 37

[19] J. Finke, "Generating Configuration Files: The Director's Cut", to appear in *Proc. LISA-XVII*, USENIX Assoc., San Diego, CA, 2003. 28, 37

[20] P. A. Grillet, *Semigroups: An Introduction to the Structure Theory*, Marcel Dekker, Inc, New York, NY, 1995. 34
[21] J. M. Howie, *An Introduction to Semigroup Theory*, Academic Press, 1976. 34
[22] E. S. Ljapin, *Semigroups*, American Mathematical Society, Providence, RI, 1963. 34
[23] Mark Logan, Matthias Felleisen, and David Blank-Edelman, "Environmental Acquisition in Network Management" *Proc. LISA XVI*, USENIX Assoc., Philadelphia, PA, 2002.
[24] M. D. Roth, "Preventing Wheel Reinvention: The Psgconf System Configuration Framework", to appear in *Proc. LISA-XVII*, USENIX Assoc., San Diego, CA, 2003. 28, 37
[25] Steve Traugott and Joel Huddleston "Bootstrapping an Infrastructure", *Proc LISA XII*, USENIX Assoc., Boston, MA 1998. 28
[26] Steve Traugott and Lance Brown, "Why order matters: Turing equivalence in automated systems administration" *Proc. LISA XVI*, USENIX Assoc., Philadelphia, PA, 2002. 28

# An Epidemic Protocol for Managing Routing Tables in Very Large Peer-to-Peer Networks

Spyros Voulgaris and Maarten van Steen

Vrije Universiteit, Amsterdam
{spyros,steen}@cs.vu.nl

**Abstract.** Building self-maintained overlay networks for message routing has recently attracted significant research interest [5, 6, 7, 8, 9]. All suggested solutions have a common goal: To build and maintain structures (routing tables) that can be used to route messages. Several of the proposed algorithms focus on efficiency of bandwidth usage. However, their behavior is uncertain in the presence of highly dynamic environments, or serious disasters (i.e. half of the nodes crashing). In this paper we present an alternative approach to managing routing tables for peer-to-peer routing overlay networks, based on the Newscast epidemic protocol [1]. We substantiate our claims by presenting experimental results. We, therefore, demonstrate the potential of the Newscast epidemic protocol to create highly robust, self-administered overlay networks, able to sustain and adapt fast to severe network changes.

## 1 Introduction

The Internet has dramatically expanded over the past few years, proving the traditional client-server model of communication inadequate for a number of services in the large scale. The network research community has realized that using centralized servers is not the way to go with respect to managing and administering very large scale distributed systems, as well as for certain applications for such systems. As a result, considerable effort has been made in designing *peer-to-peer* (P2P) overlay networks. These networks are highly (or totally) decentralized distributed systems, where nodes are equal peers cooperating to provide a service all together. The major advantage of such systems is that they do not involve any central point of administration or control.

A significant part of the recent research in P2P systems has been in designing overlay networks for routing. These networks operate in the application layer, on top of an existing physically interconnected set of nodes (such as the Internet). They assign each participating node an ID, and route messages to a node based on that, rather than based on its IP address. Performance (in terms of routing hops) is usually inferior compared to traditional IP routing. However, they offer a number of other, attractive advantages, such as higher fault tolerance, flexibility of deployment, adaptivity, as well as lack of central control. A number of such P2P systems has been proposed, such as CAN [5], Chord [6], Pastry [7], and Tapestry [8]. Their common property is that they all try to form and maintain

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 41–54, 2003.

some sort of structure across the large number of participating nodes, that is then used to route packets among them.

Other P2P algorithms (such as Newscast [1]) fall in the category of *epidemic* (or *gossip*) protocols. They aim at exploiting randomness to disseminate information across a large set of nodes to keep that set of nodes highly connected even in the event of major disasters, without keeping any static structures or requiring any sort of administration. Connection between nodes in such systems is highly dynamic. These systems are more adaptive to major network changes, and appear to have a self-healing behavior with respect to major network disasters. Their lack of structure, however, restricts them from carrying out some types of services (i.e. routing) in an efficient way.

In this paper we combine the advantages of routing overlay networks with those of highly fault tolerant, self-healing epidemic networks. In particular, we investigate how to bootstrap and maintain structures used for peer-to-peer routing based on the highly dynamic emergent behavior of Newscast. Moreover, this paper demonstrates the power of an epidemic protocol as simple as the Newscast protocol, in managing structures across very large-scale distributed systems, in a totally distributed and scalable way, with no need for external administration, and with very high fault tolerance.

Section 2 provides a brief description of Newscast, concentrating more on its epidemic protocol. Section 3 describes structures that can be used for peer-to-peer routing. The architecture proposed for management of peer-to-peer routing tables is presented in section 4. Section 5 describes the experiments we conducted, and section 6 discusses the results obtained. Finally, we present conclusions and directions for future research.

## 2 The Newscast Protocol

Newscast (introduced in [1]) is a model for information dissemination and membership management in large-scale, agent-based distributed systems. It deploys a simple, peer-to-peer data exchange protocol. The *Newscast protocol* forms an overlay network and keeps it connected by means of an epidemic algorithm. The protocol is extremely simple: each agent knows only a (continuously changing) small set of peers, and periodically picks randomly one of them to exchange information with. In the following, we present a brief overview of the protocol's operation, and explore some properties of its emergent behavior.

In Newscast information is exchanged by means of *news items*. A news item is a 4-field structure containing (a) the ID of the agent where it originated, (b) the network address of that agent, (c) a timestamp of the moment it was generated, and (d) some application-specific data. Each agent maintains a *fixed-sized* cache of $c$ news items (with typical value 20 to 40). The basic idea is that each agent periodically picks a random peer from its cache and subsequently both agents replace their cache entries with the $c$ freshest news items of the union of their original caches.

More formally, but omitting specific details described in [1], each agent executes the following four steps once every $\Delta T$ time units ($\Delta T$ is referred to as the *refresh interval*)

1. Add a fresh (agent-specific) news item to the cache.
2. Randomly select a peer agent by considering the network addresses of other agents as found in the cache.
3. Send all cache entries to the selected peer agent, and, in turn, receive all that peer's cache entries.
4. Out of the (up to) $2c$ cache entries, keep the $c$ newest ones, and discard the rest.

The selected peer from step 2 executes the last two steps as well, so that after the exchange both agents have the same cache. Note that as soon as any of these two agents executes the protocol again, their respective caches will most likely be different again.

This algorithm resembles the traditional push-pull epidemic protocol [2]. A critical difference, however, is that no correspondent knows the complete member list, but only a small, random fraction of it.

The protocol does not require that the clocks of the agents are synchronized, but only that the timestamps of news items in a single cache are mutually consistent. We assume that the communication time between two agents is negligible compared to $\Delta T$ (which is generally in the order of minutes). When an agent $A$ passes its cache to $B$, it also sends along its current local time, $T_A$. When $B$ receives the cache entries, it subsequently adjusts the timestamp of each entry with a value $T_A - T_B$, effectively normalizing the time of each new entry to those already cached.

As it turns out, this simple model of communication has desirable statistical properties. To understand the behavior of newscasting, we consider the undirected communication graphs $G_t$ at different time instants $t$. Each such graph is constructed as follows. The vertex set $V_t$ of $G_t$ contains the agents that are alive at time $t$. A link between agents $a, b \in V_t$ exists if and only if either $a$ is in the cache of $b$ or $b$ is in the cache of $a$ at that time. The cache-exchange algorithm leads to a series of graphs $G_t$, given an initial graph $G_0$. Graph $G_t$ expresses the possibility of cache exchanges, and in essence information flow, at time $t$.

Now consider the series of graphs $G_0, G_{\Delta T}, G_{2\Delta T}, \ldots$. Note that during a time interval $\Delta T$ each agent initiates the cache-exchange algorithm. In other words, after $\Delta T$ time units, all agents will have added a fresh news item to their caches, and will have exchanged and merged caches with at least one of their neighbors (and possibly more). We say that a *cycle* of the Newscast protocol has completed.

We have conducted simulations with up to 50,000 agents [1] assuming an idealized communication infrastructure with no communication delays and packet losses, and emulations by deploying up to 128,000 actual Newscast agents on a real wide-area network [4]. Both our simulations and emulations show that even for small cache sizes (say, $c = 20$), each graph $G_{k\Delta T}$ stays connected. Moreover, it turns out that the *average path length* (average length of shortest

paths between any two nodes) converges to a very low value in just a few cycles, and which is only slightly longer than the average path length in random graphs. For real experiments with 128,000 nodes, and cache size of $c$=20, 30, and 40 entries, the average path length converges to 6, 5, and 4, respectively within the first 30 cycles. Additional experiments showed insignificant dependence on network latencies and packet losses, except when these were exceptionally high.

A more significant property of Newscast is, however, its strong connectivity. Let $G'_t$ be a subgraph of $G_t$, where a number of random nodes (and their links) have been removed. Our simulations and emulations show that $G'_t$ remains connected even when more than half of the nodes are removed. This means that when even half of the agents of a Newscast network are removed, the rest of the nodes remain connected in a single cluster. In fact, Newscast's connectivity property is so strong that one needs to remove over 75% of the nodes to start breaking up the remaining network into disjoint clusters. The nodes surviving such a major disaster, quickly converge to an independent strongly connected Newscast network, capable of sustaining further major disasters of similar severity.

Our experiments also show that we need only an extremely simple way of handling membership, which is an important improvement in comparison to other epidemic models, such as [3]. Consider the worst solution to handling membership that could possibly disrupt the emergent behavior of our protocol: an agent contacts the agent running on a well-known central server and simply initiates the cache-exchange protocol with it. This approach systematically biases the content of caches, which now all depend on what is stored at the central server.

We conducted a simulation experiment in which we admitted 50 new agents at every communication cycle until 5,000 agents had joined the network, after which no new agents were allowed to join. By measuring the average path length again, we saw that shortly (i.e. approximately 15 cycles) after the last agents had been added, the average path length quickly converged to the one we would expect in a stable graph. We can conclude that even this worst-imaginable membership protocol does not affect the general properties of newscasting. In effect, when a node wants to join, it needs to know only the address of a single other node and can simply start executing the newscast protocol. Leaving is done by simply stopping communication.

## 3 Peer-to-Peer Routing

One of the key issues in designing large-scale peer-to-peer overlay networks is to provide an efficient way to do routing. Several architectures have been proposed as peer-to-peer routing substrates, such as CAN [5], Chord [6], Pastry [7], and Tapestry [8]. Such distributed systems that map "keys" onto "values" in a way similar to hash tables, are referred to as *distributed hash table* (*DHT*) based networks [9]. Two of the most popular of them, Pastry and Tapestry, employ routing based on the same concept: incrementally matching the destination's

ID, digit by digit. In this section we present the structure and operation of the principal structures used for routing, the *routing tables*.

Each node is assigned a unique numeric identifier, its *nodeId*, or simply *ID*. When presented with a message and a numeric *key*, a node routes the message towards the node whose ID is equal to the given key. NodeIds and keys are N-bit integers, forming a nodeId space that spans from 0 to $2^N - 1$. $N$ has a typical value of at least 64 to provide a sufficiently large nodeId space to accommodate possibly billions of nodes. Nodes pick their nodeIds randomly with uniform probability from the set of N-bit strings. We assume that the nodeId space is large enough compared to the actual number of nodes, such that the probability that nodes pick *unique* IDs is high. It is, therefore, assumed that nodeIds are uniformly distributed across all geographic regions, multiple jurisdictions, and various networks.

For the purpose of routing, nodeIds and keys can be thought of as a sequence of digits in base $2^b$ ($b$-bit long digits), where $b$ is a configuration parameter with typical value 4 (which implies *hexadecimal* digits). Routing a message to its destination is achieved gradually, by matching one additional digit of the message's key at a time, say, from left to right. That is, in each step the message is normally forwarded to a node whose ID shares with the key a prefix at least one digit ($b$ bits) longer than the prefix the key shares with the present node's ID, if such a node is known. If such a node is not known, routing of that message fails.

To implement the logic described above in message routing, each node maintains its *routing table*. The routing table of a node consists of $N/b$ rows of $2^b$ entries each. That is, the number of routing table rows grows logarithmically with the size of the ID space supported. A routing table entry contains the ID of a node, and its corresponding IP address. A given row of the routing table contains $2^b$ entries, and represents a matching prefix in the nodeId up to a digit position. Entries in the $r$th row ($r \in \{1, \ldots, N/b\}$) contain nodes whose IDs share the same $(r-1)$-digit prefix with the present node. The $c$th entry of the $r$th row contains such a node, with the additional constraint that its ID's $r$th digit is equal to $c$. For instance, assuming b=4 (hexadecimal digits for the nodeId), the 2nd entry of the 3rd row of the routing table for node 437BF52... ($N/4$ hex digits in total) is some node whose ID starts with 43**2**, while the 8th entry of its 5th row has a node whose ID starts with 437B**8**.

Upon receiving a message, a node compares the message's key to its nodeId. If they share a common prefix of $i$ digits, it should forward it to a node whose ID shares a prefix of $i+1$ digits with the key. To accomplish that, the present node looks up the $(i+1)$th row of its routing table, which contains nodes sharing with the key the same $i$ first digits. Out of that row, it picks the $k$th entry, where $k$ is the value of the key's $(i+1)$th digit, and forwards the message to that node. That node not only shares with the key the same first $i$ digits, but also the $(i+1)$th one. This process continues either until the node whose ID matches all digits of the message's key is reached, or, else, until the message cannot be forwarded any further.

## 4 P2P Routing Based on Newscast

An important issue in DHT-based peer-to-peer systems is managing the routing tables. These tables are kept up-to-date by having nodes that join or leave the system contact other nodes explicitly. To handle failures, heartbeat algorithms are used to probe nodes and to take measures when a failure is detected.

We propose a different approach, namely to separate routing from table management, similar to the separation deployed in Internet routing protocols such as OSPF or RIP. We believe such a separation often leads to a cleaner and simpler design, although sometimes at the cost of performance.

Newscast can typically be used as a distributed background process by which nodes are kept up-to-date in a lazy fashion. For DHT-based peer-to-peer systems, we propose to deploy Newscast for maintaining routing tables. Our method is completely decentralized, highly robust, and quickly adjusts itself to major changes in the network. These advantages come at the price of continuous bandwidth consumption.

### 4.1 The Principal Idea

Newscast's epidemic protocol has a number of important properties, as described in section 2. It maintains a strongly connected graph, it sustains disasters, it adapts very fast to (possibly major) network changes, and it is highly scalable. The idea is to combine the adaptivity strength of the Newscast epidemic protocol with the efficiency of the routing scheme presented in section 3, to create a robust, highly fault resilient, peer-to-peer overlay network for efficient routing.

Knowledge of peer nodes provided by Newscast can be used to populate the routing tables. In each iteration of the Newscast protocol every node receives references to $c$ other nodes, randomly chosen among *all* the participating nodes. Each node has to gather enough information to build and maintain all its $N/b$ rows.

Let us concentrate first on building the first row of a node's routing table. This requires references to nodes whose IDs differ from the present node's ID in the first digit, which makes a total of $2^b - 1$ nodes. Seen differently, considering $2^b$ *classes* of nodes split according to their ID's first digit, we require a reference to an arbitrary representative from each class (excluding the present node's class). Assuming evenly distributed node IDs, each class contains roughly $1/2^b$ of the nodes. Therefore, with very high probability, a node will have learned about at least one representative from each of the $2^b - 1$ classes when $2^b$ (or a few more) random nodes become known to it. Assuming $2^b = 16$ (for $b = 4$) and cache size $c = 20$, this might happen even when a node executes the Newscast cache-exchange protocol only once.

For the second row of a node's routing table, we require references to $2^b - 1$ nodes of IDs with the same first digit, but different second digit than the present node's ID. Apparently, we are seeking for representatives of much narrower node ID ranges, each containing roughly $(1/2^b)^2$ of the total nodes. In general, filling up the $k$th row requires representatives of $2^b - 1$ classes, each containing just

$(1/2^b)^k$ of the total nodes. The narrower a node ID range gets, the more difficult it becomes to find a representative from it by taking random samples across all the nodes. Obviously it would be inefficient to rely on Newscast over the whole set of nodes to find representatives of these narrow node ID ranges. A more focused approach is required, described in the following subsection.

### 4.2 Multi-layer Newscast Scheme

As we mentioned, executing the Newscast protocol can be seen as running a distributed background process by which nodes are kept up-to-date. To efficiently maintain routing tables, we run multiple instances of the Newscast protocol, each node running several Newscast agents in parallel. In fact, each node runs exactly $N/b$ Newscast agents, each one being responsible for maintaining one of the rows of the node's routing table. The agent responsible for row $r \in \{1, \ldots, N/b\}$ of node $X$ will be referred to as *agent #r of node X*.

Note that a node's agent $\#i$ deals only with nodes whose IDs share a common prefix of length $i-1$ with the present node's ID, that is, it does not accept any nodes with a different prefix in its cache. Moreover, a node's agent $\#i$ interacts, in terms of the Newscast protocol, only with agent $\#i$ of peer nodes, as shown in figure 1. Apparently, agent $\#i$ of a peer node contacted by agent $\#i$ of the present node, contains items whose node IDs share the same $i-1$ long prefix too. What we are thus seeing, is that agent $\#i$ of node $X$ maintains a small-world network with *some* other nodes whose IDs are the same in the first $i-1$ digits as the ID of $X$. Agent #1 of all nodes maintain a single connected small world of the whole set of nodes.

To collect *all* nodes with a particular prefix in a *single* connected small world, we apply the following strategy. Peers that become known to a node's agent $\#i$ are *also* reported to the same node's agent $\#(i+1)$. That agent, in turn, inserts the peers that match its prefix requirement in its cache (by replacing the oldest

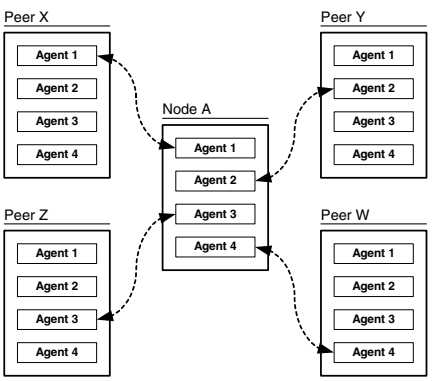



**Fig. 1.** Communication of node $A$ during one communication cycle

cache items), and further reports them to agent $\#(i+2)$ of the same node, if any. In other words, any peer that becomes known to a node's agent $\#i$, is also made known to all the agents $\#j$ $(j > i)$ of the same node that are potentially interested.

An important observation is that once agents $\#i$ of *all* nodes that share the same first $i-1$ ID digits have formed a single small world, agents $\#(i+1)$ of the nodes among them that also share an arbitrary same $i$th digit form a single connected small world very fast. Each agent $\#i$ learns about $c$ random peers with the same first $i-1$ digits every $\Delta T$ time units. Assuming evenly distributed node IDs, we expect that on average $c/2^b$ of the peers that become known every $\Delta T$ time units share the $i$th ID digit too with the present node, in addition to the first $i-1$ digits. Given typical parameter values of $c = 20$ and $b = 2$, one or more peers sharing $i$ digits become known every $\Delta T$ time units on average. This partly explains why all agents of every node form small worlds quickly, as we shall see later.

Notice that, initially, every node's agent $\#(i+1)$ forms its own (trivial) small world, disjoint from all the rest. Such a small world generally expands on each cycle of agent $\#i$, since a random peer satisfying the prefix requirement of agent $\#(i+1)$ is introduced. Moreover, two small worlds of $n$ and $m$ nodes unite if any of the $n$ nodes of the first happens to learn about the existence of any of the $m$ nodes of the other, respectively. Therefore, the larger disjoint small worlds become, the more likely they will unite. What we are seeing, is an increasingly accelerating behavior in the process of merging among disjoint small worlds. It is therefore reasonable to state that agent $\#(i+1)$ of a set of nodes sharing the same first $\#i$ digits form a small world in just a few cycles, provided agent $\#i$ of nodes sharing the same first $\#(i-1)$ digits form a small world too.

The set of all nodes' agent #1 run a pure Newscast instance that guarantees a single connected small world of *all* existing nodes. By induction, and based on the claims of the previous paragraph, we expect all instances of Newscast executed by all agents of all nodes, to quickly form the small worlds they are designed for.

## 5 Experimental Setting

We implemented the architecture described in section 4.2 in Java and deployed it on the DAS-2, a 400-processor cluster geographically distributed over a wide-area network across the Netherlands. We carried out experiments with a set of 65,536 nodes, a number of them running on each DAS-2 processor simultaneously.

Regarding the parameters related to peer-to-peer routing, we considered node IDs of length $N = 16$ bits, and digits of length $b = 4$ bits (hexadecimal digits). This setting resulted in $N/b = 4$ rows and $2^b = 16$ columns per routing table.

As far as the Newscast parameters are concerned, each node was running 4 Newscast agents, one for each of its 4 routing table rows. A cache size of $c = 20$ was used for each Newscast agent. We ran our experiments with the same refresh interval of $\Delta T = 10sec$ for all agents. That is, every 10 seconds *each* of the

4 Newscast agents of each node initiated a cache exchange. We recorded and analyzed the behavior of our architecture at intervals of 60 seconds, that is, we logged the whole network's state every 6 communication cycles.

Another facet of our experiments that is worth noting is the *bootstrapping* mechanism. By bootstrapping we refer to the procedure of providing agents with the information required to jump-start the overlay network's formation. In principle, a new agent joins by contacting *any* existing agent and exchanging caches. When the whole network starts from scratch, a systematic way has to be present to provide one or more initial communication points to each agent. In our experiments, all nodes' agents #1 were provided with the address of one single-selected node's agent #1. Providing agents with a choice of (possibly random) agents to connect to initially, enhances the randomness of the network from the early cycles. However, a bootstrapping mechanism as simple and centralized as the one we chose further endorses our claims of our architecture's fast convergent behavior, as discussed in the following section.

Finally, we imposed a fake large-scale failure while the experiment was running, to observe and analyze the behavior of our system in such cases. In particular, we killed 50% of the nodes in the middle of the experiment. Our observations of the experiments and their analysis are presented in the following section.

## 6 Experimental Results and Analysis

This section presents the output of our experiments with 65,536 agents. We recorded and analyzed two aspects of the system's behavior: dynamic forming of the routing tables when bootstrapping, and following a large-scale failure.

### 6.1 Bootstrapping

The first part of our experiment aimed at observing the system's behavior while bootstrapping. Figure 2 presents the system's fast convergence to a fully operative routing substrate. It shows the average number of routing table rows that are completely filled per node, as a function of the number of cycles elapsed from the experiment's start. A node's $i$th routing table row being completely filled means that the node can route *any* message whose key shares $i - 1$ digits with the node's ID to a peer node whose ID additionally matches the $i$th digit of the message's key. Note that the system manages to fill *all* routing table entries in *all* nodes in less than 30 cycles.

Figure 3 demonstrates the efficiency in routing messages to random destinations. From each node we routed a number of messages to random nodes. Figure 3 presents the average values. The left-hand diagram shows how many routing steps messages took on average en route to their destination. Initially, routing tables are empty, so messages cannot take any steps towards their destinations. However, as routing tables are gradually formed, messages are correspondingly routed through more steps. This diagram is similar to the one of

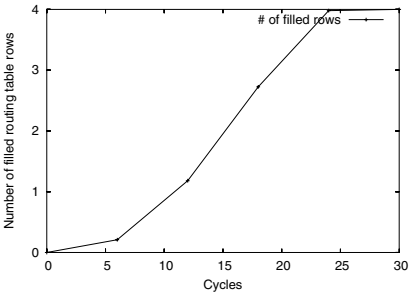



**Fig. 2.** Average number of filled routing table rows

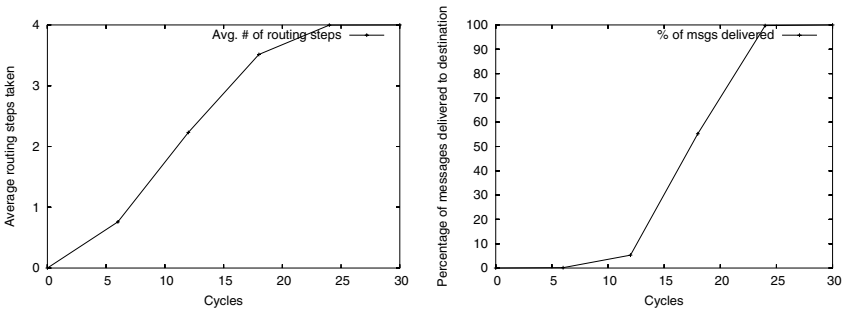



**Fig. 3.** Left: Average routing steps taken. Right: Percentage of messages delivered

figure 2, as the number of routing steps a message takes is directly dependent on the number of filled routing table rows.

The right-hand diagram of figure 3 shows what percentage of the messages manage to actually reach their destinations, as a function of the number of cycles. For the first 10 cycles few or none of the messages reach their destinations. However, as routing tables are filled, more messages are routed all the way through to their destinations. As it turns out, after the first 24 cycles 99.74% of the messages are delivered to their destinations, and after 30 cycles, this fraction increased to 99.998%.

### 6.2 Robustness to Large-Scale Failures

To test the system's behavior in the face of large-scale failures, we intentionally killed *half* of the agents after we knew that all nodes' routing tables had been completely filled, at cycle $d$.[1] More specifically, we killed all nodes with an odd

[1] This corresponded to approximately 10 minutes after the experiment's start

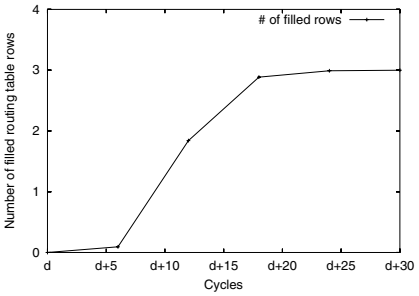



**Fig. 4.** Average number of filled routing table rows when recovering from a 50% node crash that happened at cycle $d$

ID. As we shall see, the network remains connected after such a major disaster, and adapts very quickly to the set of nodes that remain alive.

Figures 4 and 5 are analogous to the previous figures, 2 and 3. Figure 4 shows the average number of routing table rows that are completely filled (with valid entries), per node. Note that outdated entries of crashed nodes (the ones with odd IDs) are not considered valid, and therefore are not counted. Immediately after the crash none of the nodes' rows are filled, which implies that all nodes' routing rows also had some entries with odd node IDs. However, as can be seen in the diagram, routing tables are filled very quickly. Within 30 cycles from the crash all nodes' first 3 routing table rows have been filled. Note that this is the maximum number of rows that can be filled per node. Routing tables' 4th rows cannot be filled, as they would require nodes that match all possible cases for the last digit of their IDs. Since nodes with odd IDs do not exist any more, it is not possible to fill up these rows. This, however, does not affect routing, as routing paths to all existing nodes (i.e. nodes with an even ID) do exist and are complete.

The system's capability to route messages can be seen in figure 5. The left-hand diagram shows the average number of steps a message is routed through. Initially, since half of the nodes have been removed, messages are routed on average half-way through to their destination. As routing tables adjust to the change imposed by half the nodes crashing, messages are routed through more steps to their destinations. The right-hand diagram of figure 5 shows the percentage of messages that are successfully routed all the way through to their destination. Just like in the bootstrapping case, routing tables are formed very fast. It takes less than 20 cycles from the moment of the crash to form routing tables that can route any message from *any* source to *any* destination.

### 6.3 Bandwidth Considerations

In this section we provide an estimation of the individual (per node) and aggregate bandwidth used in our experiments, based on the number of bytes trans-

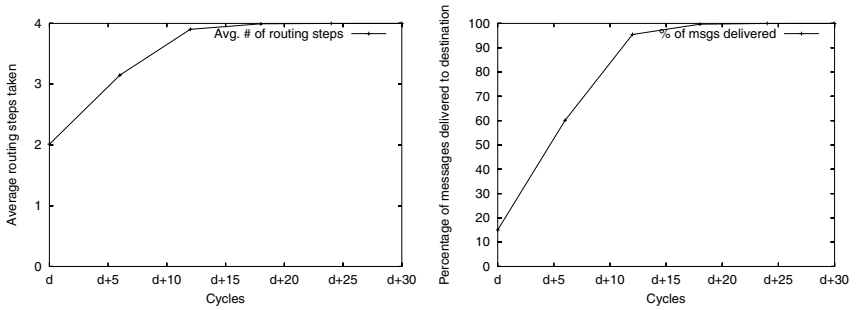



**Fig. 5.** Message routing while recovering from a 50% node crash that happened at cycle $d$. Left: Average routing steps taken. Right: Percentage of messages delivered

ferred by the application layer. Note that some additional overhead is induced by the underlying network protocols (i.e. TCP/IP), which we do not consider here. Despite the 16-bit node IDs we used in our experiments, we make the estimation assuming node IDs of 64 bits, which would be the ID size in real operation.

A cache entry consists of 16 bytes: 8 bytes for the node's 16-bit ID, 4 bytes for its ip address, 2 bytes for the port, and 2 bytes for the entry's timestamp. One cache has $c = 20$ entries, which account for 320 bytes. A cache exchange involves sending the cache to a peer *and* receiving the peer's cache, therefore causes traffic of 640 bytes. Every $\Delta T$, each node initiates exactly one cache exchange, and also participates on average in one cache exchange initiated elsewhere. Therefore, two cache exchanges cause transfer of 1280 bytes. For running 4 agents, a single node exchanges $4 \times 1280 = 5120$ bytes every $\Delta T = 10sec$. That is, 512 bytes per second, or 4096bps (4Kbps). This is the price to pay for achieving fully operative routing tables in less than $30 \times 10 = 300$ seconds, which is 5 minutes.

For the aggregate bandwidth we multiply the individual node bandwidth by the number of nodes and divide by two, since the traffic caused by each cache exchange has been counted twice, once for the exchange initiator and once for the peer node. Therefore, we have a total bandwidth of $65,536 \times 4/2 = 131,072$Kbps, which is 128Mbps. Note that even though this bandwidth seems too high, it is in fact distributed across the whole (possibly world-wide) network.

In a real system, with 64-bit node IDs, and a digit length of 4 bits, we would need 16 Newscast agents running per node. This would require the exchange of $16 \times 1280 = 20,480$ bytes every $\Delta T$ per node. Note that the refresh interval, $\Delta T$, is a configuration parameter. By setting a longer refresh interval, we can lower the bandwidth used by each node, at the expense of slower completion of the routing tables. For instance, a refresh interval of $\Delta T = 60sec$ would require a bandwidth of $20,480/60 \simeq 341$ bytes per second, or roughly 2.7Kbps. However, in that case routing tables would take longer to be filled, around 30 minutes.

## 7 Conclusions and Future Directions

This paper aimed at demonstrating the potential of the Newscast protocol in building large-scale, self-managing communities. In particular, we dealt with the application of managing routing tables for DHT-based peer-to-peer networks. We introduced a Newscast-based architecture for this application, and analyzed the system's behavior through experimentation. We showed that the proposed system forms routing tables fast, in a totally decentralized, self-organized manner.

This research is very recent, and currently under development. The results of the experiments suggest that our system can provide highly robust, non-centralized routing table management. However, more research remains to be done to discover potential optimizations for our architecture, such as in the field of bandwidth consumption. Our architecture could possibly use significantly less bandwidth if adaptive refresh intervals were applied. Also, each agent of a node could have an individually optimized set of configuration parameters, such as cache size and refresh interval. Future research aims at optimizing the current approach.

Another goal for future research, in a broader sense, is exploiting Newscast for a multitude of diverse peer-to-peer applications. We envision Newscast as being a basic background process, supporting, organizing, and managing overlay networks in a fully decentralized way.

The contribution of this paper is that it provides and analyzes a complete solution to a specific problem, showing the potential of the Newscast protocol to support such systems.

## References

[1] M. Jelasity, M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, Oct. 2002. 41, 42, 43

[2] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proc. 6th ACM Symp. Principles of Distributed Computing (PODC'87)*, pp. 1–12, Vancouver, Aug. 1987. 43

[3] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers* 52(2):139–149, 2003. 44

[4] S. Voulgaris, M. Jelasity, M. van Steen. A Robust and Scalable Peer-to-Peer Gossiping Protocol. In *Agents and Peer-to-Peer Computing* workshop, Melbourne, Australia, July 2003. 43

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM'01*, San Diego, CA, Aug. 2001. 41, 44

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM'01*, San Diego, CA, Aug. 2001. 41, 44

[7] A. Rowstron, P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems In *Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, Nov. 2001. 41, 44

[8] B. Zhao, J. Kubiatowicz, A. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, CA, Apr. 2001. 41, 44

[9] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica Looking up data in P2P systems In *Comm. ACM*, 46(2):43–48, 2003. 41, 44

# Towards Peer-to-Peer Traffic Analysis Using Flows*

Myung-Sup Kim, Hun-Jeong Kang, and James W. Hong

Department of Computer Science and Engineering
POSTECH, Korea
{mount,bluewind,jwkhong}@postech.ac.kr

**Abstract.** One of the main problems with today's Internet traffic analysis is caused by the large number of network-based applications whose types and traffic patterns are more complicated than in the past. Today, peer-to-peer (P2P), streaming media, and game traffic are continuously increasing. The difficulty the traffic analysis is that this newly emerging traffic is not as simple as past well-known port based traffic. This paper focuses on analyzing P2P traffic, which is the most complicated traffic among newly emerging Internet traffic. We describe the properties of P2P traffic and explain why P2P traffic analysis is more difficult than other types of Internet traffic analysis. Next, we propose a new algorithm suitable for P2P traffic analysis. The main idea of our algorithm is that flow grouping based on their relationships will increase the accuracy of P2P traffic analysis.

## 1 Introduction

There are two main problems in traffic monitoring and analysis of today's Internet traffic compared to the past network environment. The first is how to capture and handle the huge amount of traffic data in a real-time manner generated from high-speed network links, such as 2.5 Gbps and higher. The second is how to analyze various and complex types of traffic generated by many different types of network-based applications, such as streaming media, P2P, and game applications.

On the first problem, there have been many efforts and good research results reported. To increase the performance of capturing, network cards specialized for monitoring purpose (for example, DAG card [1]) were developed. Flow formats such as NetFlow [2] and sFlow [3] were created. Many routers now have a function to export flow information with these popular formats. For real-time traffic monitoring architecture, RTFM [4] was introduced and influenced the development of many traffic monitoring systems [5, 6, 7]. We have also developed a real-time traffic monitor-

* This work was in part supported by the Electrical and Computer Engineering Division at POSTECH under the BK21 program of Ministry of Education and HY-SDR Research Center at Hanyang University under the ITRC program of Ministry of Information and Communication, Korea.

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 55-67, 2003.

ing and analysis system called NG-MON [8], where a clustering and pipelining architecture has been applied for enhanced scalability.

The second problem, which this paper addresses, is caused by the high number of network-based applications. So the types and patterns of current network traffic are not simple as in the past. In the past network environment, HTTP, FTP, TELNET, SMTP and NNTP traffic occupied almost all Internet traffic. Today, the proportion of these well-known ports based traffic is decreasing. Instead P2P, streaming media and game traffic are increasing. The difficulty with traffic analysis is that this newly emerging traffic is not as straightforward as the well-known port based traffic. Therefore, we need a new algorithm to analyze these new types of Internet traffic.

This paper focuses on P2P traffic that is the most complicated traffic among newly emerging Internet traffic. A few years ago new types of network-based applications emerged, which are different from traditional client/server based application architecture, such as Morpheus [10], Gnutella [11], Soribada [9], etc. Soribada is a Korean version of Napster. P2P applications have changed Internet traffic patterns and directions in many ways. These new types of applications require a new method to analyze them. In this paper, for the analysis of P2P traffic we describe the properties of P2P traffic and explain why P2P traffic is more difficult than other types of Internet traffic. Next, we propose a new algorithm suitable for P2P traffic analysis. The main idea of our algorithm is that flow grouping according to the relationship among flows will increase the accuracy of P2P traffic analysis.

This algorithm is composed of four crucial steps. The first step is to make the Application Port Table (APT) by off-line exhaustive search of existing P2P applications. The second step is the Important Port Number selecting method from each flow record, which is important in the decision of P2P applications. The third step is the use of Flow Relationship information in the analysis phase. The last step is the decision of P2P application name for each flow. For the validation of proposed algorithm we designed and implemented a P2P traffic analysis system and present the result of P2P traffic analysis in the Internet junction of our campus network.

The organization of this paper is as follows. The properties of P2P applications and some other new traffic analysis efforts are described in Section 2. Section 3 describes our proposed P2P traffic analysis algorithm. The design and implementation issues are described in Section 4. In Section 5, we describe the analysis results using the proposed algorithm. Finally, concluding remarks are given and possible future work is mentioned in Section 6.

## 2 Related Work

In this section, we give a definition of P2P traffic and describe its properties. We also describe the existing traffic analysis mechanisms.

### 2.1 Definition of P2P Traffic

For the analysis of P2P traffic, the first step is to define the nature of P2P traffic. In this paper we define P2P traffic as traffic generated by P2P applications. Then what is

a P2P application? Figure 1 describes the critical difference between traditional client/server applications and newly emerging P2P applications.

As seen in Figure 1, in the client/server architecture we can explicitly divide all hosts into two groups: a server group and a client group. The client usually sends requests for some services and the server replies for each request. Direct communication among clients never occurs. However, in the P2P architecture each host can act as a server and a client simultaneously. In other words, direct communication between peers is possible. A host sends a request to other peers to obtain a certain service, and the same host simultaneously receives requests from other peers.

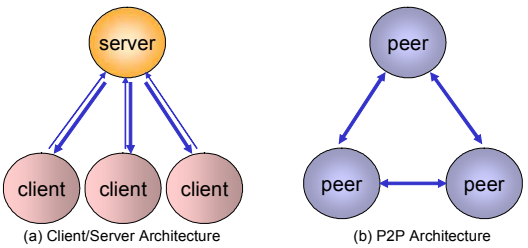



**Fig. 1.** Client/Server Architecture and P2P Architecture

The traffic pattern is also different from that of the client/server architecture. The traffic pattern in the client/server model is directional and downstream from server to client. Otherwise, the P2P traffic is bidirectional. With these properties of P2P applications, P2P traffic can be defined.

## 2.2 Categorization and Properties of P2P Applications

We can categorize P2P applications into two types; they are instant messaging applications, such as MSN Messenger [12] and Yahoo Messenger [13], and file sharing applications such as Morpheus [10], Soribada [9] and eDonkey [14].

The major functions of the instant messaging applications are message delivery, single or multi-user chatting and file transfer. The major functions of file sharing applications are searching and file transfer. Besides these major functions many additional features are provided to differentiate themselves from other applications. There are many P2P applications and their number will continuously increase in the future. Figure 2 describes a detailed communication sequence of two P2P applications: MSN Messenger [12] and Soribada [9], a Korean version of Napster. As Figure 2 shows, almost all P2P applications create multiple connections according to their different functions. Some applications use TCP and UDP simultaneously. These multiple connections make P2P traffic analysis very challenging.

We can summarize the properties of P2P applications from the traffic analysis point of view. First, there are many P2P applications. In Korea, the number of frequently used P2P applications are more than 20. The worldwide number of P2P applications is much greater than this number. Second, many P2P applications use multiple connections to support various functions. Some P2P applications use TCP and UDP simultaneously. Third, the protocol format or operation used in most P2P applications is

unknown. Fourth, the port numbers used by P2P applications are dynamically generated and many of them are not registered at IANA; sometimes they use port numbers which are already registered at IANA for some other purposes.

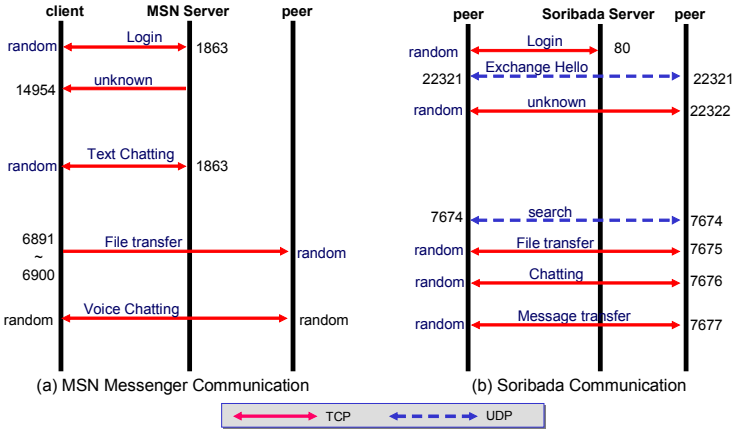



**Fig. 2.** Communication Detail of P2P applications

## 2.3 Related Work on P2P Traffic Analysis

The analysis of P2P traffic is one of the important issues in the current Internet environment. Until now a few researches on the architecture of P2P applications [15], the traffic patterns and properties of some specific P2P traffic [16] have been performed. However, the method to identify P2P traffic among all Internet traffic and decide the application name of certain traffic is still very primitive. The only method currently used is the traditional method that decides the P2P application name by the port numbers. Figure 3 (a) illustrates the traditional traffic analysis method. The architecture of most traditional Internet applications is a client/server architecture. The traffic analysis of these client/server applications is very simple. According to the port number less than 1024 from the packet header information, we can decide the application name generating that packet.

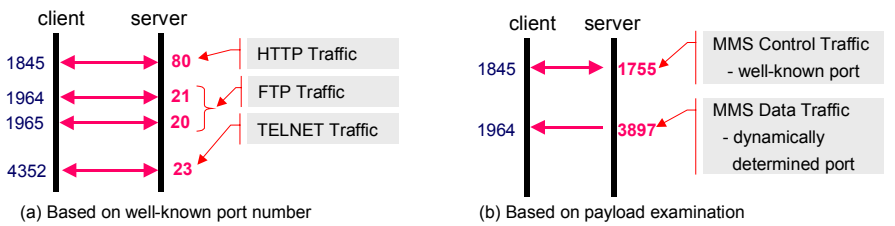



**Fig. 3.** Existing Traffic Analysis Methods

However, streaming media traffic is not as simple as traditional client/server traffic. Streaming media applications also use client/server architecture but they usually es-

tablish two connections to communicate between hosts: one for control data transfer and the other for video/audio data transfer. The port number used in a control session is a well-known fixed port. But the port number used for the data session is decided by the negotiation between the client and server. Mmdump[19] and SM-MON[20] introduced a payload examination based analysis for streaming media traffic analysis. Figure 3 (b) illustrates the method used in mmdump and SM-MON for MMS [17] traffic. They examine the payload of each control packet and detect the port number used in the data session. This method makes some overhead in the packet capture and analysis phase, because the entire packet should be captured and some processing is required in the examination of payload. This method is possible because there are few streaming media applications and protocols used worldwide, such as MMS [17] protocol and the RTSP [18] protocol. Moreover, the format of the RTSP protocol and its operation is open to the public.

Traditional well-known port based traffic analysis cannot be used in P2P traffic analysis because the port number used in the P2P application is usually over 1024 and they are using multiple connections. Further, for many P2P applications, port numbers are dynamically determined during the communication setup between the peers involved. The method of payload examination is not suitable because the number of P2P applications is large and usually the packet format and operation is not open to public. So we need a whole new method to analyze these multiple session and proprietary protocol based P2P application traffic.

## 3 P2P Traffic Analysis Algorithm

In this section, we present a new algorithm for P2P traffic analysis, which solves the problems that occur in the traditional well-known port number based analysis and payload examination based analysis. The main idea of the proposed algorithm is that flow grouping according to its corresponding applications will increase the accuracy of P2P traffic analysis. For example, the Web traffic typically uses port number 80 or 8080 for HTTP and 443 for HTTPS. The groups of flows generated by the Web server and client are obvious; the flows with port number 80, 8080, and 443 in the source or destination port can be grouped as Web traffic. In the case of P2P traffic, port number detection is more complex than Web traffic because P2P traffic applications are using port numbers over 1024 and the port number is often dynamically generated. If all P2P traffic can be selected among the entire range of traffic and then grouped according to its application name, then P2P traffic analysis will be performed with high accuracy.

For this purpose our proposed algorithm consists of four main processes, as illustrated in Figure 4. These processes are the Application Port Table (APT), the Important Port Selection, the Flow Relationship Map (FRM), and the P2P Application Decision. In our proposed algorithm we do not examine the payload of each packet; instead, we use only the header information of each packet.

The first step of the proposed algorithm is to construct the Application Port Table (APT). APT is constructed by the off-line exhaustive search of each P2P application using packet analysis tools. APT contains the P2P application names, their frequently

used port numbers and protocol numbers. This information is used in the decision of P2P application name of each flow in the P2P Application Decision process. The second step is the Import Port Number Selection process. In this step, the flow information is generated from the captured packets according to their 5-tuple information: source IP address, destination IP address, source port number, destination port number and protocol number. Then we select the important port number from the generated flow information. Because both source and destination port numbers of P2P traffic flow are usually over 1024, it is important to distinguish the important port number for the decision of P2P application from the randomly generated port number.

The third step is to construct the Flow Relation Map (FRM). Most P2P applications use multiple connections to support various functions so that it is possible to discover relationships between flows that belong to the same P2P application. The final step is to make group of flows according to the P2P application name using the results of the previous three steps.

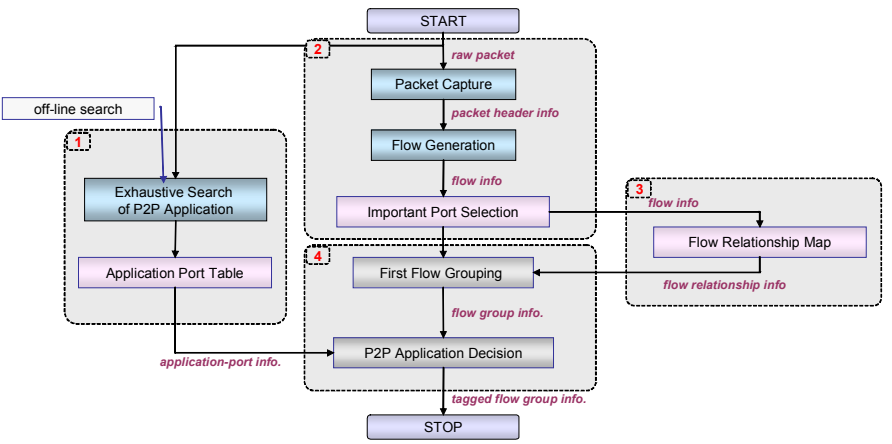



**Fig. 4.** P2P traffic Analysis Algorithm

### 3.1 Important Port Number Selection Method

This method comes from the fact that most of the Internet traffic is TCP traffic and most P2P applications use TCP. Figure 5 shows a normal TCP communication sequence. To establish a connection between a client and a server, the three-way handshaking mechanism is performed using SYN and SYN-ACK packets.

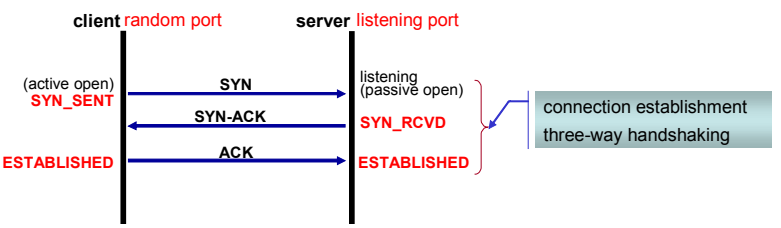



**Fig. 5.** TCP Communication Sequence

In TCP communication, usually the server port number is fixed and not changed, but the client port number is randomly generated by operating system. Therefore, the server listening port is the important port for analyzing traffic. How can the server's listening port number be selected from the captured flow information? We utilize SYN and SYN-ACK packets in the three-way handshaking mechanism. The destination port number in the SYN packet is the server listening port. Likewise, the source port number in the SYN-ACK packet is the server listening port number. Using this information, we can determine the important port number from all the TCP flows.

In case of UDP flows we cannot apply the same method because there is no three-way handshaking mechanism like in TCP. Instead we can use the flow relationship between UDP packets to decide the important port number. We know by experiments that the patterns of UDP flows are very simple compared to TCP flows. So it is not so difficult to find relationships among UDP flows.

### 3.2 Flow Grouping Using APT

To determine the P2P application name from the captured flow information we should know the P2P application names widely used by users. Through the exhaustive search of P2P applications using packet analysis tools such as tcpdump and ethereal [21] we construct the APT which contains the information about each P2P application, as illustrated in Table 1. The APT contains the P2P application names, frequently used TCP/UDP port numbers and one representative port number for each. As Table 1 shows, most P2P applications use multiple port numbers that are not mostly registered at IANA [22]. Some P2P applications use both TCP and UDP.

**Table 1.** An Example of Application Port Table

| Application Name | TCP | | UDP | |
|---|---|---|---|---|
| | representative port | well-known ports | representative port | well-known ports |
| MSN Messenger | 1863 | 1863,6981-6990,14594 | | |
| Yahoo Messenger | 5101 | 5101,5050 | | |
| AIM/ICQ | 5190 | 5190 | | |
| Soribada | 22322 | 22322,7675,7676,7677 | 22321 | 22321,7674 |
| eDonkey | 4661 | 4661,4662,6667 | | |
| Shareshare | 6399 | 6399 | 6777 | 6388,6733,6777 |

We select one port number among the frequently used port numbers by each P2P application and use it as the representative port number of that P2P application. If a P2P application uses TCP and UDP then two representative port numbers are assigned to each protocol respectively. This representative port number is used to indicate the groups of flows belonging to the same P2P application. In the final step of the proposed algorithm, the flows belonging to a P2P application are tagged with the corresponding representative port number. Therefore, all P2P flows are grouped by the tagged representative ports.

### 3.3 Flow Grouping Using Flow Relation Map

The Application Port Table (APT) and Important Port Selection cannot give 100% of accuracy in the decision of P2P application name to all P2P flows. There are two reasons for this. First, there are too many P2P applications around the world to examine. Also, the complete examination to discover all used port numbers is difficult. In many cases, dynamically generated port numbers are the important port numbers. In such cases APT cannot provide flow-grouping information for this P2P traffic. Second, it is also possible that the same important port number is used by more than two P2P applications. In this case, we cannot decide which P2P application generates this flow without flow relationship information. Therefore, we propose the third step, the Flow Relationship Map (FRM), to increase analysis accuracy.

Currently, we are using a basic and simple relationship method among flows. First, the flows are grouped according to the combinations of source port, destination port, and protocol. We give a priority value to each combination of these three flow properties according to the weight of dependency, as illustrated in Table 2 (a). For example, the UDP flows with the source port number 22321 and destination port number 22321 are grouped with priority 100. This processing is called the property dependency grouping.

After this property dependency grouping, all groups are linked with the weight. The weight value is decided by the priority values in the location dependency table, which is illustrated in Table 2 (b). The link weight between two groups is high when the source and destination IP addresses of flows in the two corresponding groups are highly dependent on each other. Otherwise, the weight is low. We call this processing the location dependency grouping.

By these two steps in the grouping method, the flows related to each other are grouped. And this group information is used in the P2P Application Decision process to increase the accuracy of analysis.

**Table 2.** Flow Dependency Table

| | protocol | source port | destination port | priority |
|---|---|---|---|---|
| 0 | | | | 0 |
| 1 | | | 1 | 20 |
| 2 | | 1 | | 20 |
| 3 | | 1 | 1 | 50 |
| 4 | 1 | | | 0 |
| 5 | 1 | | 1 | 50 |
| 6 | 1 | 1 | | 50 |
| 7 | 1 | 1 | 1 | 100 |

(a) Property Dependency Table

| | source ip | destination ip | priority |
|---|---|---|---|
| 0 | | | 0 |
| 1 | | 1 | 10 |
| 2 | 1 | | 10 |
| 3 | 1 | 1 | 100 |

(b) Location Dependency Table

# 4 Design and Implementation of P2P Traffic Analysis System

In this section, we describe the design and implementation of the P2P traffic analysis system using the proposed method. The system was developed as a plug-in to the real-time traffic monitoring analysis system called NG-MON [8].

## 4.1 Design of P2P Traffic Analysis System

Figure 6 illustrates the overall design of the P2P traffic analysis system, which consists of three main modules. They are the APT module, the Important Port Selector module, and the Flow Relations Mapper module.

The Important Port Number Selector module consists of a Packet Capturer, a Flow Generator, and a SYN Packet Table. The Packet Capturer receives raw packets from a network link and generates packet header information from each raw packet. The packet header information is sent to the Flow Generator. If a packet is a SYN or SYN-ACK packet, it is stored in the SYN Packet Table. The SYN Packet Table keeps the TCP listening port information. To select an important port number from each flow, the Flow Generator looks up the SYN Packet Table.

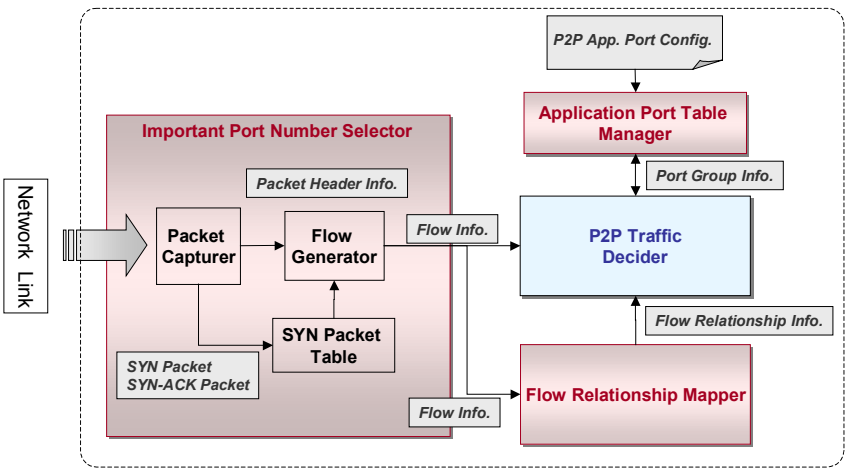



**Fig. 6.** P2P Traffic Analysis System Design

The important port determined flows are sent to the P2P Traffic Decider and the Flow Relationship Mapper. The Flow Relationship Mapper module keeps track of the flow relationship and the history of flows from each host and provides this information to the P2P Traffic Decider Module.

Through the off-line search of each P2P application, we built a P2P port configuration file with XML. We used XML because it is easy to use and many XML-related libraries are provided in various languages, such as C/C++ and Java. The APT manager reads this configuration file and keeps the port group information. When the P2P Decider module receives a flow from the Important Port Number Selector module, it

looks up the Flow Relationship Map and Application Port Table to decide the P2P application name where the flow belongs. Finally, the P2P Traffic Decider groups the flows by tagging each P2P flow with the corresponding representative port number. If the important port number of flow does not belong to a certain P2P application port group but this flow has high relationship with other flows which belong to that P2P application according to FRM, then the P2P Traffic Decider tags this flow with the same representative port number and updates the APT.

### 4.2 Integration of P2P Traffic Analysis System with NG-MON

NG-MON [8] is a real-time Internet traffic monitoring system for high-speed networks, developed at POSTECH. The P2P traffic analysis module is implemented as a plug-in module to NG-MON. Figure 7 illustrates the integration of the P2P traffic analysis module with the current NG-MON system.

The components of the Import Port Number Selector module are separated into the Packet Capturer, Flow Generator, and Flow Store phases. We use the Packet Capturer module of NG-MON as it is. The SYN Packet Table is added into the Flow Generator and the Second Level Important Port Number Selector is located in the Flow Store system. The APT manager, Flow Relationship Mapper, and P2P Traffic Decider are added in the Flow Store system. The P2P Traffic Decider Module is illustrated with the flowchart-like diagram in Figure 7. As a result of the P2P traffic analysis system, the tagged flow information with the representative port number is stored in the Flow Store system. The traffic analyzer determines the corresponding P2P application name of each P2P flow by the representative port number.

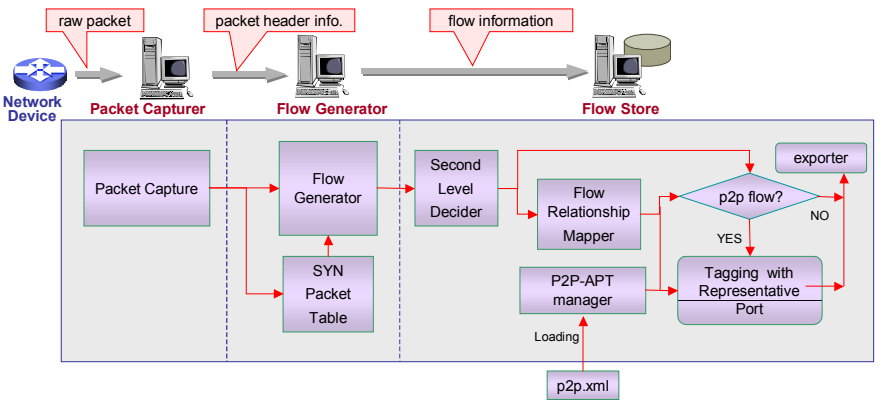


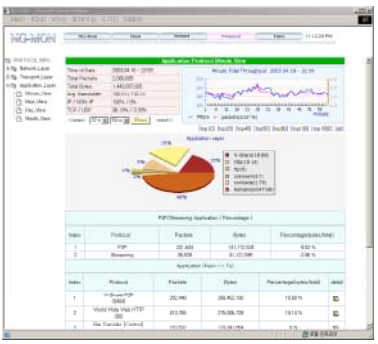

**Fig. 7.** Integration of P2P traffic analysis system with NG-MON

## 5 Result of P2P Traffic Analysis

We have deployed NG-MON with the P2P traffic analysis module in the Internet junction of our campus. Our campus Internet link is composed of two 100 Mbps

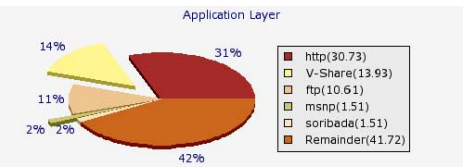
Metro Ethernet links. Considering the bi-directional traffic, the maximum amount of traffic we analyze is 400 Mbps.

We can see the result of P2P traffic analysis at the application protocol view page of the NG-MON Presenter system. Figure 8 (a) shows an example of P2P traffic analysis results. NG-MON captures all the in/out Internet traffic and analyzes them from various points of view, such as throughput analysis per host and subnet, time series analysis of the throughput changes of each host and subnet. Figure 8 (b) and Figure 8 (c) is a result of NG-MON analysis during one week. Figure 8 (b) shows a time series graph of throughput and packet size changes during the tested period. The total amount of captured data size is 11,493,562,602,529 bytes from 17,427,364,409 packets. The average bandwidth was 152.03 Mbps. The ratio of TCP and UDP among total IP packets was 82.3% and 9.9%, respectively.



(a) Application protocol view page


(b) Time series graph of Internet traffic for one week




(c) Application layer traffic distribution for one week

**Fig. 8.** Analysis Result of P2P Traffic

Figure 8 (c) shows the application layer analysis result where our P2P analysis mechanism is applied. HTTP traffic occupies the largest part of the pie chart; it is 30.73% of total IP traffic. But the second is not FTP data; the proportion of FTP traffic is only 10.61% and the third. The second largest traffic is V-share P2P application traffic [22]. The fourth and fifth largest traffic is generated by the MSN messenger application [12] and the Soribada file sharing application [9]. We examined 20 popular P2P applications and made an Application Port Table (APT). The proportion of these 20 P2P applications was 32.53% of total traffic. The percentage will increase if we examine more P2P applications.

## 6 Conclusion

In this paper, we have presented a new algorithm for analyzing P2P traffic. First, we explained the properties of P2P traffic and the reasons why the existing analysis mechanism is unsuitable for P2P traffic analysis. The proposed algorithm consists of four main components: the Important Port Number Selection, the Application Port Table, the Flow Relationship Map and the P2P Traffic Decider. Using this proposed

algorithm we designed a P2P traffic analysis system and implemented it as a plug-in to NG-NOM. Using this system we were able to analyze considerable amounts of unknown traffic which could not be determined by the traditional analysis method. The result of P2P traffic analysis on our campus Internet junction shows that the proportion of P2P traffic is steadily increasing.
The proposed algorithm can be improved still further including the Flow Relationship Map. By more experimental tests on our campus Internet junction, the efficiency of the proposed algorithm will be validated. In addition to the validation of our algorithm, we are going to apply proposed flow grouping algorithm to the analysis of other types of Internet traffic, such as game and streaming media traffic.

## References

[1] Ian D Graham and John G Cleary, "Cell level measurements of ATM traffic," Proc. of the Australian Telecommunications Networks and Applications Conference, pp. 495-500, Dec. 1996.
[2] Cisco, White Papers, "NetFlow Services and Applications," http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm
[3] P. Phaal, S. Panchen and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks", IETF RFC 3176, September 2001.
[4] N. Brownlee, C. Mills and G. Ruth, "Traffic Flow Measurement: Architecture", IETF RFC 2722, October 1999.
[5] N. Brownlee, "Traffic Flow Measurement: Experiences with NeTraMet", IETF RFC2123, March 1997.
[6] Ken Keys, David Moore, Ryan Koga, Edouard Lagache, Michael Tesch, and k claffy, "The Architecture of CoralReef: An Internet Traffic Monitoring Software Suite," PAM Workshop 2001, April, 2001.
[7] Argus, http://www.qosient.com/argus/
[8] Se-Hee Han, Myung-Sup Kim, Hong-Taek Ju and James W. Hong, "The Architecture of NG-MON: A Passive Network Monitoring System", LNCS 2506, DSOM 2002, October 2002, Montreal Canada, pp. 16-27.
[9] Soribada, http://www.soribada.com/
[10] Morpheus,http://www.morpheus.com/
[11] Gnutella, http://gnutella.wego.com
[12] MSN Messenger, http://messenger.msn.co.kr/
[13] Yahoo Messenger, http://kr.messenger.yahoo.com/
[14] eDonkey, http://www.edonkey2000.com
[15] Matei Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network", Technical Report TR-2001-26, University of Chicago, July, 2001.
[16] Subhabrata Sen and Jia Wang, "Analyzing Peer-to-Peer Traffic Across Large Networks", IMW2002 Workshop, 2002, Marseille, France.
[17] Microsoft, Windows Media Technology, http://www.microsoft.com/windows/windowsmedia/default.asp

[18] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," RFC 2336, April 1998.
[19] Jacobus van der Merwe, Ramon Caceres, Yang-hua Chu, and Cormac Sreenan, "mmdump- A Tool for Monitoring Internet Multimedia Traffic," ACM Computer Communication Review, Vol. 30, No. 5, 2000.
[20] Hun-Jeong Kang, Hong-Taek Ju, Myung-Sup Kim and James W. Hong, "Towards Streaming Media Traffic Monitoring and Analysis", APNOMS 2002, September 2002, Jeju, Korea.
[21] Ethereal, http://www.ethereal.com/
[22] V-share, http://www.v-tv.co.kr/

# MobiMan: Bringing Scripted Agents to Wireless Terminal Management*

Venu Vasudevan, Sandeep Adwankar, and Nitya Narasimhan

Mobile Platforms and Services Department, Motorola Labs,
1301, E. Algonquin Road, ILO2-2240, Schaumburg, IL 60196
{venuv,adwankar,nitya}@labs.mot.com

**Abstract.** The increasing software complexity of wireless devices and wireless data service provisioning motivates a wireless terminal management challenge. The systems management solution for this problem needs to scale up to large device populations, while being lightweight enough to be pragmatic for resource-constrained devices. The work in this paper builds upon the emerging *SyncML* standard for wireless terminal management in order to bring sophisticated policy-based management to large populations of wireless data devices. It is anticipated that this technology will simplify the upgrade and management of wireless data devices substantially, thus encouraging the adoption of sophisticated data terminals.

## 1 Introduction

The increasing complexity of wireless devices and services motivates an automated terminal management challenge. Complex client-side wireless tools (e.g. WAP browsers) need to be remotely configured upon service activation or upgrade. Mobile service operators desire the ability to dynamically upgrade applications and services on a mobile device, motivating the need for scalable software distribution capabilities. Effectively supporting a complex palette of applications on a consumer-oriented device requires pro-active diagnosis and troubleshooting of both the devices and the network. While these tasks can be done in an operator-assisted fashion, the absence of a *scalable, automated management infrastructure* can contribute to a high total cost of ownership. Early studies [KD99] estimate this number as being several times the retail cost of the device

The goals of wireless terminal management resemble those of "classical" (wired) systems management in terms of scaling and automation. However, achieving the goals in a resource-constrained, intermittently connected environment presents unique technical and business challenges. The pervasiveness of the Simple Network Man-

* We would also like to thank Kevin Cutts and Hung Tsang from Motorola's PCS business unit for many constructive discussions in this context.

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 68-81, 2003.

agement Protocol (SNMP) and its thick-manager, thin client-agent architecture make it a logical first choice. However, while SNMP agents can be deployed on thin devices, traditional client-server SNMP lacks the "elastic server" or delegation model [GY95] that is required for effective management in intermittently connected environments.

Advanced proposals within the SNMP community (such as the SNMP mid-level manager architecture [SMLM, SNMX]) provided a standards-compliant basis for a delegation model but failed to gain the widespread traction of core SNMP. Mobile agent technology can provide a delegation model for systems management that is agnostic to the management protocol. However, mobile agent infrastructure is too heavyweight for mobile devices, requiring some programming features (e.g. dynamic classloading) that are not supported by current mobile device application platforms like J2ME.

From a pragmatic point of view, another problem with SNMP is its lack of uptake in the mobile wireless space. The wireless industry, which seeks a single, integrated standard to manage device-resident information as well as device operation, has rallied around SyncML [SM02, JN01] and SyncML for Device Management (SyncML-DM)[1] [SD02] as the management standard of choice. SyncML caters to both the information management and device management needs of the wireless industry, while supporting a lightweight architecture using an XML-based command language.

While SyncML-DM is best suited for simple request-response management operations (analogous to a single SNMP *Get* and *Set*), complex operator management functions require richer predicates and procedures to capture the necessary semantics. For instance, a network operator may want to run an automated management test on all cell phones in the "847" area code. Furthermore, he may want this test to run during off-peak hours and only on terminals with sufficient battery-power levels. Although basic SyncML-DM provides the low-level mechanisms to collect the relevant telemetry, expressing such activation policies as a collection of primitive SyncML-DM interactions could be prohibitively expensive.

### 1.1 The MobiMan Architecture

In the **MobiMan** architecture, we explore an alternative "embrace and extend" approach to bringing the benefits of *scripted mobile agents* to SyncML-based wireless terminal management. MobiMan defines a SyncML-derived scripting language called ***Symple*** (**SY**ncM**L** **P**rogramming **L**anguag**E**) that extends SyncML semantics in a lightweight manner suitable for resource-constrained devices. The MobiMan architecture also extends the SyncML runtime framework with support for the scheduling, evaluation and lifecycle management of Symple agents. To strike a balance between capabilities and deployment costs, we designed Symple according to the following principles:

- *Embeddable within SyncML.* Symple agents can be embedded within standard SyncML packets, allowing us to reuse SyncML as an agent distribution protocol.

[1] Henceforth, the terms SyncML and SyncML-DM will be used interchangeably to mean SyncML-DM.

- *Lightweight.* Symple is easy-to-learn, requiring only a small amount of code to define complex requirements that can be interpreted by a lightweight client-side SyncML platform.
- *Extensible.* Devices can support different variants of Symple in accordance with their resources and computing capabilities, with Synclets being discarded without error by Symple-unaware, SyncML capable devices.

Given that the SyncML standard is of recent vintage, we provide a quick tour of SyncML in Section 2, followed by an overview of the MobiMan architecture in Section 3, with focus on the computing elements (Synclets) and the client-side runtime "container" architecture (Micropods). Section 4 delves deeper into the structure of Synclets, and their support for conditional execution. We conclude the paper with a discussion of our experiences building a Synclet-based system on Motorola wireless devices, future directions for our research, and comparisons of our work to related ideas in distributed systems management.

## 2 SyncML: A Short Tour

The SyncML standard[2] [SM02] was developed as XML-based information synchronization standard designed specifically for the needs of the wireless industry. Thus, the SyncML protocol is lightweight to cater to device limitations, language-neutral and protocol-neutral. To suit device limitations, SyncML language constructs are kept fairly minimal, with the protocol not using some features (e.g., server sockets) that are yet to become pervasive on mobile devices. Because SyncML is XML-based, it is inherently language-neutral, and supports OEM-specific extensibility. In addition, the SyncML "protocol" can run over a number of underlying transport protocols including HTTP, WSP, and OBEX.

SyncML's popularity in information synchronization led to its scope being expanded via SyncML-DM [SD02] to include device management. SyncML-DM allows management actions to be performed on management objects, where a management object might represent a device configuration or the run-time software application environment. Actions taken against the former might include reading and setting parameter keys and values, while actions taken against the latter might include installing, upgrading, or uninstalling software elements. The signatures of the Get and Set methods on a management object are type-specific and may vary substantially in complexity. For instance, a management action for setting the device clock accepts a simple textual MIME type (text/plain), while an action to change the WAP browser settings requires new WAP provisioning "blobs" to be transmitted as part of the Set operation. Software upgrades present another example of a management action with significant payload complexity.

The SyncML-DM is a 2-phase protocol consisting of a *setup* phase for authentication and device information exchange, following by a *management* phase that can be

[2] It was developed first in a separate standards body, which has since been merged with the Open Mobile Alliance (OMA).

repeated multiple times to support complex manager-to-mobile sessions. A management session may also start with Packet 0 (the trigger), where the trigger may be out-of-band depending on the environment

While SyncML-DM significantly expands the scope and utility of SyncML to include device management, it is limited in the following ways :

1. SyncML-DM based device management is limited to *terminal-at-a-time* management. It allows a cellular operator to run only one diagnostic operation at a time on each terminal.
2. SyncML-DM does not support intelligent postponement of management operations (e.g. postpone terminal operation until battery level is above 50%), something which is necessary to scale terminal management to large terminal populations.
3. SyncML-DM does not allow operators to schedule *coordinated* terminal management operations across collections of terminals.

These limitations restrict the subset of "Opex" (operational expense) minimizing management functions that a cellular operator can perform using SyncML. The goal of MobiMan is to add a more powerful computing abstraction to SyncML-DM to facilitate more comprehensive, automated, scalable systems management

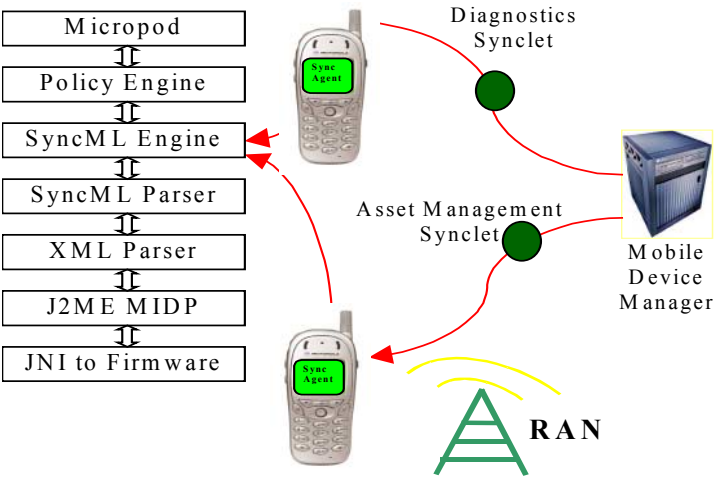



**Fig. 1.** The MobiMan runtime architecture

## 3 MobiMan: Architecture and Runtime

As shown in Figure 1, MobiMan augments the standard SyncML-DM runtime framework in terms of both the basic execution entities and the execution model. In doing so, it extends the SyncML-DM framework with two elements: ***Synclets*** and ***Micro-***

***pods***. ***Synclets*** are scripted agents written in Symple, allowing complex sequences of management instructions to be expressed in a single executable object. ***Micropods*** are terminal-resident containers that have the ability to receive Synclets and manage their lifecycle; for instance, micropods can activate and deactivate Synclets based on state.

The key to the MobiMan container model is to organically expand SyncML to support *intelligently postponable* computing objects, whose execution triggers are sophisticated. A cellular operator can exploit this sophistication to perform large management operations in a manner that conserves bandwidth, avoids terminal operations when the terminal is in an unsuitable state, and coordinates complex multi-terminal operations with complex orchestration policies. We describe Synclets and Micropods in detail in the following sections.

### 3.1 Synclets and Synclet Bundles

*Synclets* are the basic unit of computation in *MobiMan*. A Synclet is an executable script consisting of Symple commends, where Symple is an extension of SyncML. A Synclet specification comprises of two parts: a *policy* and an *action routine*. The Synclet *policy* specifies non-functional aspects of the Synclet such as if, when and how often it should be executed. The *action routine* is the functional code that makes up the Synclet. Synclets are interpreted and executed by the extended client-side SyncEngine known as a Micropod.

Synclets are transported over the SyncML protocol, which in turn is bearer-agnostic. Thus Synclets may be transported over HTTP, SMTP or other IP-based protocols. To be compatible with Synclet unaware clients, Synclets are carried as the payload – i.e., as nested tags – in a *Synclet* XML element within the SyncML body. Standard SyncML engines that are not MobiMan-enabled will simply ignore the Synclet scripts within the enclosing Synclet tags. As Synclet upload and execution are decoupled, multiple Synclets with differing execution policies may be carried in a single SyncML session, i.e. nested within the same *MobiMan* tag.

Synclet *bundles* are a convenience mechanism, analogous to Java packages or modules in programming languages. The bundle mechanism allows a group of Synclets to be loaded in a single SyncML session, and be henceforth accessed by the remote operator using a single bundle name. The bundle notion facilitates packaging, distribution and policy management of Synclets.

### 3.2 Micropods

*Micropods* are terminal resident containers layered over the standard client-side *SyncEngine*. They host and manage Synclets that are dispatched to the terminal. Micropods perform functions relating to Synclet lifecycle management, and serve as secure sandboxes for Synclet execution. Lifecycle functions include (de) activation, multitasking between concurrently executing Synclets, and the suspension and the revival of Synclets that were stopped due to adverse terminal conditions (e.g. deteriorating battery level). To avoid race conditions on the device, Micropods support "virtual multitasking" where multiple terminal-resident Synclets might have partially executed at any point in time, but only one Synclet is actually executing at any instant.

As previously mentioned, Synclet specifications include a model of conditional, policy-based execution analogous to the UNIX *cron* model. Micropods augment the standard SyncEngine with a *policy engine*, which determines the subset of Activatable Synclets based on evaluating individual Synclet policies. Policies could be based on absolute time, invocation cardinality (number of times a Synclet should be run), and device state. More sophisticated micropods could support the interleaved execution of multiple synclets. These could include allowing a certain maximum number of concurrent synclets, deadlock avoidance or resolution between concurrent synclets, and fairness policies for Synclet swap-out whereby idle synclets are swapped out for other waiting synclets.

As with Java applet environment and J2EE servlet containers, micropods provide a constrained environment to executing synclets, bounding and monitoring their access to the underlying terminal. Synclets are privileged applications have limited access to retrieving and modifying the device state via a special set of "closed classes" in Java. Two factors simplify sandboxing in MobiMan. First, the Synclet dispatching capability is accessible only to a small number of trusted parties (namely service operators). So authentication solutions such as digital signatures can check the signature against a very small universe of trusted sources. Secondly, the use of a scripting language allows for language safety verifiers to be developed, and for sandboxes to suspend (and resume) the script at arbitrary points in the program.

Synclets face the unique situation in executing on mobile wireless devices, namely that the device may be powered off at any time without operator control. Micropod lifecycle management techniques need to allow for script re-entrancy in such adverse situations. Script re-entrancy might involve re-prioritizing Synclets when they are restarted, to reflect the new environment (analogous to adjusting the *nice* value of processes in Unix). This is handled by re-evaluating Synclet policies of awakening Synclets in the changed environment.

## 4 More Synclet Anatomy

As described previously, Synclets consist of *policy* and *action routine* components. The action routine is the body of what the Synclet does, and the policy is the trigger condition for the Synclet. The policy, action routine separation allows the same functional Synclet to be reused in different circumstances. This section elaborates on the kinds of policies supported in Symple, and the SyncML extensions supported in an action routine. SyncML is extended in two ways in the action routine language: the addition of a few new commands, and support for conditional command execution (aka *guarded* commands)**.** environment.

### 4.1 Policies

Policies are specifiable at three levels of granularity: *terminal*, *synclet bundle*, and *synclet*. *Terminal* level policies apply to all scripts that will execute on the terminal from the time the policy is installed. For instance, a terminal policy may dictate that only one Synclet shall be active at a time. *Synclet bundle* policies apply to a collection

of synclets that were loaded as an aggregate, perhaps because they collectively perform a single cohesive task. *Synclet* policies govern the execution of the Synclet's action routine, and may include the maximum time a Synclet is allowed to run, resources that should be allocated before the Synclet should be activated, or recovery actions when Synclet execution is interrupted. Terminal policies tend to be more static and persistent than Synclet bundle and Synclet level policies.

### 4.2 Extended SyncML Command Set

SyncML provides a fairly minimal set of commands for device management, but is missing some important primitives that are building blocks for device and service management. *Symple* currently extends the SyncML command set with three commands: *assert*, *schedule*, and *perform*. All these commands may contain a redirect block (see section 4.3 below).

```
<Synclet>
  <Guard>
     <Attribute> Battery_level </Attribute>
     <Condition> GREATER_THAN </Condition>
     <Threshold> 5 </Threshold>
  </Guard>
  <Assert>
     <Item>
        <Target>
           <LocURI>
              ./Sync/DM/WAP/WAPSTNG2/GRPS_APN
           </LocURI>
        </Target>
        <Data>
           internet2.voicestream.com
        </Data>
     </Item>
  </Assert>
</Synclet>
```

**Fig. 2.** Synclets with guarded commands

1. *Assert* (see Figure 2) allows the Synclet to assert a certain device (or network) condition, and take an exception action in case this isn't true. Assert is useful where the absence of a condition (e.g. non-null values for WAP session parameters) requires corrective action.
2. *Schedule* is used to represent timed and/or repetitive commands. Parameters in the schedule command may specify the maximum number of times the command is executed, the time interval between successive iterations, and the delay between the loading of a Synclet and the first “run” of the scheduled command.
3. *Perform* performs a non-local service invocation (e.g. an http *get*) from the terminal. Network operators can use this to measure service performance across a statistically significant number of terminals.

While our present extension to the SyncML command set is restricted to these operations, we envisage future extensions to facilitate service management.

### 4.3 Guarded Commands and Output Redirection

At the finest level of granularity are policies that govern the execution of individual SyncML commands (or command blocks) within the action routine. However, we tend to not view these in the same way as the policies described in the previous section, as these are part of the functional definition of the action routine and cannot be viewed as orthogonal to the script function. Commands governed by such policies are also referred to as *guarded* commands, and the policy pertaining to the command as the *guard*. A battery-level *guard* might govern a command that fetches a number of terminal attributes. This guard would prevent the command from executing if the battery-level on the terminal is below a certain threshold. Figure 2 shows a guarded command with a battery guard**.** The standard SyncML command is prefixed by a guard element that constrains the GET to execute only when the battery level on the terminal is above the threshold. Guards may be used to protect terminal or network resources, and to coordinate with external events.

Traditionally, the results of a SyncML session are returned synchronously to the caller when the command(s) are complete. In Symple, we allow a more flexible output communication by allowing the results of partially executed scripts to be exported to URLs external to the terminal, and by allowing commands within an action routine to post their results to different URLs. *Redirect* is an output redirection primitive in Symple that delivers the output of a command to the appropriate URL Redirect will support a variety of standard protocol prefixes including http, sockets and RMI.
Allowing external access to partial script results allow the operator greater visibility into script execution and greater flexibility in controlling the script. The operator may decide to shut down an otherwise expensive and long-running script based on viewing the partial results, or he might decide to take alternative actions based on what is observed. Partial result availability is also useful in scaling the Symple paradigm to concurrent multi-terminal operations, where the partial results of a running script on one terminal may cause side effects on another.

## 5 Implementation and Usage Experience

MobiMan has been used in a number of operator management scenarios, of which one pertaining to wireless system performance management is described here. A wireless network operator may be interested in performance metrics such as average latency experienced by users in a particular geographic area, perhaps as a way to validate the service level agreement with an enterprise customer.

Latency can be characterized by the time taken to download a HTML or WML page from a remote server. To measure latency over multiple devices and a statistically meaningful period of time, the operator can compose guarded synclet (see Figure 3), and can push it to of the mobile device fleet.

The synclet will execute at the time specified by the date attribute and will connect to specified *URI* to download HTML page. It will repeat this operation for times as specified in *repeat* attribute. This synclet will average these latency measurements and send it to operator server. The server correlates the values obtained from number of

mobile devices over period of time at different times and can collate the data to make decisions about re-provisioning the network.

```
<Synclet>
  <Guard>
     <Attribute> Signal_Strength </Attribute>
     <Condition> GREATER_THAN </Condition>
     <Threshold> 20 </Threshold>
  </Guard>
  <Schedule>
     <Date> 01028526240000 </Date>
     <Period> 1 </Period>
     <Repeat> 20 </Repeat>
     <URI> http://www.yahoo.com/index.html </URI>
     <Item>
        <Target>
           <LocURI>
              ./Sync/DM/Performance/Network/Latency
           </LocURI>
        </Target>
     </Item>
  </Schedule>
</Synclet>
```

**Fig. 3.** Synclet with guarded commands for latency measurement

Table 1 details our experience in running performance management Synclets over a GSM network on a Java-enabled mobile handset. The data shows a total time of about 30 seconds for Synclet loading and execution. While this number is acceptable, it could benefit from improvement. Better networks will reduce this latency, while Synclet bundling allows latency to be amortized across multiple management operations. Synclets can be encoded in Wireless Binary XML (WBXML) instead of plain XML representation that can significantly reduce size of data sent (and hence time) over wireless network. WBXML is a binary format compact representation of XML that encodes the structure and content of the document entities while removing meta-information contained in document.

The Micropod, policy engine along with SyncML engine and parser occupy 100K on MIDP KVM with peak runtime memory consumption of 230 K. Local runtime operations on the mobile device (e.g. Synclet parsing) make up about 25% of the total cycle-time, and will improve as Moore's law increases the horsepower dedicated on handsets to data services. Overall, the numbers indicate the viability of a Synclet based approach for today's handsets, and its growing value with handset and network evolution.

**Table 1.** Synclet Execution. The phases, and some performance measurements

| | **Synclet Operation flow** | **Size of data sent** (bytes) | **Time taken to send data over GSM network** (seconds) |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1. | **Creation of SyncML package 1**<br>Server initiates SyncML Engine on mobile device (e.g.. by SMS push). SyncML Engine creates SyncML package with capabilities data such (e.g. Manufacturer/ Model name. and credentials. | 1185 | 3.3 |
| 2. | **Getting Synclet from Server**<br>Mobile device sends SyncML package DM Server. Server parses the package, checks for credentials and synclets that need to be sent to the device. Server returns new SyncML package containing synclet (HTTP reply). | 1822 | 18.5 |
| 3. | **Invoking Synclet**<br>SyncEngine parses/extracts Synclet transfers to Policy Engine. Policy engine verifies synclet safety and semantics, conditionally schedules Synclet. Synclet reception status sent back to server. | 745 | 8.2 |
| 4. | **Status/Result of Synclet**<br>Server responds to synclet receipt status message. | 595 | 11.3 |
| 5. | **Status of Synclet**<br>Mobile device parses SyncML message and continues to process synclet. | | 2 |
| 6. | **Synclet Execution**<br>Scheduled Synclets are executed in separate thread. For long-running synclets (e.g., latency monitoring), steps 4 and 5 are repeated. Results sent back to server. | | |

## 6 Intelligent Distribution of Tasks in MobiMan

Thus far, the MobiMan architecture has focused on the design and deployment of *Synclets* as a means for intelligent *scheduling* of tasks at the targeted terminal. However, the richer predicates offered by Synclet-enhanced terminal management also enables operators to define tasks that span multiple terminals and involve complex orchestration policies for successful completion.

This functionality is supported by a *SmartCloud* extension [NA03] that integrates a tuplespace-backend (provided by the **Mojave**[3] system [VL01]) into the MobiMan server infrastructure. The SmartCloud extensions provide three core mechanisms:

[3] The Mojave System developed at Motorola Labs provides a tuple-space based agent architecture where task agents can dynamically clone/relocate to the agent container that best facilitates task completion.

1. **Intelligent TM Server Selection.** Current TM operations assume the existence of a centralized server whose identity is known to the TM client. This raises performance and reliability concerns as the server becomes both a bottleneck and a single point of failure. Instead, we envision a "multi-server" approach where multiple TM servers exist, any of which are capable of delivering the task to the device. The TM servers could be co-located as part of a "server farm" or could be in deployed as individual "kiosks" in high-traffic areas such cafeterias, banks and airports. Choice of server for task dispatch is now based on *opportunity*, i.e., the association[4] of a TM client with a particular server causes that server to register with the SmartCloud as the dispatcher for that TM client. If a client associated simultaneously with two dispatchers (cellular and short-range), the dispatcher with better QoS criteria (e.g., higher bandwidth) is chosen.
2. **Task Dispatch Priority Escalation.** Intelligent server selection raises the question of how long the SmartCloud should wait for the right "opportunity". Operators (and users) may prefer the user of the quicker, more reliable short-range network for TM operations; however, users may not always be within range of a suitably equipped server. The SmartCloud extension solves this problem by using an "escalation policy" for task dispatch. Operator submit tasks with an appended deadline for task dispatch.. The SmartCloud will hold on to tasks, waiting for the *best* dispatch opportunity; however if a specified deadline is near expiry, the SmartCloud becomes more aggressive, opting to use the *first* opportunity it sees.
3. **Automated Coordination Through Aggregation.** While the multiple-server model improves the distribution of TM tasks, it can complicate the coordination of complex tasks that span multiple terminals (e.g. an operator request for 100 terminals in the "847" code to respond with bandwidth availability information). Here, the single operator request actually translates into multiple terminal-specific tasks – each being dispatched to the target over a potentially different server. Monitoring the tasks, coordinating follow-up actions and returning a unified result to the operator can become a logistics nightmare. The use of a SmartCloud backend alleviates this problem by providing a common backend "shared memory" structure that can be used to aggregate results from various tasks and automate the firing of follow-on tasks such as request-termination, and result-display.

By integrating the SmartCloud backend into the MobiMan server infrastructure, we achieve three objectives: *flexibility, efficiency* and *ease-of-use*. Operators are now required only to design the task using Synclet semantics to define the appropriate criteria for task execution. The MobiMan system ensures that the tasks are delivered within deadline – and in the most effective manner – to the targeted terminal(s). It also *automates* the monitoring and completion of coordinated multi-terminal tasks thereby reducing the burden on the operator and minimizing opportunities for operator error.

[4] An association (client-server communication) could happen over a wide-area (e.g. cellular) or a short-range (Bluetooth, Adhoc WiFi) network.

Furthermore, because the SmartCloud extensions are server-centric, they add minimal resource or execution overhead to the resource-constrained mobile terminals.

## 7 Related Work

A number of lightweight programming languages [LL01] provide design examples that Symple tries to emulate. SIMSpeak [KM01] aims to provide programmability to devices whose only programmable component is an extremely limited "smart card" supporting the Javacard specification. An on-card interpreter supports a simple stack-based language that uses registers instead of variables to save space. Similar choices of language primitives are made in 3GPP's USAT specification [3GPP]. Mobile code is pushed to the device via the short-message service (SMS) and security is handled partly in the device and partly in a network gateway. MobiMan supports an SMS based script dispatch in a manner similar to SIMSpeak, but has the luxury of living in a slightly less constrained Java environment than the Javacard.

The SNMP world has at least two proposals that augment the classic client-server SNMP model with scripted agents. The SNMP mid-level manager [SMLM] proposal includes a scripting language proposal called SNMPScript that is used to specify and distribute scripted agents to managed nodes. SNMX [SNMX] is another script proposal, although an SNMX interpreter weighs in at a "chunky" 400KB. A size that would be somewhat taxing on today's J2ME powered mobile wireless devices.

Sloman [MS98] and others have made a case for policy-based systems management as a means to change management policies without changes to management agents. Some of their proposed primitives (e.g. positive obligation policies) resemble those proposed in this paper. However, Symple adopts a lightweight policy framework to cater to resource-constrained terminals, includes device states as policy predicates, is tightly integrated with SyncML, and supports multi-terminal policies via a network-resident coordination infrastructure.

Heidemann [HS00] proposes a Cron-derivative (based on Xcron [GK99], another cron derivative) for intermittently connected laptops, and articulates intermittent connectivity issues similar to those discussed by us. However, the policy language here focuses on *time*-based policies, and has no support for policies based on device state.

## 8 Conclusions

MobiMan aims to provide complex scheduling primitives to wireless systems management, while operating within the limitations of resource-constrained Java devices. So far our experience has been encouraging, showing that a fairly sophisticated set of capabilities can be supported on a wireless terminal. Easy authoring of coordinated operations across large sets of terminals (e.g. a cellular region) remains a challenge. This requires application infrastructure support in the network and interaction primitives to be defined for greater interplay between network controller objects and the currently executing process on a particular terminal. Emerging smart wireless termi-

nals that support *pJava* (or J2ME/CDC) provide another systems management innovation opportunity, as they are substantially more resource-rich than the *kJava* (J2ME/CLDC) devices targeted in this paper.

## References


[GK99] G. Kuenning, *A Cron Daemon for Portable Computers*, UCLA Computer Science Department Technical Report UCLA-CSD-990044, Sep 1999

[GY95] G. Goldszmidt and Y. Yemini, Distributed *Management by Delegation*, in Proc. of the 15th ICDCS Conference, IEEE Computer Society, pp 333-340, 1995

[HS00] J. Heidemann and D. Shah, *Location-Aware Scheduling with Minimal Infrastructure*, In USENIX Conference Proceedings, pp.131-138, Jun 2000

[JN01] A. Jonsson. and L. Novak, *SyncML – Getting the mobile Internet in sync*, Ericsson Review No. 3-2001, pp. 110-115

[KD99] K. Dulaney, *TCO for PDAs: Higher than Expected*, Strategic Planning, SPA-08-7900, Research Note, Gartner Group, July 1999

[KM01] R. Kehr and H. Mieves, *SIMspeak – Towards an Open and Secure Application Platform for GSM SIMs*, in Proceedings of the Intl. Conference on Smart Cards, E-smart 2001, Lecture Notes in Computer Science, pp. 135-149, Springer 2001

[LL01] MIT Lightweight Languages Workshop, 2001, http://ll1.mit.edu/

[LS01] D. Levi and J. Schoenwaelder, *Definitions of Managed Objects for the Delegation of Management Scripts*, IETF Network Working Group RFC, Aug 2001, www.ietf.org/rfc/rfc3165.txt

[MS98] M. Sloman, *Policy Based Management of Telecommunication Systems and Networks*, First UK Programmable and Telecommunications Workshop, HP Labs, 1998

[NA03] N. Narasimhan, S. Adwankar and V. Vasudevan, *SmartCloud: Automated, intelligent task distribution in MobiMan*, Internal Draft, Motorola Labs, 2003

[OS96] O. Shivers, *A universal scripting framework, or Lambda: the ultimate "little language"*, in Concurrency and Parallelism, Programming, Networking and Security, Lecture Notes in Computer Science, pp. 254-265, Springer 1996

[SD02] *SyncML Representation Protocol Device Management Usage*, version 1.1, at http://www.openmobilealliance.org/syncml February 2002

[SM02] *SyncML Data Synchronization and Device Management*, official website http://www.openmobilealliance.org/syncml

[SMLM] *SNMP Research: Mid-Level Manager (MLM)*, White Paper, SNMP Research, http://www.snmp.com/products/mlm.html

[SNMX] SNMP Frameworks, Inc., *The Simple Network Management Executive (SNMX) Scripting Language* , http://www.snmx.com

[3GPP] 3GPP Technical Specification Group Services and System Aspects: USIM/SIM Application Toolkit (USAT/SAT), Doc# 3G TS 22.038 v5.2.0 (2001-02)

[VL01] V. Vasudevan and S. Landis, *Malleable Services,* International Journal of Software Engineering and Knowledge Engineering, Vol, 11, no. 4, pp. 389-406, 2001

# Dynamic Surge Protection: An Approach to Handling Unexpected Workload Surges with Resource Actions that Have Lead Times

E. Lassettre, D. W. Coleman, Y. Diao, S. Froehlich, J. L. Hellerstein, L. Hsiung, T. Mummert, M. Raghavachari, G. Parker, L. Russell, M. Surendra, V. Tseng, N. Wadia, and P. Ye

IBM Corporation
{edlass,xinu,diao,stevefro,hellers,larryh,mummert,raghavac,gkp,lancerus, suren,vtseng,noshir,ye}@us.ibm.com

**Abstract.** Today's information technology departments have significantly varying demands for resources due to unexpected surges in subscriber demands (e.g., a large response to a product promotion). Further complicating matters is that many resource actions done in response to surges (e.g., provisioning or de-provisioning an application server) have substantial delays (lead times) between initiating the resource action and its taking effect. This paper describes dynamic surge protection, an approach to handling unexpected workload surges in systems that have lead times for resource actions. Dynamic surge protection incorporates three technologies: adaptive short-term forecasting, on-line capacity planning, and configuration management. The paper includes empirical results from evaluations done on a research testbed, including favorable comparisons with a threshold-based heuristic. The results from an extended test also show that service objectives can be maintained cost-effectively.

## 1 Introduction

Today's information technology (IT) departments are besieged with uncertainty. New applications are deployed, but their resource demands are unknown. Traditionally, these situations have been addressed by over-provisioning IT resources and/or manual resource re-allocations. Unfortunately, these approaches are costly – the former in terms of equipment, license, etc and the latter in terms of expert operators. Furthermore, many resource actions involve **lead times**, such as server provisioning, which introduces delays between action initiation and effect. We present an approach to moderate the effect of unexpected workload surges so as to preserve a service level objective (SLO) in a cost effective way while taking resource actions with lead times.

Examples of subscriber overloads abound. On September 11, 2001, the CNN web site was overwhelmed by traffic that doubled every 7 min to a peak of 20 × normal volume [7]. The Victoria's Secret web site had a similar experience as a

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 82–92, 2003.

result of an advertising campaign during the 1999 Super Bowl [10]. Others have noted that "sites such as Encyclopaedia Britannica, egg.com, and H&R Block have suffered massive overload from subscribers" [17].

Many systems (e.g., [15], [11], [19], [13]) have been developed to adapt to changes in workload. Sun's N1 [15] and HP's Utility Data Center (UDC) [11] initiatives provide convenient ways for operators to move resources between applications. Underlying this is the ability to describe logical application topologies and the physical configuration. However, no automation is provided to determine when to move resources between applications. The MVS workload manager incorporates algorithms for adjusting resource allocations to achieve SLOs [1]. These algorithms assume that actions take effect immediately (e.g., changing CPU priorities) and so do not address actions with substantial lead times. Another relevant technology uses load forecasting together with performance estimation to balance file allocation across a network attached storage system to meet a response time objective under varying load [9]. The ThinkProvision technology of Think Dynamics provides automation for model-based optimization of dynamic resource provisioning [16], and the DynamicIT technology of ProvisionSoft uses long-term forecasts to anticipate time-of-day effects [14]. An important difference between the the current work and approaches that use long-term forecasting is that the latter is appropriate for workloads with periodic variation. Conversely, it is not suited for unexpected workload surges, which have no regular pattern.

As reported in [3], there is considerable benefit in rapidly adjusting resource allocations to handle variable workload. This is a challenge if reallocating resources has substantial lead times. One approach is to reduce the lead times, but this may be of limited utility. A second incorporates long-term forecasting to predict workloads and initiate resource actions sufficiently early to accomodate lead times. However, long-term forecasters can require substantial data to learn patterns and thus work poorly for unexpected surges.

In line with IBM's autonomic computing initiative for self-managing systems [6], we describe a system that has self-configuring characteristics. Our approach, which we refer to as **dynamic surge protection**, employs three technologies: adaptive short-term forecasting, on-line capacity planning, and configuration management. The forecasting approach we use is designed to be responsive to rapid changes, yet robust. On-line capacity planning determines resources needed to preserve service levels in a cost effective way (e.g., releasing resources when not needed). Configuration management provides the means for adjusting resources, such as by tuning, provisioning, and/or workload throttling to adapt to the rise and decay of unexpected surges.

## 2 Architecture and Algorithms

The system is structured into three layers as shown in Figure 1. The Application layer provides the business function. We use a two-tier web application with one or more application servers and a database server. In general, we require that the

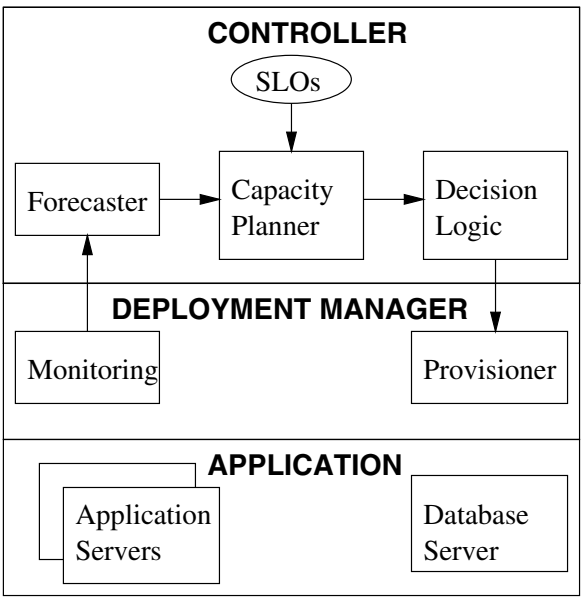



**Fig. 1.** Architectural layers for dynamic surge protection. The arrows show the control and data flow

application tier scale horizontally (i.e. resources can be added/removed without system shutdown).

The Deployment Manager provides a generic interface for monitoring and configuring the application layer. In this work, transaction rates and response times were monitored. Configuration management was focused on provisioning (addition/removal of application servers). The architecture assumes that for each resource type there is a provisioning function that manages a resource pool that can be shared among applications. Longer-term configuration management will also address configuration parameter (e.g., buffer pool sizes) and admission control adjustment.

The Controller monitors the application layer state and initiates appropriate actions if a SLO violation is anticipated or if the SLO can be satisfied in a more cost-effective way. Figure 1 also depicts the control and data flow for dynamic surge protection. Workload data from monitoring are input to the forecaster, which predicts future workload. The capacity planner takes as input the prediction and SLO to determine the resource requirements (e.g. number of servers). The decision logic manages the information flow and determines the resource adjustments (by comparing to the deployment state data). These adjustments are effected by the provisioner. Note that this flow is straight through (not iterative) since the required inputs are known at each step. The system operates based on six time intervals.

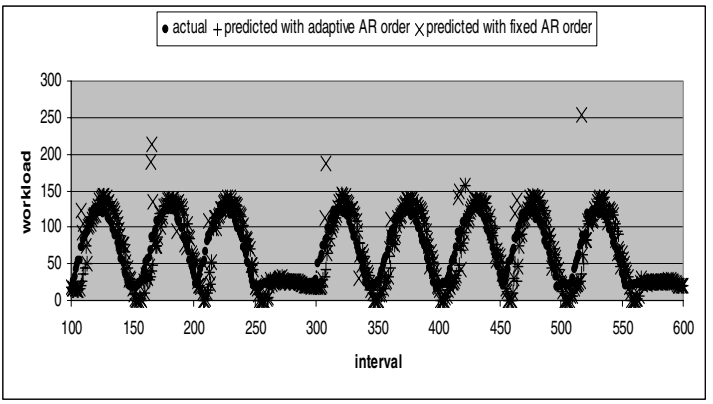



**Fig. 2.** Short term forecaster showing the comparison between an actual time series and 1 min (6 interval) ahead predicted values. Note that spurious predictions (e.g. high flyers) occasionally resulting from the fixed order model are greatly minimized with the adaptive order model

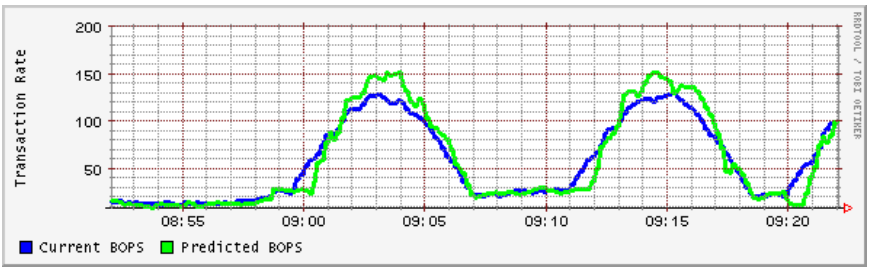



**Fig. 3.** Actual and 10000 min ahead predicted values of the workload

1. A measurement interval $M$ chosen based on overhead and noise vs responsiveness concerns.
2. The control interval $P$ ($P \geq M$), between executions of the control flow in Figure 1.
3. The lead time $S$, between initiating a resource action (e.g. add/remove a server) and its completion.
4. The prediction horizon $H$, should be $\geq S$. The trade-off is that a longer $H$ increases variability ($\sim \sqrt{H}$).
5. The overflow interval $O$, between situations when resources need to be adjusted to avoid SLO violations. The control engine can respond if $O \geq P+S$.
6. The underflow interval ($U$) between when a SLO can be met with fewer resources and when the extra are removed. The control engine can handle $U \geq P + S$.

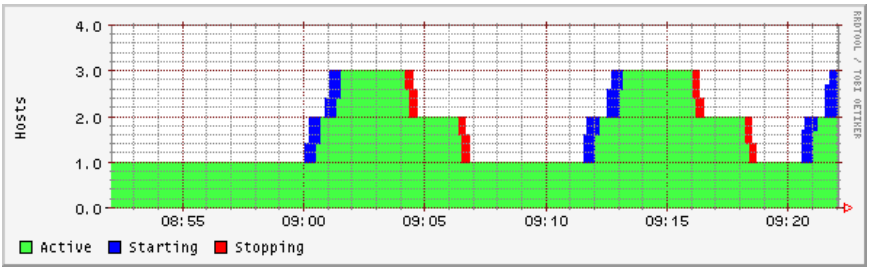



**Fig. 4.** State and number of application servers

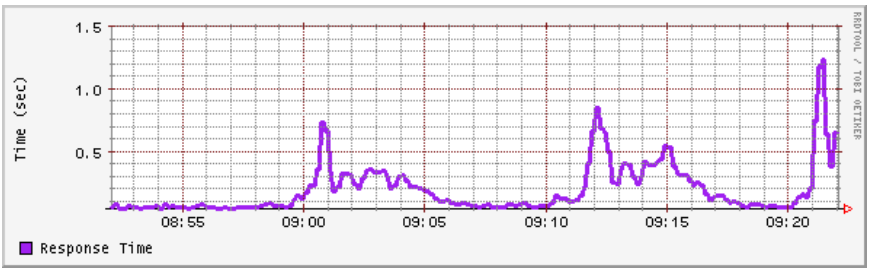



**Fig. 5.** Response time of application

We conclude that $H \sim P + S$ is reasonable. In our prototype (more details below) $S$ is $\sim 30$ sec, and we found that $M = P = 10$ sec worked well. Incorporating some "safety margin", we use $H = 60$ sec.

The capacity planner provides both performance estimates and the ability to determine resource requirements given the workload and the SLO. We used an IBM internal tool [5] that has been widely used in service engagements over the last two years. This tool uses analytic queueing approaches to estimate performance and capacity of a web deployment based on workload patterns (predefined or modeled by the user), performance objectives, together with hardware and software specifications. Other approaches to online capacity estimation are described in the literature [12].

We use a short term forecaster (or predictor) since our emphasis is on managing unexpected surges. The short term forecaster needs to provide useful information about the leading edge of the surge. Key to this is the ability to learn quickly, which is achieved by using short history data, and not relying on extensive training. In this work, the short term forecaster typically used 2 min (12 points) of history to predict 1 min into the future. The short term predictor is almost "memoryless" – it does not retain knowledge of past surges. This is important since the occurence of one unexpected surge (or busy period) is not assumed to provide information about when to expect the next surge. Analysis of Web traffic [4] suggests that busy periods are not strongly auto-correlated

with idle periods (time between busy periods). Within a busy period there can be reasonable auto-correlation [18], hence non-seasonal ARIMA (autoregressive, integrated, moving average) models [2] are effective.

An undesirable effect of using a short history for prediction is increased inaccuracy. Thus, we dynamically adjust the model order (the number of, AR, or autoregressive terms) based on the stability of the estimates. The latter is determined by checking if the poles of the transfer function lie within the unit circle in the complex plane, which is requirement for stability in discrete time systems. This stability criterion is very related to checking if the AR model is stationary (location of roots of the characteristic equation in relation to the unit circle) [8]. If the AR model with order $p$ is deemed unstable (note that order 0 model is stable), the model order is reduced and model coefficients are recalculated. The advantage of the adaptive model order is shown in Figure 2. While most of the predictions from a fixed AR order effectively track the actual time series, there are occasions when spurious predictions occur. These spurious predictions, which drive unnecessary control actions, are significantly minimized by adapting the model order.

We note that the short term forecasting we discuss above has also been used in conjunction with a long term forecaster [9] which is effective at capturing cyclic variation. Currently, the training data of the long term forecaster would include the unexpected surges since doing so simplifies data management. However, this approach can increase forecast variability and result in predicting phantom surges.

## 3 Results

Figure 3-Figure 5 show the results of experiments conducted on a research testbed. The testbed consists of a workload driver, multiple application servers running IBM's Websphere Application Server (WAS) v5.0, a single database server running IBM's DB2 v8.1, and a manager machine that incorporates code for the Controller and Deployment Manager. The provisioner leverages WAS 5.0's cellular cluster capability and uses its startServer/stopServer commands to add/remove servers. We note that both the Controller and Deployment Manager incur minimal CPU load.

The application that we deployed on our two tier system simulates the supply chain management of a manufacturing company. Briefly, the application processes injected order transactions and kicks off manufacturing transactions internally. The total transaction (business operations) rate is the sum of the actual order and manufacturing rates, and is normally (no transaction rollbacks) $\sim 1.75\times$ order injection rate. The workload driver can vary the order injection rate, where the inter-arrival time is exponentially distributed. Large workload surges (e.g. to mimic an influx of new users) are randomly triggered, and the Controller's SLO is to keep response time below 2 sec.

Figure 3 plots the actual (blue or darker line) and predicted 1 min into the future (lighter or green line) business operations per second (BOPS), the met-

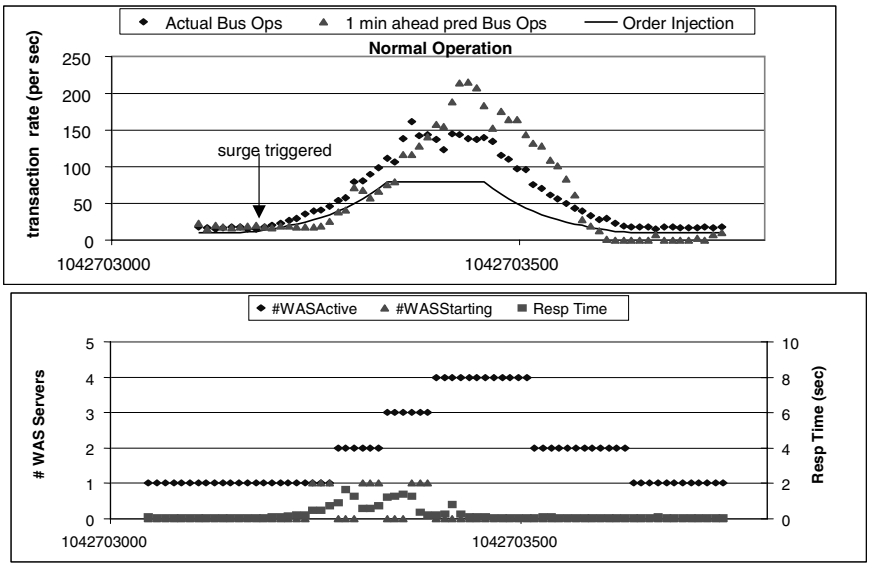



**Fig. 6.** Operation of system using dynamic surge protection

ric used to characterize workload. During the non-surge or normal periods (e.g. 8:52–8:59), BOPS has little variation. When a surge begins (e.g., 8:59, 9:11), BOPS increase rapidly to a peak of 120, ($\sim 6 \times$ normal). Not surprisingly, predicted BOPS for the first 60 sec are significantly below actual BOPS since the prediction is based on non-surge data. Within a few control intervals into the surge, the prediction accuracy improves considerably. Note that prediction accuracy during the surge (especially in regions of highest curvature) is not nearly as good as during the non-surge periods. It is important to emphasize that while the forecaster cannot predict the occurence of the surge, it can quickly recognize the workload trend change, and thus provide information on the anticipated evolution of the surge. Also, we reiterate that the forecaster essentially does not retain any knowledge about past surges since the history used for prediction ($\sim$ 2 min) is less than the surge duration.

Figure 4 shows the changes of the state and the number of application servers in response to the actual and predicted workload. When a rapid increase in load is detected just after 9:00, a server is added (the leading dark or blue section). This is about 40 sec before a server would have been added had the decision been based on the actual BBOPS vs predicted BOPS. A second server is added at 9:01 as the short-term forecaster anticipates the progression of the surge. As the surge subsides around 9:04, servers are released (the trailing red or less dark section). Note that while the decisions about adding resources is made quite aggressively, the removal of servers is more gradual, due to damping introduced by the Controller decision logic to minimize repeated add/remove server operations.

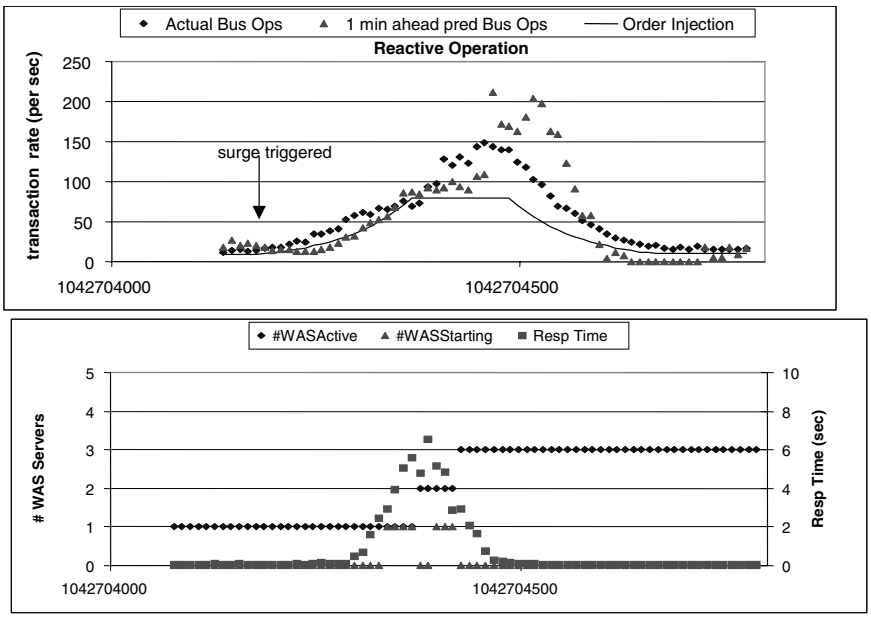



**Fig. 7.** Operation of system using a threshold-based heuristic

Figure 5 depicts the effect of these actions on response times. We see an initial bump in response time around 9:00. In part, this is due to the increased load that cannot be handled until the server has completed its startup phase. But there is also some delay introduced by the action of adding a server. Nonetheless, the SLO is not violated We note that because of the stochastics of the system, the response times and control actions responding the different surges are not identical.

Although in Figure 3 we show one type of surge, we have tested the system with different types of surges (e.g. multi-peaked, different shapes, etc). For instance the surge shown in Figure 6 has exponential rise/decay with a plateau in between vs the half sine wave surge in Figure 3). In an effort to test the responsiveness of the system, we experimented with surges (peak = 8 × base) that have different exponential rise rates (characterized by doubling time). While the observations are dependent on the specifics of the deployment, we find that the system can maintain the SLO when the doubling time is 60 sec, and even at 45 sec, but has trouble keeping up when it is 30 sec. We have also successfully tested it with larger surges (20 × base @ 60 sec doubling time, which is similar in relative size to the CNN surge, but at higher ramp rate [7]). We note that size of the surge that handled with our approach is bounded by the capacity of the database server. The online capacity planner used here is also capable of providing the appropriate sizing of the database tier. Dynamically scaling the database tier is, however, more complex than scaling the application tier.

In addition, to estimate the efficiency of our approach, we ran a 10 hour marathon with surges (about 60) initiated at random times (Figure 2 is a part of this run). We then calculate the optimal deployment, which we define as minimum number of servers required to handle the actual workload at any time (see [3]), based on a server capacity of 60 BOPS. The average optimal deployment over the run is 1.61 servers. Our approach used an average of 1.92 servers, which is only 20% more than optimal. Conversely, a static deployment that can handle all the surges would require 3 servers.

We also note that this approach has been successfully tested on several different deployments. The results in Figure 6 and Figure 7 are from different hardware deployments.

For comparison, we explore a threshold-based heuristic (i.e. no forecasting, no online capacity planning) as an alternative to dynamic surge protection. We use the following threshold-based heuristic:

> Increase number of servers by 1 if response time exceeds SLO and wait 60 sec (sufficient time for server to be taking load normally) before next control action is considered.

Figure 7 displays results from this threshold-based approach, which can be compared to the performance of the dynamic surge protection approach in Figure 6. It appears that the heuristic is set too high to accomodate the provisioning lead time. By the time the heuristic reaction takes effect, response times have already grown very large, and BOPS cannot keep up with the order injection due to transaction rollbacks. (normally BOPS $\sim$ 1.75 $\times$ order injection). Also, there needs to be a rule for releasing servers when they are no longer needed. Choosing a lower thresholds for when to add (high water mark) and remove (low water mark) servers could address these concerns. However, response time is often quite noisy. On another hardware deployment where we did some characterization of response time variability, we found that with 3 WAS servers and order injection rate = 70 (which was about the capacity of 2 servers) the response time ranged from 0.1–0.3 sec. However, on increasing the rate to 90 (still below capacity for 3 servers – average CPU idle $\sim$ 30%) the range increased significantly (0.2–0.9 sec). Choosing high and low (especially) water marks while trying to avoid the possibility of cycling servers in and out in this situation can be challenging. More robust high/low watermarks can be based on transaction rate, but that would involve some form of capacity estimation.

## 4 Conclusions

This paper describes dynamic surge protection, a technique for handling unexpected subscriber surges in systems that have resource actions with lead times (e.g. provisioning an application server). Dynamic surge protection incorporates three technologies. Short-term forecasting provides a way to anticipate the trajectory of workload demands of a surge. On-line capacity planning determines the

resources required to maintain a SLO based on the anticipated workload. Configuration management via provisioning (adding/removing application servers) is how the the onset and subsiding of unexpected surges is managed.

We have conducted a number of experiments on testbed systems to gain insight into the characteristics of dynamic surge protection. Overall, we have found it to be well behaved, and the performance compares favorably to a threshold-based heuristic. In addition, the approach is cost effective – in one extended test, we found that it uses only 20% more resources than a theoretical optimal deployment and 35% less resources than a static deployment.

Our future work will address handling of multiple workloads, and resource actions that include tuning and admission control. We also plan to leverage Grid services.

## References


[1] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger. Adaptive algorithms for managing a distributed data processing workload. *IBM Systems Journal*, 36(2), 1997. 83

[2] G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976. 87

[3] A. Chandra, P. Goyal, and P. Shenoy. Quantifying the benefits of resource multiplexing in on-demand data centers. *Proceedings of the First ACM Workshop on Algorithms and Architectures for Self-Managing Systems - to appear*, 2003. 83, 90

[4] M. E. Crovella. Performance characteristics of the world wide web. *Performance Evaluation, LNCS*, 1769, 2000. 86

[5] IBM. High volume web site performance simulator. *http://www7b.boulder.ibm.com/wsdd/library/techarticles/hvws/perfsimulator.html*, 2002. 86

[6] IBM. Autonomic computing. *http://www.ibm.com/autonomic*, 2003. 83

[7] Bill Lefebvre. Facing a world crisis. *USENIX LISA*, 2001. 82, 89

[8] G. G. Judge R. C. Hill W. E. Griffiths H. Lutkepohl and T. C. Lee. *Introduction to the Theory and Practice of Econometrics: Second Edition*. Wiley, 1988. 87

[9] L. W. Russell S. P. Morgan and E. G. Chron. Clockwork: A new movement in autonomic systems. *IBM Systems Journal*, 42(1), 2003. 83, 87

[10] Kathleen Ohlson. Victoria's secret knows ads, not the web. *Computer World*, February 1999. 83

[11] Hewlett Packard. HP utility data center. *http://www.hp.com/go/hpudc*, 2003. 83

[12] M. Goldszmidt D. Palma and B. Sabata. On the quantification of e-business capacity. *Proceedings of the 3rd ACM conference on Electronic Commerce*, 2001. 86

[13] K. Appleby S. Fakhouri L. Fong M. K. G. Goldszmidt S. Krishnakumar D. Pazel J. Pershing and B. Rochwerger. Oceano–SLA-based management of a computing utility. *Proceedings of the IFIP/IEEE Symposium on Integrated Network Management*, 2001. 83

[14] ProvisionSoft. ProvisionSoft Home Page. *http://www.provisionsoft.com*, 2003. 83

[15] Sun Micro Systems. Sun N1. *http://wwws.sun.com/software/solutions/n1*, 2003. 83

[16] ThinkDynamics. Thinkdynamics Home Page. *http://www.thinkdynamics.com*, 2003. 83
[17] Zeus. Why web technology is vital to your business. *Computer World*, 2003. 83
[18] M. S. Squillante L. Zhang and D. Y. Yao. Web traffic modeling and web server performance analysis. *Proceedings of the 38th IEEE Conference on Decision and Control*, 5, 1999. 87
[19] J. Rolia X. Zhu and M. Arlitt. Resource access management for a utility hosting enterprise applications. *Integrated Network Management VIII*, 2003. 83

# A Method on Multimedia Service Traffic Monitoring and Analysis*

Hun-Jeong Kang, Myung-Sup Kim, and James Won-Ki Hong

Department of Computer Science and Engineering
POSTECH, Korea
{bluewind,mount,jwkhong}@postech.ac.kr

**Abstract**. The use of multimedia service applications is growing rapidly on the Internet. These applications are generating a huge volume of network traffic, which has a great impact on network performance and planning. For various purposes, obtaining information on multimedia service traffic is important. However, traditional analysis methods based on well-known ports cannot be used to analyze such traffic. Because the majority of multimedia service applications use dynamically allocated port numbers, the traditional methods misidentify multimedia service traffic as unknown traffic. This paper presents a method for monitoring and analyzing multimedia service traffic. Our method detects transport protocol and port numbers for dynamically created sessions during a control session. We then use such information to analyze traffic generated by the most popular multimedia service applications, namely Windows Media, RealMedia, Quicktime, SIP and H.323. We also present a system architecture that uses our method to monitor and analyze multimedia service traffic.

## 1 Introduction

The use of streaming media and multimedia conferencing applications is growing rapidly. Many Internet sites provide various rich media content of broadcast, movies, and music. We call the network traffic generated by the streaming media and multimedia conferencing applications as multimedia service traffic. This multimedia service traffic is becoming increasingly dominant in IP networks and is affecting the network performance and planning. Therefore, it is important to monitor and analyze multimedia service traffic for acquiring information about the network usage.

However, most existing traffic monitoring systems cannot be used to analyze multimedia service traffic. These systems use well-known port numbers for identifying applications [1]. Most multimedia service applications make use of port

* This work was in part supported by the Electrical and Computer Engineering Division at POSTECH under the BK21 program of Ministry of Education and HY-SDR Research Center at Hanyang University under the ITRC program of Ministry of Information and Communication, Korea.

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 93-105, 2003.

numbers that are not well-known but are dynamically allocated during set up sessions. As a result, traffic to transfer multimedia service data is misidentified as unknown traffic in these systems [2, 3].

This paper presents a method and a system architecture to monitor and analyze multimedia service traffic. We have developed a dynamic session analyzer which parses control protocols. This analyzer processes the payload of a packet associated with a control protocol and extracts information such as transport protocol and port numbers used for transferring multimedia service data. We then use this information to analyze popular multimedia service traffic, namely Windows Media [4], RealMedia [5], Quicktime [6], SIP [7], and H.323 [8] traffic.

This paper is organized as follows. Section 2 provides an overview of several popular multimedia service protocols. Section 3 discusses related work on traffic monitoring and analysis. Section 4 presents our analysis method for multimedia service traffic. Section 5 describes the architecture of our multimedia service traffic monitoring and analysis. Finally, Section 6 summarizes our work and discusses possible future work.

## 2 Overview of Multimedia Service Protocols

This section describes backgrounds of multimedia service protocols and their characteristics. In streaming media service, we are aiming to most popular services: Windows Media Technology (WMT), RealMedia, and QuickTime. These services differ in their protocols, as illustrated in Table 1. In Internet multimedia service conferencing services, most applications are based on SIP (Session Initiation Protocol) or H.323. Table 2 describes protocols used in these applications.

**Table 1.** Streaming Media Service Protocols

| streaming media service | control session protocol | data session protocol |
|---|---|---|
| RealMedia | RTSP | RDT |
| QuickTime | RTSP | RTP |
| WMT | MMS | MMST/MMSU |

**Table 2.** Multimedia Conferencing Protocols

| application | control session protocol | data session protocol |
|---|---|---|
| based on SIP | SIP | RTP |
| based on H.323 | Q.931, H.245 | RTP |

During a multimedia service, two types of sessions are created between a client and a server: a **control session** and a **data session**. The control session is responsible for setting up connection and controlling navigation, such as play and pause. This session uses control protocols such as RTSP (Real Time Streaming Protocol) [9] and MMS (Microsoft Media Server) [4]. The data session sends the multimedia service contents to the client over the data session protocol, including RDT (RealNetworks Data Transfer) [5], RTP (Realtime Transfer Protocol) [10], and MMST/MMSU (MMS

over TCP/UDP) [4]. We designate each packet related to the control session and data session as a control packet and a data packet, respectively.

Figure 1 illustrates a client/server interaction for a control and data transfer session. To begin, a control session is set up through a well-known port number. As described in Figure 1 (a), streaming media services (e.g., RealMedia, QuickTime) or applications based on SIP have one control session. On the other hand, H.323 applications have two control sessions: Q.931 [8] and H.245 [8] sessions. A control session creates a new data session by negotiating a transport protocol and port numbers. Then the data session transfers multimedia data through the dynamically assigned transport protocol and port numbers. In this paper, we introduce a new term, **dynamic session,** that makes use of the transport protocol and port numbers that are dynamically negotiated by the control session, such as the data session and second control session in Figure 1 (b).

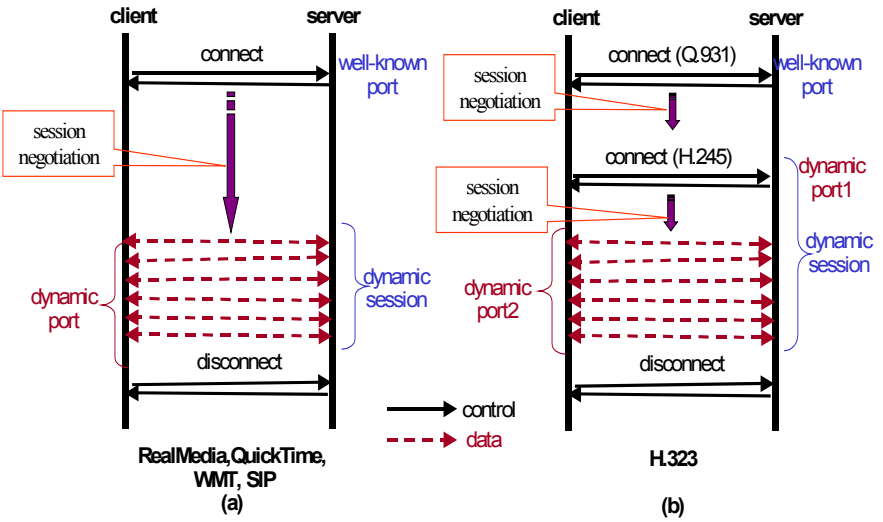



**Fig. 1.** Multimedia Service Control and Data Session

When a control session negotiates about a dynamic session, the packet payload of the control session contains negotiation results such as a transport protocol and port numbers used in the dynamic session. By selecting and analyzing the control packet, we can discover information about the dynamical session, which is called the **dynamic session information** in this paper.

The use of dynamic sessions in multimedia services causes disadvantages in traffic monitoring, although the use benefits in delivering data. These services can send multimedia data efficiently by changing appropriate protocols for streaming and conferencing. On the other hand, new and not well-known port numbers appear after the session negotiation. Because of these unknown port numbers, the traffic used by dynamic session is misidentified as unknown traffic by most traffic monitoring systems that use well-known port numbers for identifying applications. That is the reason why we find dynamic session information and use it when determining multimedia service traffic.

## 3 Related Work

A flow represents a series of packets traveling between ‘‘interesting" end points. There are various definitions about the flow [13, 14, 15]. In this paper, we define a flow as a sequence of packets with the same 5-tuple: source IP address, destination IP address, source port, destination port, and protocol number. By aggregating related packets into a flow, one can reduce system overhead to process data. Due to this compressibility, many systems, such as NG-MON [16], analyze traffic based on flows.

**Flowscan** [11] is also a flow-based traffic analysis system. Its monitoring target related to multimedia service is the traffic using RTSP. It uses a heuristic method as follows. The system records ongoing control sessions. When a flow is seen with an unknown port number on two hosts, it checks to verify whether an active control connection exists between the same hosts. If so, it assumes that the flow corresponds to a dynamic session. However, this analysis may provide inaccurate information. The reason is that traffic seen with an unknown port number may not be related to the active control connection that exists between two connected hosts. Further, some multimedia service data can be transferred from another source that does not participate in the active control connection. In this case, this heuristic method misidentifies the multimedia service traffic as unknown traffic, because no active control connection exists between these hosts that transfer multimedia service data.

**mmdump** [3] is a tool for monitoring multimedia traffic on the Internet. This tool is used to investigate the characteristics of multimedia service traffic over RTSP and H.232. The tool contains a parsing module for the RTSP and H.323 protocol. It parses the control messages to extract the dynamically assigned port numbers. The parsing module then dynamically changes a packet filter to allow packets associated with these ports to be captured. By changing the packet filter, this tool can capture only packets that contain listed port numbers, while reducing the resource requirements and capture overhead. However, it is also a burden to frequently compile and change the packet filter. In addition, this tool reveals the following problems. First, it does not analyze MMS [4] that is considered to be the most widely used streaming service in the world. Next, it does not consider IP-fragmentation. We observed that about 40~70% of WMT packets are fragmented during our tests. Similarly, some applications send large streams into the network, and these data are fragmented. The port number of these fragmented packets cannot be identified without reassembly. Because mmdump captures a packet by referencing only port numbers, it misses fragmented packets, even though they are multimedia service packets. Further, it may commit a false-rejecting error, where the real data packet is misidentified as not associated with the multimedia service session. Consider a streaming data packet that belongs to a data session but is not contained in the packet filter to be captured. Some data packets pass the probing point after deletion of port numbers from the filtering list. Then the packet may pass the probing point without being captured. In these cases, the analysis results of mmdump are not accurate.

# 4 An Analysis Method for Multimedia Service Traffic

## 4.1 Analysis Procedure

In this section, we present our proposed method for analyzing multimedia service traffic. Figure 2 is a flowchart to illustrate packets being captured and processed. The overall procedure consists of three major parts: flow generation, dynamic session analysis, and traffic analysis.

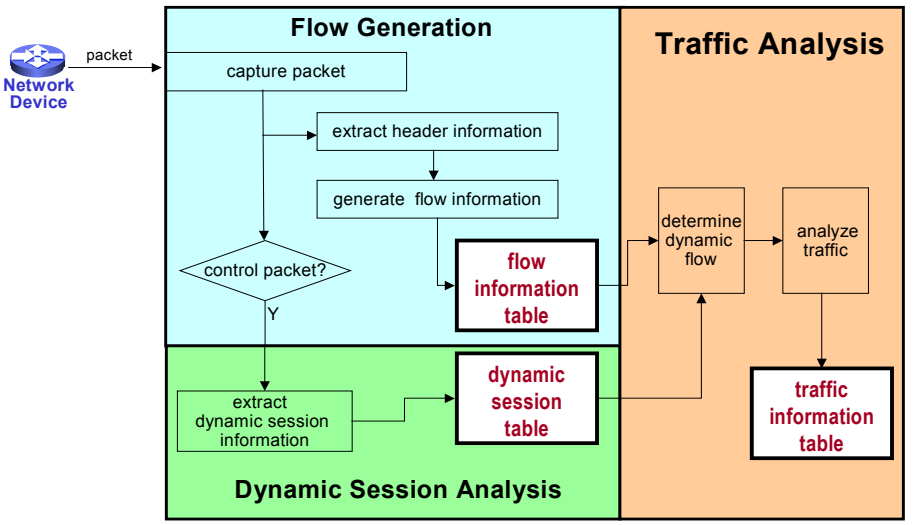



**Fig. 2.** Flowchart for Multimedia Service Traffic Monitoring and Analysis

The flow generation part captures packets and analyzes their header. By collecting and aggregating related packets, this part generates flow information. Based on this flow information, the traffic analyzer generates various traffic information into the traffic information table. However, it is insufficient to identify dynamic session traffic with only a port number. The reason is that dynamic sessions do not use well-known ports. Therefore, we need dynamic session information to decide whether or not a flow with an unknown port number is related to multimedia service traffic. This information can be extracted in the dynamic session analysis part. When a packet is analyzed by the flow generation part, the control packet is sent to the dynamic session analysis part. Next, the packet is analyzed to determine whether or not it contains dynamic session information. In the following sections, we describe the dynamic session analysis part in detail.

## 4.2 Dynamic Session Analysis

Figure 3 describes our algorithm to discover dynamic session information from the control packet. The dynamic session analyzer receives a control message, including packet header information, and a payload of the transport layer. First, the procedure determines if the FIN flag is set to identify it as a session disconnect request. If not,

the module analyzes whether the packet contains dynamic session information. We can reduce the analysis overhead by selecting a packet, which is likely to contain dynamic session information. The payload of the selected packet is parsed according to each control protocol (line 6, 9, 11, 14, or 16). After parsing, the procedure confirms if the dynamic session information is discovered (line 17). If so, this information is stored into the dynamic session table that contains information on active dynamic sessions (line 19).

```
1  Procedure DynamicSessionAnalyzer ( Msg )
2   BEGIN
3    if   FIN Flag in Msg is NOT set
4      then if protocol in Msg = RTSP
5         then if SourcePort in Msg = RTSP server port number
6            then result = ParseRTSP (payload of Msg ) ;
7         else if protocol in Msg = MMS
8            then if DestinationPort in Msg = MMS server port number
9               then result = ParseMMS (payload of Msg ) ;
10           else if protocol in Msg =SIP
11              then result = ParseSIP (payload of Msg ) ;
12              else if protocol in Msg = Q.931
13                 then if SourcePort in Msg = Q.931 receiver port
14                    then result = ParseQ931 (payload of Msg ) ;
15                 else if protocol in Msg = H.245
16                    then result = ParseH245 (payload of Msg ) ;
17     if result= TRUE then
18      create new dynamic session information;
19      insert dynamic session information into dynamic session table;
20   else
21    delete session information from dynamic session table;
22  END
```

**Fig. 3.** Multimedia service Traffic Analysis Algorithm

When a multimedia service is completed, information on the dynamic session must be removed. This information is usually deleted from the dynamic session table (line 21) when the TCP FIN flag is set to disconnect the control session. However, the FIN packet may never be captured because of such effects as packet losses or route changes [3]. In such cases, the information is removed from the table by selecting a session, which shows no activity for a certain period of time. Consequently, the traffic analysis module can identify the application of dynamic flows by referencing the dynamic session table.

### 4.2.1 Analysis of RTSP

RealMedia and QuikTime applications use RTSP as control protocol. Figure 4 (a) illustrates messages of RTSP during negotiation of a dynamic session. A client sends a SETUP request to a server, along with the candidates for a data transfer protocol and port number (or a range of port numbers) to be used for receiving multimedia service data. Next, the SETUP response contains the protocol and port numbers

chosen by the server. Accordingly, the procedure ascertains whether the source port of the packet is an RTSP server port (i.e., 554) (line 5 in Figure 3), and whether the packet from the server contains 'RTSP RESPONSE Expression' in Figure 4 (b). Then, the procedure parses the payload and searches for 'TRANSPORT Expression': "Transport:", ";client_port=", the number or range of numbers, and ";".

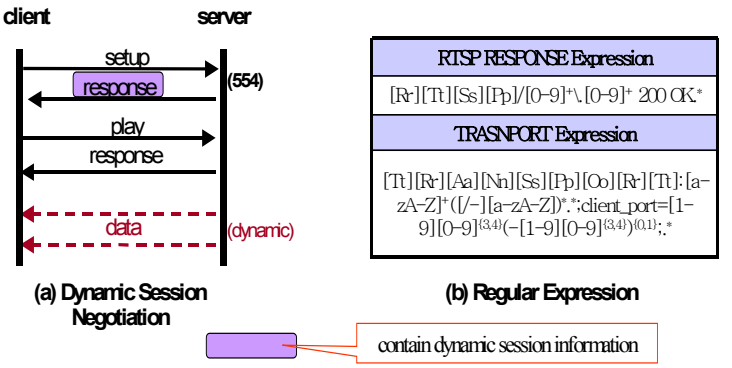



**Fig. 4.** Dynamic Session Construction in RTSP

### 4.2.2 Analysis of MMS

MMS is a control protocol of Windows Media Technology (WMT). Although its specification is not publicly open, we have discovered by observing and analyzing packets that the client's request in MMS contains the transport protocol and port numbers used for transferring multimedia service data. Therefore, the destination port number is checked to verify that it is the MMS server port number (i.e., 1755) (line 8 in Figure 3) for the purpose of choosing the client request packet. Among the client request packets, the only SETUP packet, named for convenience in this paper, contains dynamic session information. Accordingly, the procedure verifies the client request packet contains 'SETUP Expression' as illustrated in Figure 5 (a). Even though we have not ascertained the specification of MMS, we can analyze by searching for 'TRANSPORT Expression': string of "MMS", 'URL-string format', "TCP" or "UDP," and the port number.

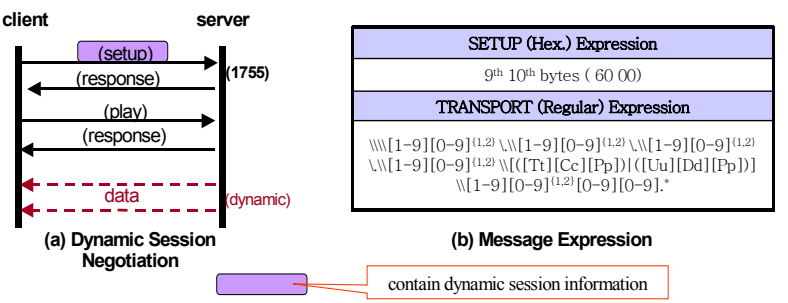



**Fig. 5.** Dynamic Session Construction in MMS

### 4.2.3 Analysis of SIP

Figure 6 (a) illustrates messages of SIP during negotiation of a dynamic session. A client sends an INVITE request to a server, along with the port number that is used to for the client to receive multimedia service data. Then the server sends the RESPONSE packet that contains port number through which the server receives data from the client. For this reason, the procedure selects packets with 5060, SIP server port (line 10 in Figure 3). Then, it verifies if they are invite or response message by matching the payload of the selected packet with 'INVITE Expression' or 'RESPONSE Expression' in Figure 6 (b). After selecting, the procedure extracts dynamic session information from a SDP (Session Description Protocol) [15] part of the payload. It finds 'MEDIA Expression', which consists of components, such as "M=", media type, port number, transport protocol, and payload type.

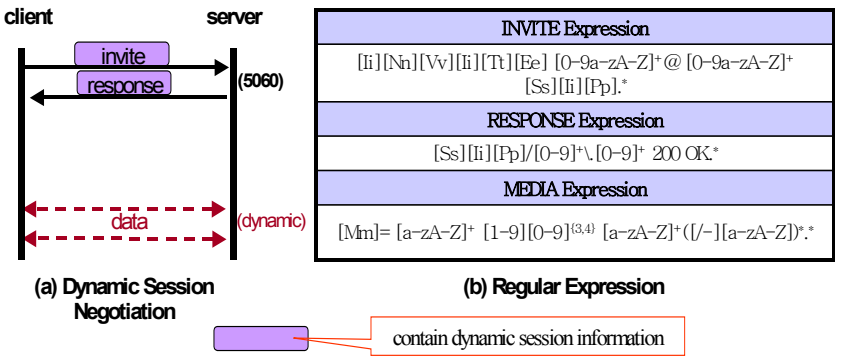



**Fig. 6.** Dynamic Session Construction in SIP

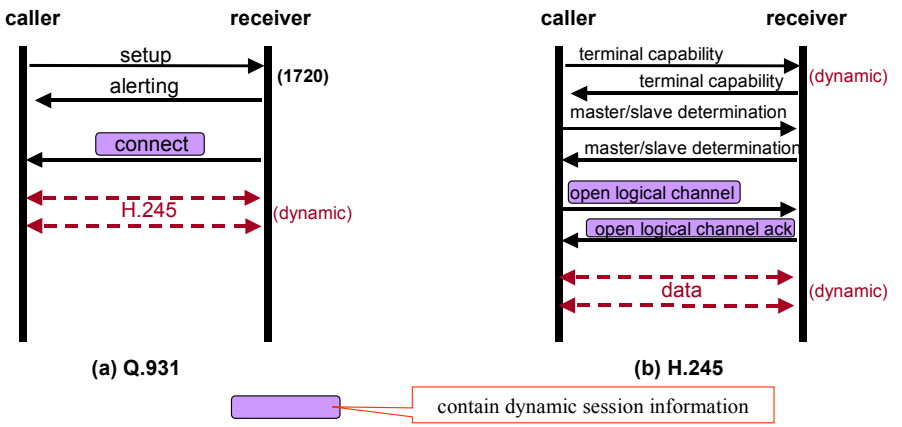



**Fig. 7.** Dynamic Session Construction in H.323

### 4.2.4 Analysis of H.323

Services based on H.323 have two dynamic sessions, as illustrated in Figure 7 (a) and (b). First, Q.931 uses a well-known receiver port, 1720. Because the information about H.245 is contained in the connect message, the procedure checks if the source port is the receiver port (line 13 in Figure 3) and if it is a connect message. In the case of H.245, this session uses a port number that is dynamically allocated by a Q.931 session. Accordingly, we need to determine whether a captured packet is related to a H.245 session by matching dynamic session information that generated by the Q.931 session (line 15 in Figure 3). Then the procedure selects an open logical channel or open logical channel ack packet in the H.245 session.

Contrary to the above text-based protocols of RTSP, MMS, and SIP, the procedure searches for the locations of port number in Q.931 and H.245 packets. In Q.931, the dynamic session analyzer extracts a dynamically assigned port number from a port in an User-User info Element information. It can discover the port number of H.245 in tsap Identifier of a forwardLogicalChannel, reverseLogical Channel, or network access parameter.

## 5 Architecture for Multimedia Traffic Monitoring and Analysis

We have developed a system for monitoring and analyzing multimedia service traffic. We have adopted the system architecture of **NG-MON** [16] and integrated the proposed method with NG-MON. As illustrated in Figure 8, traffic monitoring and analysis tasks are divided into several phases, which are serially interconnected using a pipelined architecture. One or more systems may be used in each phase to distribute and balance the processing load. Each phase performs its defined role in the manner of a pipelined system. This architecture can improve the overall performance and scalability, with each phase configured with a cluster architecture for load distribution. We have also defined a communication method between each pair of phases. Each phase can be replaced with more optimized modules as long as they provide and use the same interfaces. The divided architecture provides flexibility. By assigning tasks to each phase, this architecture enables us to easily append or remove modules for added work such as dynamic session analysis.

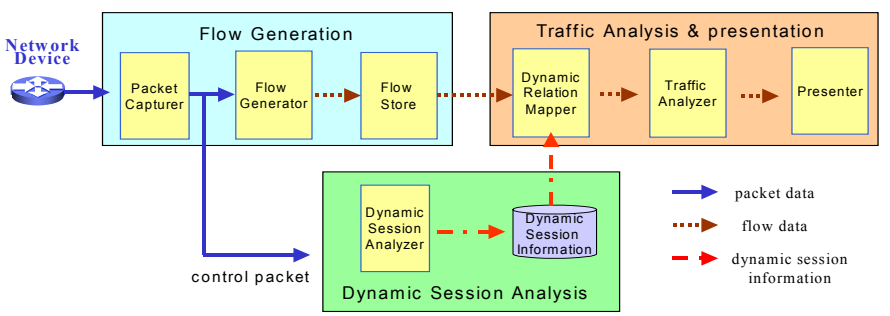



**Fig. 8.** Multimedia Service Traffic Monitoring Architecture

### 5.1 Flow Generation

The flow generation module consists of a packet capturer, a flow generator, and a flow store. The packet capturer collects packets passing a probing point. Another function of the packet capturer is to extract information from the packet header and to send it to the flow generator. The format of the packet header information is also shown in Figure 9. The time stamp represents the time when the packet is captured.

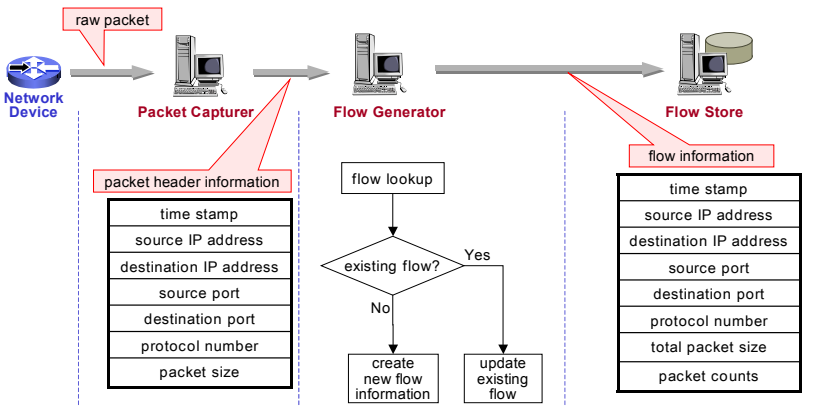



**Fig. 9.** Flow Generation Module

The flow generator creates a flow by collecting a series of packets. Figure 9 illustrates the function and parameters of the flow generator. Whenever receiving packet header information, the flow generator looks up the flow table to search for an existing flow to which the packet belongs. If a matched flow exists, the packet is added to the flow by updating the flow information by increasing the count and total size of the packet. If not, a new flow is constructed from the packet header information. Next, the flow is inserted into the flow table. Flows in the table are periodically stored into a database. Here, the period can be configured according to the flow time-out in order to aggregate flow information during the predetermined time, such as one minute.

### 5.2 Dynamic Session Analysis

The dynamic session analysis module provides information for identifying multimedia service traffic. If a packet is determined to be a control packet, the packet capturer sends the packet to the dynamic session analyzer. By using the algorithm for streaming analysis described in Figure 3, the dynamic session analyzer discovers the information on the dynamic session. This information is stored into the dynamic session table and referenced by the relation mapper in the traffic analysis module.

| dynamic IP address | dynamic port | transport protocol | |
|---|---|---|---|
| Control client address | control client port | control server address | control server port |
| session start time | | session end time | |

**Fig. 10.** Dynamic Session Information

Figure 10 shows the format of the dynamic session information. In this format, the control server address and control server port are IP addresses and the port number of the server in the control session that created the dynamic session. Similarly, the control client address and control client port are the IP address and the port number of client in the control session. By making use of this information, the system is aware of the relationship between the control and dynamic sessions. The session start time is the time when the dynamic session information is newly created, and the session end time is the time when the control session is disconnected. The session end time is set either when a TCP FIN flag of a control packet is set, or when no packet in the same session is captured during a predetermined threshold time. These time fields are used to determine whether a dynamic session is active or inactive.

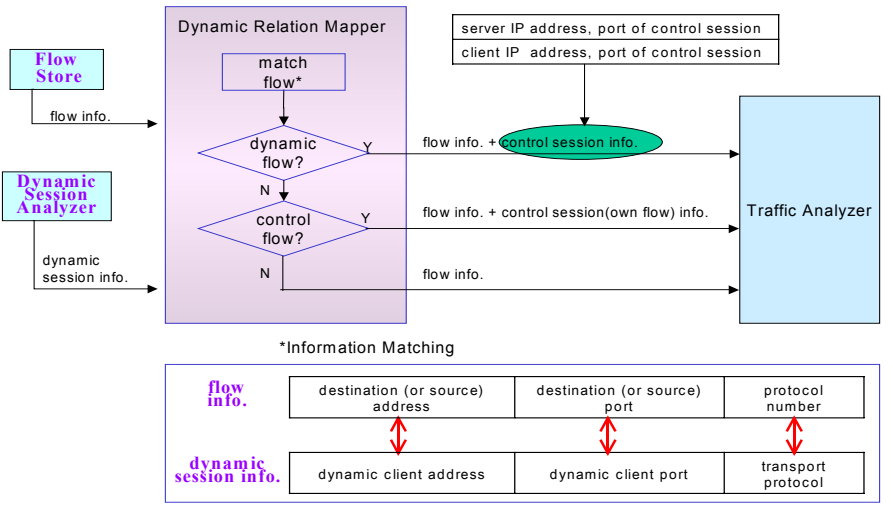



**Fig. 11.** Dynamic Relation Mapping

## 5.3 Traffic Analysis

The dynamic relation mapper decides the relation between a dynamic flow and a control flow. This module identifies whether a flow with unknown port number is related to a dynamic session. As illustrated in Figure 11, this module matches flow information with dynamic session information. Tuples to be compared are as follows: destination (or source) IP address and dynamic client address, destination (or source) port and dynamic client port, and protocol number and transport protocol. If the compared tuples are equal, some fields are added to the flow information, such as the IP address and port number of the control session. By adding these fields, we can map the dynamic flow and the control flow that creates the dynamic flow. In the case of control flows, the control server and client information are filled up with its own IP address and port number. Otherwise, only flow information is sent to the traffic analyzer without an addition of fields.

The traffic analyzer performs an analysis of traffic by querying the flow data stored in the database. It can analyze multimedia service traffic at the session level. It is possible for a multimedia service to open several sessions. The traffic analyzer can discover and analyze sessions separately. In addition, it integrates the information of sessions which belong to the same multimedia service. For example, it can analyze the traffic volume exchanged in the control and data sessions related to the same multimedia service.

## 6 Conclusion and Future Work

In this paper, we presented a method and system architecture for monitoring and analyzing multimedia service traffic. This method analyzes control protocol messages and extracts information on dynamic sessions. The extracted information includes dynamically selected protocol and port numbers, which are used to determine whether or not the unknown traffic is multimedia traffic. This approach makes it practical to monitor previously unknown multimedia service traffic, as well as other services.

This method boosts the analysis of traffic from the packet level to the session level. It does not simply extract header information of a packet, but makes it possible to analyze traffic per session by acquiring session information. In addition, it overcomes the problems with existing approaches that use only well-known port numbers of TCP or UDP for identifying the application of traffic. By analyzing application messages, this method discovers the status of the application and raises the analysis application level.

We are currently integrating our multimedia service traffic analysis method with NG-MON. We plan to analyze the multimedia service traffic on our campus network and then use our system to monitor and analyze ISP networks in Korea. We are planning to extend the proposed analysis method to other types of traffic that creates and use dynamic sessions.

## References

[1] Internet Assigned Numbers Authority, http://www.iana.org/

[2] James W. Hong, Soon-Sun Kwon and Jae-Young Kim, “WebTrafMon: Web-based Internet/Intranet Network Traffic Monitoring and Analysis System,” Computer Communications, Elsevier Science, Vol. 22, No. 14, Sept. 1999, pp. 1333-1342.

[3] Jacobus van der Merwe, *et. al*., “mmdump - A Tool for Monitoring Internet Multimedia Traffic,” ACM Computer Communication Review, 30(4), October 2000.

[4] Microsoft, WMT, http://www.microsoft.com/windows/windowsmedia/default.asp

[5] Real Networks, Real Media Technology, http://www.realnetworks.com/

[6] Apple, QuickTime, http://www.apple.com/quicktime

[7] M. Handley, H. Schulzrinne, E.Schooler, J. Rosenberg, "SIP: Session Initiation Protocol," RFC 2543, March 1999.
[8] ITU-T, "Recommendation H.323: Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-guaranteed Quality of Service," 1996.
[9] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," RFC 2336, April 1998.
[10] H. Schulzrinne, S. Casner, R. Frederick, V, and Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC1889, January 1996.
[11] Dave Plonka, FlowScan, http://net.doit.wisc.edu/~plonka/FlowScan/
[12] Siegfried Lifler, "Using Flows for Analysis and Measurement of Internet Traffic," Diploma Thesis, University of Stuttgart, 1997.
[13] J.Quittek, T. Zseby, B. Claise, K.C. Norsth, "IPFIX Requirements," Internet Draft, http://norseth.org/ietf/ipfix/draft-ietf-ipfix-architecture-00.txt
[14] CAIDA, "Preliminary Measurement Spec for Internet Routers," http://www.caida.org/tools/measurement/measurementspec/
[15] M. Handley, V. Jacobson, "SDP: Session Description Protocol," RFC 2327, April 1998.
[16] S. H. Han, M. S. Kim, H. T. Ju and J. W. Hong, "The Architecture of NG-MON: A Passive Network Monitoring System", DSOM 2002, Montreal, Oct., 2002, pp. 16-27.

# Traffic Measurements for Link Dimensioning
## A Case Study

Remco van de Meent, Aiko Pras, Michel Mandjes,
Hans van den Berg, and Lambert Nieuwenhuis

University of Twente
PO Box 217, 7500 AE Enschede, The Netherlands
r.vandemeent@utwente.nl

**Abstract.** Traditional traffic measurements meter throughput on time scales in the order of 5 minutes, e.g., using the Multi Router Traffic Grapher (MRTG) tool. The time scale on which users and machines perceive Quality of Service (QoS) is, obviously, orders of magnitudes smaller. One of many possible reasons for degradation of the perceived quality, is congestion on links along the path network packets traverse. In order to prevent quality degradation due to congestion, network links have to be dimensioned in such a way that they appropriately cater for traffic bursts on time scales similarly small to the time scale that determines perceived QoS. It is well-known that variability of link load on small time scales (e.g., 10 milliseconds) is larger than on large time scales (e.g., 5 minutes). Few quantitative figures are known, however, about the magnitude of the differences between fine and coarse-grained measurements. *The novel aspect of this paper is that it quantifies the differences in measured link load on small and large time scales.* The paper describes two case studies. One of the surprising results is that, even for a network with 2000 users, the difference between short-term and long-term average load can be more than 100%. This leads to the conclusion that, in order to prevent congestion, it may not be sufficient to use the 5 minute MRTG maximum and add a small safety margin.

## 1 Introduction

Some believe that QoS in networks will be delivered by the use of technologies such as DiffServ and IntServ, which ensure the "right" allocation of available resources among different requests. Concerns about deployment, operational complexity and, in the case of IntServ, scalability, give rise to alternative approaches of providing QoS. One of these alternatives is overprovisioning [1]. The idea behind overprovisioning is to allocate so many resources that users no longer experience a QoS improvement in case additional resources get allocated. This paper focuses on the bandwidth overprovisioning of individual network links. An example of such link is the access line between an organization's internal network and its Internet Service Provider.

Since bandwidth may be expensive, managers who rely on overprovisioning as mechanism for delivering QoS need to know the amount of traffic that users of

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 106–117, 2003.

a specific link may generate at peak times. To find this figure, managers usually use tools like `MRTG` [2]. Such tools are able to measure the average load of network links by reading the Interface Group MIB counters every 5 minutes (by default), and plot the results in a graph. The peaks in these graphs, plus a certain safety margin, or often used to dimension the specific link.

In cases where overprovisioning is used as mechanism to provide QoS, load averages of 5-minutes may not be adequate to properly dimension network links. With web browsing, for example, traffic is exchanged in bursts which last between parts of a second and several seconds. If, within these seconds, the link gets congested, the user will not perceive an acceptable QoS. Also distributed computer programs, which interact without human intervention, "perceive" QoS on time scales smaller then seconds. The traditional 5 minute figures of `MRTG` do not give any insight in what happens on these small time scales. It is therefore important to increase of the *time-granularity* of the measurements, i.e., to decrease the size of the time-window that is used to determine the link load. To overprovision, load figures are needed on the basis of seconds, or even less.

### 1.1 Contribution

The goal of this paper is to quantify the differences in measured link load at various time scales. For this purpose, two case studies have been performed. In the first study traffic was measured on the external link of a university's residential network; this link was used by thousands of students. Because of this high number, we expected that the differences between the long- and short-term averages would be relatively small. To get some stronger differences, the second study was performed on the access line of a small hosting provider, which served some tens of customers.

The outcome of the measurements came as a surprise. On the university's link, with thousands of users, the differences between long- and short-term averages could be more than hundred percent. In the case of the hosting provider, the differences could even be thousands of percents.

### 1.2 Organization

The remainder of this paper is organized as follows. Section 2 describes the two networks on which the measurements were performed, the measurement equipment that was used, as well as the way this equipment was connected. Section 3 discusses some factors that influence the achievable granularity of our measurements. Section 4 discusses the tools that were used for processing, analyzing and visualizing the gathered data. Section 5 presents the analysis of the measurements and determines the ratio of peak versus average throughput on the links of our case studies. The conclusions are provided in Section 6.

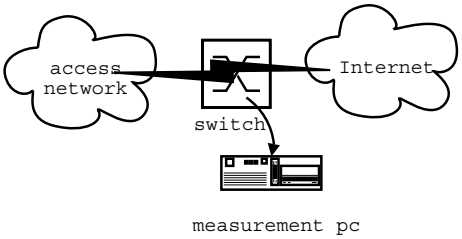



**Fig. 1.** Measurement Setup

## 2 Measurement Setup

In this study we have concentrated on network technologies that are common in both production and academic environments. We assume such networks to consist of (normal, Fast and Gigabit) Ethernet links, which are connected by hubs, switches and routers. In general it is important that normal network operation should not be disturbed by the measurements. For example, it may not be acceptable to interrupt network operation to install optical splitters that copy all network traffic to a measuring device. It will often be better to rely on the "mirror" facilities provided by current middle- to high-end switches, and copy all traffic that should be monitored to one of the free interfaces of that switch. The measurement device can than be connected to that interface, and all traffic can be analyzed without disturbing the network. See Figure 1 for a schematic representation of the measurement setup.

To validate whether it is practically feasible to measure throughput on small time scales, and to compare traditional `MRTG` measurements to throughput measurements with fine granularity, two case studies have been performed.

The first case study measures throughput on a backbone link that is used by thousands of users. With this large number of users it is expected that the traffic generated by a single user will have limited impact on the traffic aggregate; it is therefore expected that the difference in "long-term" and "short-term" throughput averages will be relatively low. The second case study measures throughput on a link that is used by some tens of users. With this small number of users it is anticipated that the results will differ much more.

### 2.1 Campus Network

The Campusnet [3], a residential network of the University of Twente, connects about 2000 students to the Internet. Each student has a 100 Mbit/s full duplex connection to the network. The network is hierarchically structured, and has a full-duplex 300 Mbit/s backbone link to the rest of the world. This link, with a 5-minute average load of about 50%, has been monitored in the first study. The question whether or not this link is overprovisioned, is hard to answer beforehand, given the expected variability of the throughput on smaller time scales.

**Table 1.** Measurement PC Configuration

| Component | Specification |
|---|---|
| CPU | Pentium-III 1 GHz |
| Mainboard | Asus CUR-DLS (64 bit 66 MHz PCI) |
| Hard disk | 60 + 160 Gigabyte, UDMA/66 |
| Operating system | Debian Linux, 2.4.19-rc1 kernel |
| Network interface | 1 x Gbit/s Intel Pro/1000T |
| Main memory | 512 MB reg. SDRAM |

### 2.2 Hosting Provider Network

The second case study involves the network of a small hosting company in The Netherlands. This network connects some tens of customers over a switched 100 Mbit/s network to the Internet. The link to the Internet, with a 5-minute average load of 5–10%, has been monitored in the second study. As this link is only mildly loaded, we anticipate that it can be regarded as being overprovisioned.

### 2.3 Measurement Device

An important requirement for this study is to use as much as possible common hardware and software. For the measurement device it was decided to take an off-the-shelf PC; the software was based on Linux. The details of the measurement device are shown in Table I.

The advantage of using off-the-shelf hardware and commodity software, is the low price of the measurement device and the simplicity of the software installation and maintenance. A potential drawback is the possible poor performance and scalability. In particular problems may occur because of limited CPU and I/O (e.g., network, disk) speed. To avoid such problems, we selected a Gigabit Ethernet card and a motherboard with a 64 bit bus. It turned out that this PC could easily capture hundreds of Mbit/s.

If, in future measurements, performance and scalability becomes problematic, it should be possible to migrate to some alternative measurement setup, such as TICKET [4], NG- MON [5] or IPMon [6].

There are also hardware based solutions for very fine-grained measurements, such as the DAG cards [7], but these are very expensive. One could also argue that measurements on time scales smaller than 10 milliseconds, as offered by alternative solutions, do not add to the notion of perceived QoS.

## 3 Time-Granularity

The choice for a specific measurement device and setup imposes certain constraints on the achievable time scales for metering. In this section we will discuss

four factors that determine which time granularity is practical for our specific measurement setup.

A first factor is that the switch should copy all traffic from the ports to be monitored to the port to which the measurement device is connected. This switch, like all network devices, contains buffers to temporarily store packets that cannot be transmitted immediately. The switch that was used for our case studies was able to buffer frames for some tens of milliseconds. Since the capacity of the link that connected the switch to our measurement device could easily handle all monitored traffic, delays remained short and buffer overflow did not occur.

A second factor plays a role whenever a full-duplex link is monitored. On such links packets flow in two directions. If both directions of traffic should be copied to a single outgoing link, delays are introduced since only one packet can be copied at a time. As long as the link to the measurement device supports at least twice the speed of the full-duplex link, the delay will be less than the time it takes to transmit a single packet. For Gigabit Ethernet links, this delay will be a fraction of a millisecond.

A third factor is that it takes some time between forwarding a packet within the switch, and timestamping the packet within the measurement device. Without special equipment, it is hard to exactly determine this time, but it is at most a fraction of a millisecond, and the same for all packets.

The fourth factor, which is the most important one, is the resolution of the timestamp itself. This resolution depends on the software that runs on the measurement device. With standard Linux/x86, which was used in our case studies, this resolution is 10 milliseconds (100 Hz). For each packet that the network card delivers to the Linux kernel, a timestamp is added by the network driver in the `netif_rx()` routine. The resolution can be improved by the use of the Time Stamp Counter (`TSC`) feature that is present on modern x86 CPUs. It is also expected that the granularity can be further improved by using the `mmtimer` feature that will probably be around in future chipsets. This feature is designed for multimedia purposes, but can be used for measurement purposes as well. Both `TSC` and `mmtimer` give nanosecond time precision, but are not fully used in current Linux kernels.

From the above considerations it can be concluded that a safe value for the reasonably achievable time-granularity is 10 ms (100 Hz). In the next sections we will use this value as the minimum measurement interval.

## 4 Measurement Tools

The measurement process can be divided into three stages: 1) capturing, 2) processing and analyzing, and 3) visualizing network traffic. These steps are described in more detail below.

For our case studies we chose to process, analyze and visualize the gathered data “off-line”, i.e., network data is stored to disk, and further processing is done afterwards. The reason to do this “off-line”, is that we want to experiment with

analyzing and visualizing the data. In principle, however, throughput-evaluation can be done "on-line", e.g., by using simple counters.

### 4.1 Capturing

The measurement PC receives all traffic from the monitored network link via a Gigabit Ethernet interface. The traffic is captured using `tcpdump` [8] and the associated `libpcap` library. For the throughput analysis performed in this study, it is not necessary to store the complete contents of all frames; for our purpose it is sufficient to store only the first 66 octets of each frame. These octets contain all header information up to the transport layer (e.g., TCP port numbers, if available); this is sufficient to perform, for example, flow arrival analysis. To improve performance and to avoid potential privacy problems, the rest of the payload is ignored. For each frame that is captured, `libpcap` also adds to the capture file: 1) a timestamp (as provided by the Linux kernel), 2) the size of the captured fraction of the frame (i.e., 66 in this study), and 3) the total size of the original frame. Note that capturing results in a huge amount of data, particularly on high-speed networks; a measurement period of 15 minutes can involve multiple gigabytes of data. To handle capture files bigger than 2 gigabytes, it is important to compile `tcpdump` and `libpcap` with Large File Support enabled.

### 4.2 Processing and Analysis

The second step involves preparation for, and the actual analysis of, the gathered data. In studies that focus on throughput, this step is relatively simple: anonymization of the data (if required) and grouping of the captured packets according to the required granularity.

Although from a technical point not a required part of the measurement process, anonymization should be done for privacy reasons. Various tools, implementing different anonymization schemes, are around. In this study the `tcpdpriv` [9] utility has been used, which can be configured for different levels of protection (scrambling of only parts of the IP address, scrambling of transport port numbers, etc.).

For throughput analysis, all packets in the capture file should be grouped. Every group consist of all the packets captured within a certain time interval. In accordance with the discussion in Section III, we chose the minimum time interval to be 10 milliseconds. Depending on the network load, a single group can consist of hundreds of packets. Obviously, the throughput of each interval can be calculated by summing up the sizes of all packets within the associated group, and dividing the resulting number by the length of the time interval (10 ms).

### 4.3 Visualization

From the interim-results of the analysis step, graphs can be plotted. In our case studies, the `GD` [10] library has been used to create images, using `Perl` scripts.

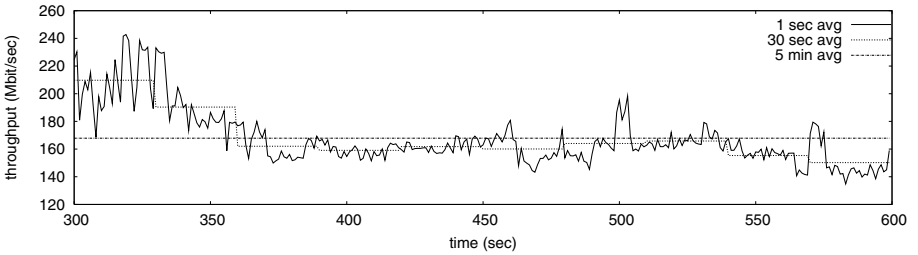



**Fig. 2.** Campusnet - Time-Granularity < 5 min

A problem with visualization is that the amount of information may be too large to properly display in a single graph. Therefore a reduction may be required, e.g., by plotting only the highest 10 ms average throughput value of a longer, for example 10 seconds time interval. The analysis and visualization tools that have been developed as part of this research, are available online from [11].

## 5 Measurement Results

The main goal of this study is to get quantitative figures showing the difference between the traditional 5 minutes traffic measurements of `MRTG`, and finer-grained measurements with time scales up to 10 ms. This section presents the results.

Figure 2 shows, for the Campusnet scenario, the difference between common `MRTG` statistics and measurements on smaller time scales. The time-granularity is increased from 5 minute throughput averages, to 30 second averages and finally 1 second averages. Note that the 5 minute average is around 170 Mbit/s. From the picture it is clear that, within that interval, the average throughput in the first minute is considerably higher than the 5 minute average. This is true for both the 30 seconds as well as the 1 second averages. Some of the measured 1 second average throughput values are even 40% higher than the traditional 5 minute average value. It should be noted that all measurements span 15 minutes; for visualization reasons, the graphs show only part of that interval.

Figure 3 zooms in on the first half second of the measurement of Figure 2. Time-granularity is further increased from 1 second, to 100 ms and finally 10 ms. It should be noted that each 10 millisecond interval still contains hundreds of packets. The graph shows that the 100 ms averages are relatively close to the 1 second average throughput. This is not a general rule, however; other measurements on the same network have shown differences of up to tens of percents. It is interesting to see spikes of over 300 Mbit/s for the 10 ms averages – almost twice the value of the 5-minute average. Note that the figure shows the aggregation of traffic flowing in and out of the Campusnet, hence the possibility of values higher than 300 Mbit/s.

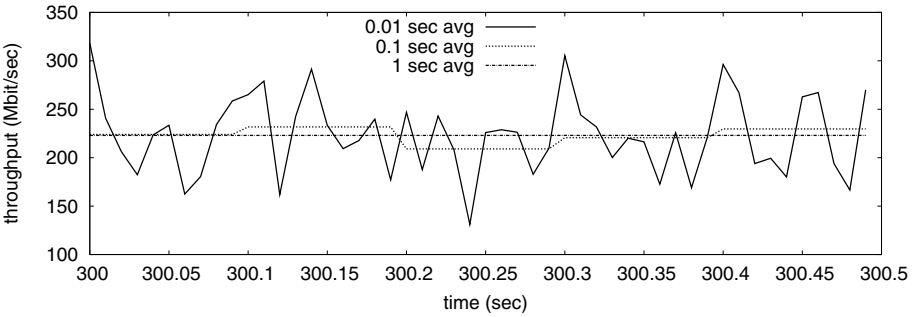



**Fig. 3.** Campusnet - Time-Granularity: $< 1$ sec

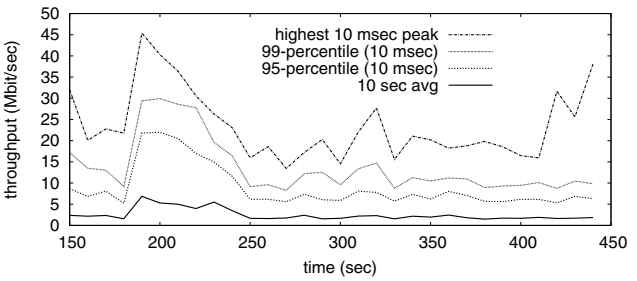



**Fig. 4.** Hosting Provider's Network: Average, Peak and Percentiles

Figure 4 shows throughput statistics for the hosting provider's network. The utilization of this network is relatively low, with some tens of concurrent users at the most. The 5 minute average throughput (not in the graph) for this interval is around 2 Mbit/s. The lowest line shows the average throughput with a time-granularity of 10 seconds. The figure shows that for the 190th till 250th second, the 10 second average throughput values are a multiple of the traditional 5 minute average. The top line in Figure 4 shows the highest 10 ms average within each 10 second interval – thousands of percents higher than the 5 minute average. The other two lines are the 95th and 99th percentile of all 10 ms averages within each 10 second interval. These percentiles are regarded to be a better performance measure than the absolute peak value, as they better describe user-perceived QoS. Also these values are considerably higher than the longer term average throughput. An explanation for the huge differences in "short-term" and "long-term" averages, is the combination of a small number of users, and the high speed at which a single user can send – a modern server can easily saturate a 100Mbit/s LAN. Hence a single user can have, when he sends traffic, a big impact on the traffic aggregate.

To compare between the hosting provider's network and the Campusnet, Figure 5 shows for the Campusnet scenario the same kind of information as Figure 4.

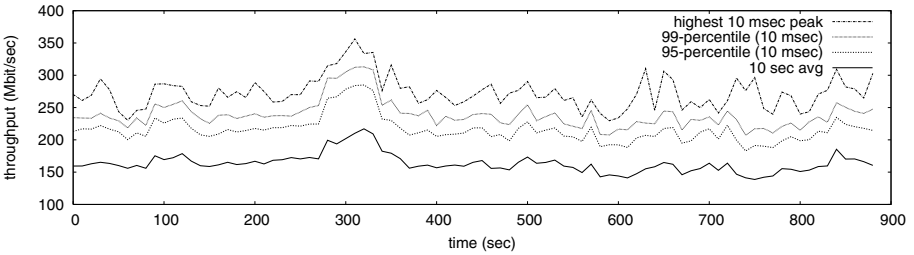



**Fig. 5.** Campusnet: Average, Peak and Percentiles

Note that the 95/99-percentiles of the 10 ms measurements are also well above the "long-term" averages. The (relative) fluctuations, compared to the longer time averages are less, however, than for the hosting provider's network. This is probably caused by the fact that an increase in the number of concurrent users and related link load, in general leads to a decrease in burstiness of the traffic aggregate [12]. But still, the percentiles of the 10 millisecond measurements are tens of percents higher than the 10 second averages. It is important to take this into account while dimensioning a network.

In order to get a better idea on how the (peak) throughput averages increase with a decreasing time window size, we split the measurement data (900 seconds) in partitions, see Figure 6 for the Campusnet scenario, and Figure 7 for the hosting provider scenario. After 0 splits, the average throughput of the entire 900 seconds time window is plotted. After 1 split, we look at the first 450 seconds and the second 450 seconds time window, after 2 splits, we have 4 windows of 225 seconds, etc. After 14 splits, we have time windows of approximately 50 milliseconds. For each different window size, we can now determine the maximum throughput of all windows of a certain size, the standard deviation, and the 95/99-percentiles.

It is obvious that the maximum (average) throughput increases when the time-window size is split in half. It can also be seen, by looking at the deviation plots, that the fluctuations increase when the time window size becomes smaller. It is interesting to see that the maximum, as well as the 95/99 percentiles grow steadily, up to approximately 12 splits, i.e., a window size of about 200 milliseconds; from that point on, the "growth rate" increases. Other measurements on the same networks also show an increasing "growth rate", but the window size at which this change happens varies per measurement.

## 6 Conclusions

In this paper we have quantified the difference between the traditional 5 minutes traffic measurements of `MRTG`, and finer-grained measurements with time scales up to 10 ms. We have performed two case studies, one on the external link of

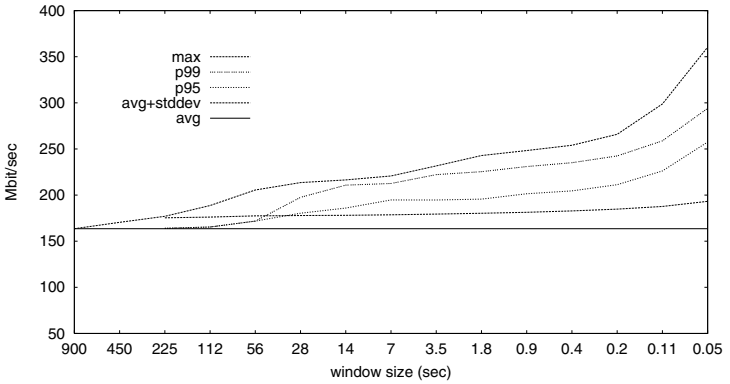



**Fig. 6.** Campusnet: Throughput vs. Time Window

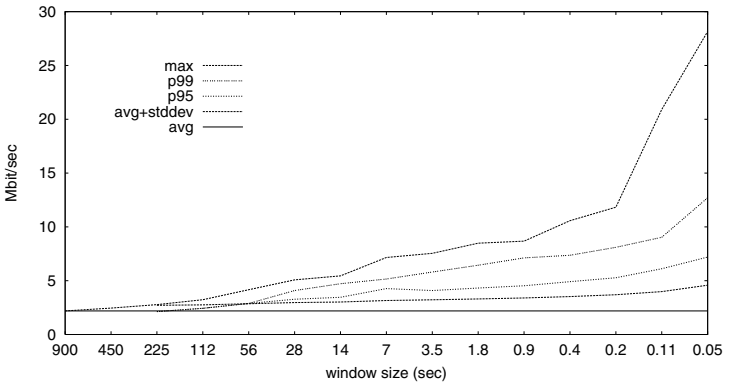



**Fig. 7.** Hosting Provider's Network: Throughput vs. Time Window

a university's residential network, and one on the access line of a small hosting provider.

The access line of this hosting provider served some tens of customers. With such small number of customers, we expected major differences between the 10 ms load figures and the traditional 5 minutes figures. Our measurements showed that these differences could be thousands of percents.

Because the university link served thousands of students, we expected that the differences between the long- and short-term averages would be relatively small. The outcome of our measurements came as a surprise, however. It turned out that, even with this number of users, an overdimensioning of about 100% is required to cater for 99% percent of the "peaks".

Our graphs show the limitations of using `MRTG` figures to overprovision network links. In case bandwidth overprovisioning is the approach to ensure QoS,

it may be better to base decisions on finer-grained measurements, which correspond to the time scale that determines perceived QoS.

We assume that the increasing variability of the throughput on small time scales is influenced by a number of parameters, e.g., the number of concurrent users, the users' access rates, and other traffic characteristics such as file size distributions. In future work we expect to quantitatively investigate the influence of each of these factors, and to derive simple dimensioning rules related to intelligent overprovisioning.

As a side-result, this paper showed that, without any special hardware or software, it is possible to perform traffic measurements with a granularity of up to 10 milliseconds, on network links carrying hundreds of Mbit/s. The tools that we've used to perform these measurements, as well as the tools that were used to analyze the results, can be downloaded from the web.

## Acknowledgements

The authors would like to thank the ITBE, the university's network managers, for the opportunity to meter traffic on the network of the University of Twente and for providing assistance whenever necessary.

The authors would also like to thank Virtu Secure Webservices BV for providing access to its networking infrastructure and support in the measurement efforts.

The research presented in this paper is sponsored by the Telematica Instituut, as part of the Internet Next Generation (ING) project and the Measurement, Modelling and Cost Allocation (M2C) project.

## References

[1] C. Fraleigh, F. Tobagi and C. Diot, *Provisioning IP backbone networks to support latency sensitive traffic*. In *Proceedings IEEE Infocom 2003*, San Francisco CA, USA, 2003.

[2] T. Oetiker. *MRTG: Multi Router Traffic Grapher*. `http://people.ee.ethz.ch/~oetiker/webtools/mrtg/`

[3] R. Poortinga, R. van de Meent and A. Pras. *Analysing campus traffic using the meter-MIB*. In *Proceedings of Passive and Active Measurement Workshop 2002*, pp. 192–201, March 2002.

[4] E. Weigle and W. Feng. *TICKETing High-Speed Traffic with Commodity Hardware and Software*. In *Proceedings of Passive and Active Measurement Workshop 2002*, pp. 156–166, March 2002.

[5] S. Han, M. Kim and J. W. Hong. *The Architecture of NG-MON: A Passive Network Monitoring System for High-Speed IP Networks*. In *Proceeding of 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pp. 16–27, October 2002.

[6] Sprint ATL, IP Monitoring Project, `http://www.sprintlabs.com/Department/IP-Interworking/Monitor/`

[7] Endace Measurement Systems. `http://www.endace.com`

[8] Lawrence Berkeley National Laboratory Network Research. TCPDump: the Protocol Packet Capture and Dumper Program. http://www.tcpdump.org/
[9] Ipsilon Networks. tcpdpriv. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html
[10] Boutell.com, Inc. GD Graphics Library. http://www.boutell.com/gd/
[11] R. van de Meent. Homepage. http://wwwhome.cs.utwente.nl/~meentr/
[12] J. Cao, W. Cleveland, D. Lin and D. Sun, *Internet traffic tends to Poisson and independent as the load increases*, Bell Labs Technical Report, 2001.

# Automating Enterprise Application Placement in Resource Utilities

J. Rolia[1], A. Andrzejak[2], and M. Arlitt[1]

[1] Hewlett Packard Laboratories
Palo Alto, CA 94304, USA
{jerry_rolia,martin_arlitt}@hp.com
[2] Zuse Institute Berlin (ZIB)
Takustraße 7, 14195 Berlin-Dahlem, Germany
andrzejak@zib.de

**Abstract.** Enterprise applications implement business resource management systems, customer relationship management systems, and general systems for commerce. These applications rely on infrastructure that represents the vast majority of the world's computing resources. Most of this infrastructure is lightly utilized and incurs high operations management costs. Server and storage consolidation are the current best practices for decreasing costs of ownership in such environments. However, capacity related decisions about which applications should be placed on a consolidated server are often made informally. This paper presents an approach for automating such exercises. We characterize the complex time varying demands of such applications and then assign them to a small number of servers such that their capacity requirements are satisfied. The approach can be repeated on an on-going basis to ensure the continued efficient use of resources. A case study using data from 41 data center servers is used to demonstrate the effectiveness of the technique.

## 1 Introduction

Today's enterprise infrastructure is lightly utilized. Computing and storage resources are often cited as being less than 30% busy. This is in significant contrast to most large scientific computing centers which run at much higher utilization levels. Unfortunately, the complex resource requirements of enterprise applications, and the desire to provision for peak demand are reasons for such low utilization.

Infrastructure consolidation is the current best practice for increasing asset utilization and decreasing operations costs in enterprise environments. *Storage consolidation* replaces disks within hosts with virtual disks supported by storage area networks and disk arrays. This greatly increases the quantity of storage that can be managed per operator. *Server consolidation* identifies groups of applications that can execute together on an otherwise smaller set of servers without causing significant performance degradations or functional failures. This can greatly reduce the number, heterogeneity and distribution of servers that

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 118–129, 2003.

must be managed. For both forms of consolidation, the primary goal is typically to decrease operations costs.

Recent advances in utility computing make it possible for resource utilities to offer secure server and storage resources to enterprise applications on-demand [1][12]. Each resource utility requires a Resource Management System (RMS) to govern access to and automate the configuration of its resources [13]. Ideally, such an RMS should also automate the process of assigning applications to servers. A set of RMS may then cooperate as part of a grid for enterprise applications. By supporting more applications on fewer servers, such a grid can help to decrease operations and infrastructure costs.

In this paper, our focus is on a specific class of enterprise applications. We define an *application* of this class as a group of operating system processes that are assigned to the same server. These applications operate continuously and have time-varying demands but often require only a fraction of the capacity of their server.

Our contributions in this paper are as follows. We describe the workload characteristics of the above class of enterprise applications and a model for characterizing their resource demands. We then introduce two techniques for assigning applications to a consolidated set of servers. The first is based on a linear integer programming model, the second on a genetic algorithm. We apply the techniques in a case study involving data from 41 data center servers. Our results indicate that the genetic algorithm behaves well with respect to the integer programming approach, requires less computation, and provides greater opportunities for enhancement.

The rest of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 describes our system under study, our workload model, and our experimental design. Assignment algorithms are introduced in Section 4. A case study is given in Section 5. Section 6 offers concluding remarks.

## 2 Background and Related Work

Virtualization features are becoming common within today's enterprise infrastructures. Examples of virtualization features include: virtual Local Area Networks, Storage Area Networks and disk arrays that support virtual disks, and virtual machines and other partitioning technologies the enable resource sharing within servers. These features make it possible for resource management systems to offer joint, secure, and programmatic configuration and control for computing, networking and storage infrastructure [7][12].

The above technologies help to enable a control-loop that can continuously re-evaluate and realize assignment alteratives. To close the loop, applications must provide interfaces that support their migration from one server to another while maintaining their qualities of service.

Assignment must deal with both the long term and short term consequences of consolidating applications on a server. Long term issues pertain to capacity planning. Shorter term issues deal with arbitration, for example, who should

get access to resources when demand exceeds supply. *Long term consequences* include interactions and correlations between the time varying demands of the applications. The demands are correlated if they tend to move together within their demand ranges. *Short term consequences* deal with the scheduling of resources to applications at specific times in a manner that best meets the goals of the resource utility. Short term decisions typically operate on time scales of tens of seconds to hours [3][11]. In this paper, our focus is on long term issues.

There is a substantial literature on the use of optimization methods for resource management problems in computing systems. This includes the seminal work by Chu that uses zero/one integer programming for task and file allocation [4]. This is known to be an NP-hard problem. The model we consider is most closely related to the zero/one multiple knapsack problem which is also NP-hard. [8] demonstrates that genetic algorithms can be used to find good solutions for such problems. We prefer to apply genetic algorithms over other knapsack heuristics [2][10] as they appear to be more robust with respect to future enhancements to our problem definition. Dick *et al.* use a genetic algorithm to schedule embedded system specifications consisting of multiple periodic task graphs [5]. Our problem differs but we aim to exploit the same properties of genetic algorithms. In particular, the ability to consider multiple objectives and to consider general functions when evaluating the merit of an assignment.

## 3 Methodology

This section describes our system under study, the characteristics of the workload we examine, and the experimental design for our study.

### 3.1 System under Study

For the purpose of our study we were able to obtain CPU utilization information for 41 servers in a data center. The data was collected between September 2, 2001 and October 24, 2001. For each server, the average CPU utilization across all CPUs in the server was recorded for each five minute interval. This information was collected using HP OpenView Performance Agent. We consider the work on each server as a single application and model per-interval CPU demand as the product of the number of CPUs on the server and their reported aggregate average utilization.

**Table 1.** Server Groups

| Group | Number of Servers | Server Class | CPU per Server | MHz | Memory (GB) |
|---|---|---|---|---|---|
| 1 | 10 | K-class | 6 | 240 | 4 |
| 2 | 8 | N-class | 2-6 | 550 | 4-8 |
| 3 | 16 | N-class | 2-8 | 440 | 2-10 |
| 4 | 7 | L-class | 2-4 | 440 | 2-8 |

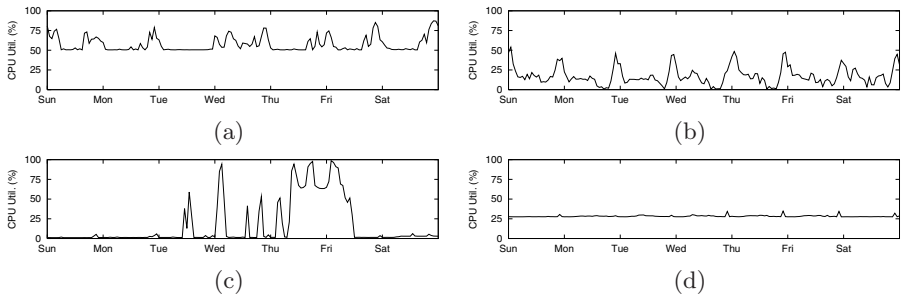



**Fig. 1.** One Week Traces of Application CPU Utilizations

We identified four groups of servers that had similar hardware configurations. We treat these groups separately. Table 1 provides a breakdown of the groups of servers. The most homogeneous is Group one, where all 10 servers have the same number (six) and speed (240 MHz) of CPUs, and the same amount of memory (4 GB). The other groups consist of servers from the same family with the same CPU speed, but vary in the number of CPUs and the amount of memory per server.

For the purpose of our study, we treat the demand of each server as the demand of one enterprise application. We characterize each application using its trace of per interval CPU utilization. In general, we can also consider other attributes such as measures of bandwidth to and from the server and the amount of real memory used.

Figure 1 illustrates the per-CPU utilization of four applications, each over a one week period. These applications portray the kinds of behavior we have observed. Figures 1(a) and (b) illustrate clear time of day effects. The utilizations are similar for the other weeks of data. The first has some constant background load, the other's utilization drops to near zero levels at certain times of day. Figure 1(c) shows the behavior of an application that is less predictable. It has a surge of work towards the end of the week. For this application, some other weeks were basically idle, and others had surges that were more or less pronounced. Figure 1(d) illustrates a server with a flat CPU utilization. There were several of these, some with higher and some with lower CPU utilizations.

Traces provide us with a historical view regarding application behavior. They must be long enough to capture behavior that is representative. We assume that such behavior will repeat itself; therefore we use the traces as input for our assignment algorithms.

When consolidating applications, we assume that an application from a server group is assigned to a consolidated server with a similar configuration. We also assume that the consolidated server has sufficient memory to support the aggregate demand of its assigned applications.

### 3.2 Experimental Design

This section describes our experimental design. In each case the goal is to minimize the number of servers needed for the entire time under study. Our design explores the sensitivity of assignment to several experimental factors as described below.

We consider the following scenarios for assignment:

**A.** Large multi-processor server: we assume that the target server is a single server with a large number of CPUs (e.g., 32, 64). Each CPU's worth of *load* from an application is assigned to a CPU for exactly one measurement interval. We assume processor affinity, i.e. the load cannot be served by any other CPU in this measurement interval. However, the CPU hosting the load is re-evaluated at the boundaries of measurement intervals.

**B.** Small servers without fast migration: here the target is a cluster of identical servers with a small number of CPUs (e.g., 8). Each application is assigned to one server for the whole computation, but we do not assume processor affinity, i.e. the load might receive concurrent service from its server's CPUs.

**C.** Small servers with fast migration: this case is the same as case **B**, but the applications are assigned to the target servers for each separate measurement interval. This implies that an application may migrate between servers at the measurement interval boundaries. We do not consider the overhead of migration in our comparison.

For the above cases, we consider several additional factors.

- *Target utilization* $M$ per CPU. We assign the applications to CPUs or servers in such a way that a fixed target utilization $M$ of each CPU is not exceeded. The value of $M$ that we use is either 50% or 80%, so each CPU retains an *unused capacity* that provides a soft assurance for *quality of service*. Since our input data is averaged over 5-minutes intervals, the unused capacity supports the above-average peaks in demand within an interval. $M$ can also be specified to set aside resources for expected server virtualization overheads. Finally, if per-CPU input utilization data exceeds the value of $M$ for a measurement interval, it is truncated to $M$. Otherwise the mathematical programs can not be solved since no allocation is feasible. With our assignment algorithms, if a utilization is truncated, the application gets assigned to its own server but doesn't fully benefit from the $1 - M$ in soft assurance.
- *Interval duration* $D$. In addition to the original data with measurement intervals of 5 minutes duration, we average the CPU demands of three consecutive intervals for each original server, creating a new input data set with measurement interval length of 15 minutes. This illustrates the impact of measurement time scale and, for the case of fast migration, the impact of migrating applications less frequently.
- $s$, the number of CPUs in each small server. For cases **B** and **C**, we assume that the number $s$ of CPUs of a single server is 6, 8 or 16.

Table 2 summarizes the factors and levels, and indicates when they apply.

**Table 2.** Summary of Factors and Levels

| Factor | Symbol | Level | Applicable Cases |
|---|---|---|---|
| Target Utilization | $M$ | 50%, 80% | **A, B, C** |
| Interval Duration | $D$ | 5 min, 15 min | **A, B, C** |
| Fast migration | - | false, true | **B, C** |
| # of CPUs per server | $s$ | 6, 8, 16 | **B, C** |

## 4 Assignment Methods

This section presents two assignment methods. The first is a linear integer programming approach. It takes traces of application demand as input and packs them onto as small a set of servers as is possible with a limited computation time. This is a compute intensive process but gives us a baseline set of results that can be used for comparison. Our second approach is a genetic algorithm.

### 4.1 Linear Integer Programming Approach

Our goal is to assign the offered applications to as small a set of CPUs or servers as is possible. To encode constraints for this goal, we introduce an *idle application*. Each CPU or server either hosts an idle application or at least one of the offered applications. The demand of the idle application is set to $M$ for case **A** and $s \cdot M$ for cases **B** and **C**.

We designate:

$I = \{0, 1, \ldots\}$ as the index set of applications, where 0 is the index of the idle application, these are the applications under consideration;

$J = \{1, \ldots\}$ as the index set of CPUs (case **A**) or target servers (cases **B** and **C**);

$T = \{1, \ldots\}$ as the index set of measurement intervals (with durations of either 5 or 15 minutes); and,

$u_{i,t}$ as the demand of an application with index $i \in I$ for measurement interval with index $t \in T$. As mentioned above, for all $t \in T$ we set $u_{0,t} = M$ for case **A** and $u_{0,t} = s \cdot M$ for cases **B** and **C**.

Furthermore, for $i \in I$ and $j \in J$ let $x_{i,j}$ be a 0/1-variable with the following interpretation: 1 indicates that the application with index $i$ is placed on a CPU or server with index $j$, 0 when no such placement is made. Note that $x_{0,j} = 1$, $j \in J$ indicates that the CPU or server with index $j$ is hosting the idle application, i.e. it is unused.

The following class of 0/1 integer programs cover our cases. The objective function is:

– Minimize the number of CPUs (case **A**) or servers (cases **B** and **C**) used by maximizing the number of idle applications:

$$\text{maximize} \quad \sum_{j \in J} x_{0,j}.$$

With the following constraints:

- Let the unused CPUs (case **A**) or servers (cases **B** and **C**) be those with highest indices. For each $j \in J \backslash \{1\}$:

$$x_{0,j-1} \leq x_{0,j},$$

  where $\backslash$ denotes removal from a set.
- Each non-idle application $i \in I$ is assigned to exactly one CPU (case **A**) or server (cases **B** and **C**). For all $i \in I \backslash \{0\}$:

$$\sum_{j \in J} x_{i,j} = 1.$$

- For case **A**, each target CPU must not exceed target utilization $M$ in each time interval $t$. Each $t \in T$ gives rise to a new integer program to be solved separately. For all $j \in \mathbf{J}$:

$$\sum_{i \in \mathbf{I}} u_{i,t} x_{i,j} \leq M.$$

- For case **B**, each target server must not exceed target utilization $M$ for each CPU for all measurement intervals $t \in T$ *simultaneously*. For all $j \in J$ *and* all $t \in T$:

$$\sum_{i \in \mathbf{I}} u_{i,t} x_{i,j} \leq sM.$$

- For case **C**, each target server must not exceed target utilization $M$ for each CPU for each measurement interval $t$. Each $t \in T$ gives rise to a new integer program. For all $j \in J$:

$$\sum_{i \in \mathbf{I}} u_{i,t} x_{i,j} \leq sM.$$

As mentioned above, for cases **A** and **C** we solve one integer program for each measurement interval. Case **B** requires one integer program that includes all time intervals.

### 4.2 Genetic Algorithm Approach

We rely on the GALib genetic algorithm library as our genetic algorithm solver [9]. It is an extensible and mature C++ library for genetic algorithm optimization capable of general representations and extensible genetic operators. To use the library, it is necessary to prepare an initial assignment for the solver, to provide an objective function that is called by the solver to evaluate the utility of a next assignment, and to provide a mutation function and crossover method that are called by the solver to generate alternative assignments.

The encoding of a genome, a single assignment, is based on a one-dimensional array where a value $v$ at position $i$ means that application $i$ is assigned to server $v$. For the initial assignment, we simply place all applications on a first server.

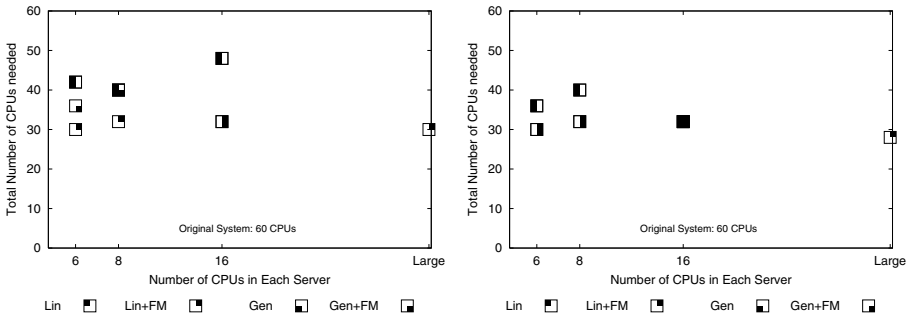
The solver aims to minimize an objective function. In our case, the objective is to minimize the number of servers used. When an assignment is presented to the objective function we compute the peak of aggregate demand on each server. To compute the peak of aggregate demand, we traverse the traces of all applications under consideration. By re-visiting these traces for each possible assignment, we take into account both interactions and correlations as described in Section 2. If the peak demand exceeds $M$ per CPU for any server, then the assignment, i.e. the genome, is invalid. To correct the genome, we use a greedy algorithm that starts with the first server. The algorithm simply keeps a subset of applications on the first server such that the peak of their aggregate demand does not exceed $M$ per CPU. Those applications that must be moved are moved to the next server, possibly causing a server to be added. We note that the crossover method, described later, does not change the number of servers. This process is repeated with the remaining servers until all applications are assigned. The objective function returns the number of servers used. If an individual application requires more capacity than a server can offer, we mark the result of the optimization as infeasible and exit.

The solver invokes mutation and crossover routines to perturb the current assignment and arrive at the next assignment. Our code for the mutation routine implements three kinds of mutations. The first is to take a number of applications and re-assign them to the next higher server. This tends to add servers to the system. The second randomly swaps pairs of applications. The last takes all the applications from the last server and randomly assigns them to the other servers. This last step tends to decrease the number of servers used. As a result our approach both adds and removes servers to increase coverage of the solution space and avoid local minima.

We rely on the solver's built in uniform crossover method to generate next assignments based on earlier assignments. With this approach, the assignment of each application to a server is taken from one of two parents, each with probability $\frac{1}{2}$.

Finally, we describe the termination criteria used by the solver. The solver generates a population with sixty members (assignments); each member starts with the initial solution. The population is permitted forty generations (iterations). The solver returns the best assignment observed. When generating assignments, we specified that 80% of an assignment should be perturbed from one iteration to the next. The crossover probability and mutation probability were set to 0.75 and 0.1, respectively. These values offered good solutions without excessive running time.

For the **B** cases, the assignments in the initial solution are based on the peak of the aggregate demand as computed over the traces. For the **C** cases, we take into account the ability of applications to migrate after each measurement interval. Our problem is to deduce how many servers are needed to support the applications in the presence of such migration. To do this, we consider a *daily profile* of demand. Time of day is partitioned into slots and each slot has a duration equal to that of a measurement interval $D$. We create an initial solution for




(a) $D = 5$, $M = 50\%$ (b) $D = 15$, $M = 50\%$

**Fig. 2.** Numbers of CPUs Required for Group 1

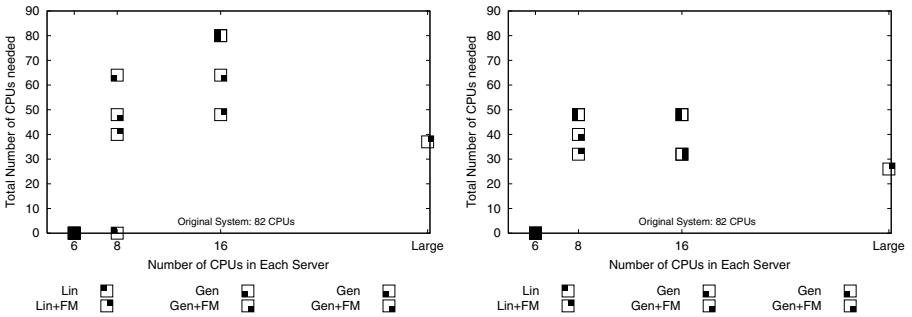
each slot and apply the solver to each slot separately. When computing peak demands for an assignment for a slot, we consider only the subset of measurement intervals from the traces that apply to the slot. To compare the solution with the results of the linear integer programming approach, we report the maximum number of servers that are needed over all slots.

## 5 Case Study

This section presents the results of the assignment methods for the experimental design. Due to space constraints, a subset of results for cases **A, B,** and **C** are presented. We note that there is no guarantee that either of the assignment algorithms offers an optimal assignment.

Figure 2 presents results for Group 1 with $D = 5$ minutes and $M = 50\%$. Case **A** is the Large server case as identified on the x-axis. Cases **B** and **C** correspond to the six, eight, and sixteen CPUs per server cases, without and with fast migration, respectively. Squares in the figure identify the number of CPUs required on the y-axis versus the number of CPUs per server, as shown on the x-axis. The quadrants of each square show whether a solution method/fast migration scenario combination corresponds to that number of CPUs. For example, the upper-left most square in Figure 2(a) shows that for the six CPU per server case, without fast migration, both the linear integer program and genetic algorithm found an assignment with forty two CPUs. The squares below it show that for the fast migration case, the genetic algorithm's assignment required thirty six CPUs while the linear integer program's assignment required thirty CPUs.

As the number of CPUs per server increases, we expect consolidation to be more effective. However, as Figure 2(a) shows, each server, in particular the last server, isn't always well utilized. For this reason, the sixteen CPU per server case shows a jump in the required number of CPUs. The Large case required only thirty CPUs. It has the least waste. The results for the Large case were computed using the linear integer program in a manner similar to the fast migration case.




(a) $D = 5$, $M = 50\%$ (b) $D = 5$, $M = 80\%$

**Fig. 3.** Numbers of CPUs Required for Group 3

Figure 2(b) shows the impact of increasing $D$ to fifteen minutes. This *smooths* reported bursts in utilization, in some cases, letting us pack more work onto fewer CPUs. For example, we report fewer CPUs as required for the six CPU per server case without fast migration, and the sixteen CPU per server case without fast migration.

Figure 3(a) and (b) show results for Group 3 with $D = 5$ minutes and $M = 50\%$ or $M = 80\%$, respectively. As the utilization threshold increases, clearly more work can be added to each server. We note that in both Figure 3(a) and (b), solutions were not possible for the six CPU per server cases. This is because some applications required more than six CPUs even after truncation of demand with respect to $M$. These are identified as squares at $y = 0$ (i.e. 0 CPUs). Note that given a limited computation time, the linear integer program did not return a valid solution for the eight CPU per server case without fast migration in Figure 3(a). The genetic algorithm was able to find a solution. Computation times are discussed in the next subsection.

The results for Groups 2 and 4 were nearly identical for both the linear integer program and the genetic algorithm. Valid solutions were found for all cases in Groups 1, 2, and 4.

From our case study, we find that the genetic algorithm typically finds that the same number of servers are required as the linear integer programming approach. In some cases an additional server is required. These cases usually required an additional CPU or two that force the addition of an entire server.

### 5.1 Computation Times

The linear integer program computations were performed on a server with a 440 MHz PA-RISC processor and 16 GB of memory running HP-UX. We relied on the AMPL 7.1.0 modeling environment and CPLEX 7.1.0 optimizer [6]. CPU time limits were established for each run of the CPLEX tool. For each run, we report the best solution found within its time limit. For scenarios in cases **A** and **C**, time limits were generally 128 seconds. Case **B** scenarios had limits of

4096 seconds. There were a few scenarios in **B** that did not yield good allocations. These were cases where we deemed the bounds for the solution, as reported by CPLEX, to be too wide. They then received 131072 seconds of computation time. As illustrated in Figure 3(a), the eight CPU per server case without fast migration did not return a valid solution at all. Several case **C** scenarios received 2048 seconds of processing time. Unfortunately, we only had limited access to the modeling environment; some runs did not lead to satisfactory allocations. We note that knapsack heuristics [2][10] or a more sophisticated formulation of the linear integer program may reduce its solution time.

The genetic algorithm was performed on a laptop with a 2GHz Pentium IV processor. It typically required tens of seconds per solution, and never required more than two minutes. Solutions were found for each scenario. Even after taking into account differences in processor speeds between the laptop and server, the genetic algorithm performed quickly, especially with respect to the time consuming case **B** scenarios.

## 6 Summary and Conclusions

In this paper, we describe two assignment algorithms for assigning a class of enterprise applications to servers in resource utilities. The first is a linear integer programming approach, the second is based on a genetic algorithm. The algorithms are used in a case study involving data from 41 data center servers.

For our data set, the linear integer programming approach offers the best solutions but had relatively large solution times. It seems most appropriate as an off-line algorithm. The genetic algorithm offered solutions that were nearly as good as the programming approach. However solution times were much lower. It appears to be a good candidate for an on-line assignment algorithm that can be used in a control-loop that automates the assignment and re-assignment of applications. The genetic algorithm permitted us to compute peak aggregate demands as assignments were considered. This helped to reduce the problem size with respect to the linear integer programming approach. With the latter approach, the same strategy is not possible. Furthermore, the genetic approach can be easily enhanced. For example, the objective function can be modified to consider features of assignments that can not be expressed in a linear integer program.

We intend to extend our work in several ways. We plan to collect data from additional data centers for greater periods of time. With longer traces, we can use parts of the traces for deciding assignments and the remainder for validation. In addition to collecting measures of CPU utilization, we intend to collect information on network and storage utilization, in order to get a more complete picture of data center usage. We also plan to extend these techniques to consider multiple demand attributes and classes of service regarding access to resources [13]. Finally, we plan to explore more efficient linear programming strategies and improve our use of the genetic algorithm paradigm.

## Acknowledgements

The authors thank the anonymous reviewers for their helpful comments.

## References

[1] K. Appleby, S. Fakhouri, L. Fong, M. Goldszmidt, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano – SLA based management of a computing utility. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, May 2001. 119

[2] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operations Research*, 123:333–345, 2000. 120, 128

[3] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2001. 120

[4] W. Chu. Optimial file allocation in a multiple computer system. *IEEE Transactions on Computers*, C-18:885–889, Oct 1969. 120

[5] R. Dick and N. Jha. Mogac: A multiobjective genetic algorithm for hardware-software co-synthesis of distributed embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):920–935, 1998. 120

[6] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press / Brooks/Cole Publishing Company, 1993. 127

[7] Hewlett-Packard. HP utility data center architecture. `http://www.hp.com/solutions1/infrastructure/solutions/utilitydata/architecture/` 119

[8] S. Khuri, T. Bäck, and J. Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In *Proc. of the 1994 ACM Symposium of Applied Computation*, pages 188–193, 1994. 120

[9] Illinois Genetic Algorithms Laboratory. Galib. `http://www-illigal.ge.uiuc.edu/index.php3` 124

[10] E. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979. 120, 128

[11] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, pages 247–261, March 2003. 120

[12] J. Rolia, S. Singhal, and R. Friedrich. Adaptive Internet Data Centers. In *SSGRR'00*, L'Aquila, Italy, July 2000. 119

[13] J. Rolia, X. Zhu, and M. Arlitt. Resource access management for a resource utility for enterprise applications. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, pages 549–562, March 2003. 119, 128

# Managing the Performance Impact of Administrative Utilities

Sujay Parekh[1], Kevin Rose[2], Joseph Hellerstein[1], Sam Lightstone[2], Matthew Huras[2], and Victor Chang[2]

[1] IBM T.J. Watson Research Center
Hawthorne, NY, USA
{sujay,hellers}@us.ibm.com
[2] IBM Toronto Lab
Toronto, ON, Canada
{krrose,light,huras,vicchang}@ca.ibm.com

**Abstract.** Administrative utilities (e.g., filesystem and database backups, garbage collection in the Java Virtual Machines) are an essential part of the operation of production systems. Since production work can be severely degraded by the execution of such utilities, it is desirable to have policies of the form "There should be no more than an $x$% degradation of production work due to utility execution." Two challenges arise in providing such policies: (1) providing an effective mechanism for throttling the resource consumption of utilities and (2) continuously translating from policy expressions of "degradation units" into the appropriate settings for the throttling mechanism. We address (1) by using self-imposed sleep, a technique that forces utilities to slow down their processing by a configurable amount. We address (2) by employing an online estimation scheme in combination with a feedback loop. This throttling system is autonomous and adaptive and allows the system to self-manage its utilities to limit their performance impact, with only high-level policy input from the administrator. We demonstrate the effectiveness of these approaches in a prototype system that incorporates these capabilities into IBM's DB2 Universal Database server.

## 1 Introduction

The day-to-day operation of many important software systems involves the execution of administrative utilities needed to preserve the system's integrity and efficiency. These administrative actions are distinct from the functions provided by that system for its users. For example, services provided by database management systems to users are SQL parsing, construction of query plans, query execution, and run time management of database resources. The administrative utilities address recoverability (backup/restore), data reorganization and statistics collection (among other things). In Unix™systems, `cron` jobs are often used to do batch tasks such as recycling of log files. In Java Virtual Machines, garbage collection is an asynchronous administrative utility. In distributed applications,

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 130–142, 2003.

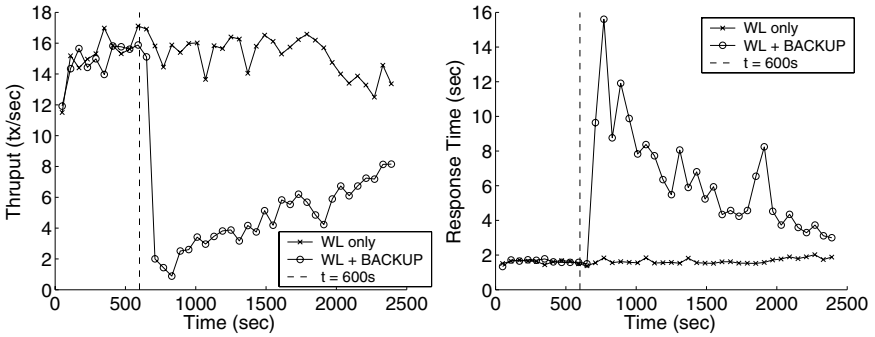



**Fig. 1.** Performance degradation due to running utilities. Plots show time-series data of throughput and response time measured at the client, averaged over a 60s interval

there are "heart beats" that are used to verify that application components are alive.

Such administrative utilities have the following characteristics: (1) their execution is essential to the integrity of the system; (2) however, they can severely impair the performance of the user work (hereafter, referred to as **production work**) if executed concurrently with that work. Hence, administrators typically use overnight periods, holidays or scheduled downtimes to execute such tasks. With the advent of 24×7 operation, such administrative windows are disappearing, creating a significant problem for the system administrator. Therefore, it is highly desirable to provide enforceable policies for regulating the execution of utilities.

Fig. 1 demonstrates the dramatic performance degradation from running a database backup utility while emulated clients are running a transaction-oriented workload against that database. (Details of the testbed are discussed in Sect. 4.1.) The throughput of the system without this backup utility (i. e. , workload only) averages 15 transactions per second (tps). When the backup utility is started at $t$=600sec, the throughput drops to between 25–50% of the original level, and a corresponding increase is seen in the response time. Moreover, over the duration of the utility execution, its impact on the workload decreases (indicating that the resource demands of the utility decrease). Thus, enforcing policies for administrative utilities faces the challenge of dealing with such dynamics.

How should the impact of administrative utilities be managed? Low-level approaches, such as assigning per-resource quotas or priorities for utilities (e. g. , I/O bandwidth quotas) are problematic, since it is a non-trivial problem to determine the appropriate setting of these values. The required values may be different for different resources, and also for different utilities. A higher-level interface is required. Based on our understanding of the requirements of database administrators, we believe that they are interested in policies which are expressed in terms of degradation of production work. One form for such a policy is

> *Administrative Utility Performance Policy*: There should be no more than an $x$% performance degradation of production work as a result of executing administrative utilities.

In these policies, the administrator thinks in terms of "degradation units" that are normalized in a way that is fairly indepedent of the specific performance metric (e.g., response time, transaction rate). It is implicit that the utilities should complete as early as possible within this constraint, i. e. , the system should not be unnecessarily idle.

There are two challenges with enforcing such policies.

> *Challenge 1*: Provide a mechanism for controlling the performance degradation from utilities.

We use the term **throttling** to refer to limiting the execution of utilities in some way so as to reduce their performance impact. One example of a possible throttling mechanism is priority, such as `nice` values in Unix systems (although this turns out to be a poor choice, as discussed later).

> *Challenge 2*: Continuously translate from degradation units (specified in the policy) to throttling units (understood by the mechanism), even when the system and load characteristics are changing.

Such translation is essential so that administrators can work in terms of their policies, not the details of the managed system. The translation should be done by the managed system itself so as to meet the administrative goals. Unfortunately, accomplishing this translation is complicated by the need to distinguish between performance degradation of the production work caused by contention with the administrative utilities and changes in the production work itself (e.g., due to time-of-day variations).

We address the two challenges described above as follows. Our approach to throttling employs a technique that we refer to as **self-imposed sleep** (SIS). By SIS, we mean that the utility algorithm is modified to include points at which the utility invokes an API that removes the utility from the dispatch queue for a prescribed period of time. Our experience has been that it is relatively easy to make these modifications to administrative utilities and that SIS is very effective in practice. Our approach to translating degradation units into throttling units uses a feedback-driven approach. This allows us to generate suitable throttling values without a-priori knowledge of the mapping. Further, it allows the system to adapt to changes in this mapping between the units.

There is a wide range of literature on scheduling and enforcing policies for quality of service (QoS) or differentiated service. For example, reservation based schedulers [1, 2] allocate fractions of bandwith on resources to applications. Such schedulers may allow the administrative policy to be implemented, but these are not readily available in popular commercial OSes. The solution we suggest here is purely at the application level, and since it does not require any changes to the OS, it can be "retrofitted" in most systems. The IBM workload management (WLM) system [3] enforces performance policies for absolute response times

and velocity according to workload classes, by adaptively tuning allocations of multiple resources. However, it is also deeply embedded in the OS, so it is not generally usable.

The theme of our work is similar to the cycle-stealing work of Ryu & Hollingsworth [4] who discuss mechanisms to allow "guest" applications to execute on user workstations without a high performance penalty to the normal users. However, their mechanisms are focused mainly on CPU scheduling, whereas the problem we face involves a multitude of resources. Our proposed administrative policy has a similar form to that often used for real-time garbage collection[5], which is that the application should be able to use the CPU for a given fraction of the time (at some timescale). Again, this is a single-resource scheduling problem, which is difficult to generalize to multiple resources. In [6], the authors describe a time-based scheduler that uses fixed quanta for alternating GC and application execution, as opposed to work-based schedulers that run the GC based on the amount of allocation. This scheme is similar in some ways to our proposed SIS mechanism, except that in their case the quanta sizes are fixed, whereas we seek to find the optimal quantum size to meet the policy requirement.

The approach of using feedback control for administrative policies has also been discussed in the literature. The work of Lu et al.[7] supports a policy of maintaining relative performance levels between different classes of work for a web server, and therefore the classes must share common performance units. In our case, the utility work is not end-user oriented, so its performance metrics are quite different than (and not comparable to) the production workload. The current paper is a continuation of our previous work [8] in enforcing performance policies by using feedback control to translate from high-level policy units into system-level configuration settings. In this paper, the novel ideas include firstly the design of a practical yet general control mechanism (the SIS scheme) and secondly handling the problem where the policy is defined in terms of a quantity which is not directly measurable.

The remainder of the paper is organized as follows. Sect. 2 describes the SIS approach to throttling administrative utilities. Sect. 3 details the feedback control techniques we use to translate from degradation units (as specified in policies) to throttling units (as used to control administrative utilities). Sect. 4 presents the results of experiments using IBM's DB2 Universal Database server and an emulated user workload. Our conclusions are contained in Sect. 5.

## 2 Throttling Mechanism

The purpose of utilities throttling is to regulate the resource consumption of utilities. It is desirable that the mechanism be sufficiently general so that it applies to different operating systems and to utilities with different resource consumption profiles (e.g., CPU bound, I/O bound).

One approach is to use operating system (OS) priorities, an existing capability provided by all modern operating systems. Throttling could be achieved by

```
FUNCTION Utility()
BEGIN
    WHILE (NOT done)
    BEGIN
        ... do some work ...
        SleepIfNeeded()
    END
END
```

(a) Inserting SIS point

```
FUNCTION SleepIfNeeded()
BEGIN
    (workTime, sleepTime) = GetThrottlingLevel() ;
    timeWorked = Now() - workStart ;
    IF (timeWorked > workTime)
        SLEEP( sleepTime ) ;
        workStart = Now() ;
    ENDIF
END
```

(b) SIS implementation

**Fig. 2.** High-level utility structure and sleep point insertion

making the utility threads less preferred than threads doing production work. In principle, such a scheme is appealing in that it does not require modifications to the utilities. However, it does require that the utility executes in a separate dispatchable unit (process/thread) to which the OS assigns priorities. Also, a priority-based scheme requires that access to *all* resources be based on the same priorities. Unfortunately, the priority mechanisms used in most variants of Unix and Windows only affect CPU scheduling. Such an approach has little impact on administrative utilities that are I/O bound (e.g., backup).

Our approach is to use self-imposed sleep (SIS). SIS relies on another OS service: a sleep system call which is parameterized by a time interval. Most modern OSes provide some version of a sleep system call that makes the process or thread not schedulable for the specified interval. Fig. 2 describes a throttling API that uses this sleep service.

To elaborate, many administrative utilities are structured as an outer loop that iterates over some object. For example, in DB2 BACKUP, the outer loop iterates over low-level storage units to be written to the backup device; in garbage collection [5], iteratation is done across memory addresses. Fig. 2(a) depicts how this flow can be augmented by inserting a sleep point called `SleepIfNeeded()`. This is the first main piece of our throttling API.

As shown in Fig. 2(b), the control of utilities is regulated by two variables: a `workTime` and a `sleepTime`. These values are in turn obtained by calling `GetThrottlingLevel()`, which is discussed in Sect. 3.2. The sleep point ensures that when it has been at least `workTime` seconds since the thread was last forced to sleep, the thread sleeps for the prescribed `sleepTime`. In order to get the maximum benefit from this API, the sleep point must be inserted in each place where some basic work unit is processed. Care must be taken that highly contended resources (eg, locks) are not held during the execution of this API.

The sum of `workTime` and `sleepTime` constitutes the time between taking actions that affect utility execution. We refer to this as the **action interval**. Our approach forces the action interval to be a constant that is large enough to encompass several iterations of the work loop of the utility. This value can be either fixed by the system developer or determined at runtime. With a fixed

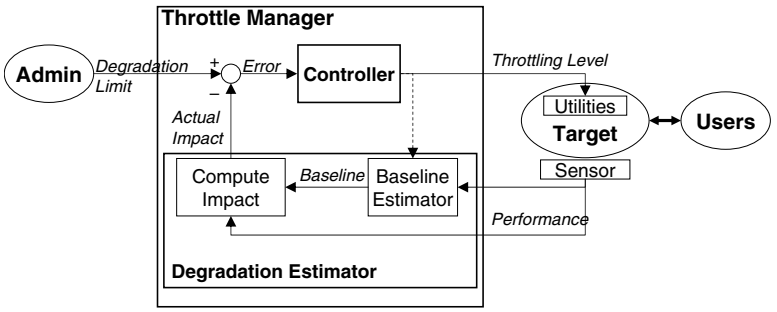



**Fig. 3.** Throttle Manager architecture details

action interval, the throttling level can now be described by one parameter: the **sleep fraction**, defined as $\frac{\texttt{sleepTime}}{\text{action interval}}$, which will be a value between 0 and 1. That is, if the sleep fraction is 0, the utility is unthrottled. If the sleep fraction is 1, the utility is fully throttled.

## 3 Feedback Control for Policy Enforcement

The throttling mechanism by itself does not provide enforcement of any throttling policy. The purpose of the feedback control system described here is to translate degradation units (specified in the policy) into throttling units. Moreover, this system should also adapt quickly to changes in the resource requirements of utilities and/or production work. The system described here is targeted towards supporting policies in the form of "$x$% performance degradation".

The overall operation of our proposed automated throttling system is illustrated in Fig. 3. Administrators specify the degradation limit, which corresponds to the $x$ in the policy described in Sect. 1. The main component is the Throttle Manager, which determines the throttling levels (i. e. , sleep fraction) for the utilities based on the degradation limit as well as performance metrics from the target system.

The internal architecture of the Throttle Manager is also shown in Fig. 3. It consists of two main pieces: a Degradation Estimator and a Controller, which are described below. The Throttle Manager operates in a loop starting with the collection of performance metrics from the target system, and ending with the computation of a new throttling level for the utilities. This loop is executed periodically, at an interval which is related to the desired responsiveness of the Throttle Manager. This interval is called the **control interval**.

### 3.1 Degradation Estimator

The Degradation Estimator component is used to continually estimate the performance degradation due to utilities. It works in two stages, first utilizing a Baseline

Estimator to estimate the **baseline**, which is the performance of the system if there were no utilities running. The baseline value is compared to the most recent performance feedback to calculate the current degradation (as a fraction):

$$Degradation = 1 - \frac{performance}{baseline}$$

A straightforward way to determine the baseline is to suspend all utilities for a brief period and measure the performance during that period as the baseline. This procedure may be repeated periodically to adjust for changing user workloads. Clearly, the responsiveness of the system to a sudden surge in workload will be limited, since the throttling system may not be aware of an underlying baseline change until the next measurement period. Moreover, the abrupt pausing and resumption of the utilities may lead to undesirable short-term end-user performance. Finally, such pauses during idle periods may be unnecessary and hence lead to underutilized system resources.

Alternatively, we can leverage the SIS mechanism to provide a more responsive Throttle Manager. The key observation is that at `sleepTime`=100%, the system should behave as if the utility were not present. We collect datapoints of the form $<$ `sleepTime`, *performance* $>$. We will see below in Sect. 4.2 that for BACKUP, `sleepTime` affects performance in a nearly linear fashion. This is true of all utilities we have studied to this date. Hence, we perform adaptive curve fitting to find the parameters $\theta$ of a static linear model (shown in Eqn. 1) of the effect of `sleepTime` on the selected performance metric ; however, more complex models (e. g. , autoregressive or non-linear) may be used if simple linear models are not adequately accurate.

$$performance = f(\texttt{sleepTime}) = \theta_1 * \texttt{sleepTime} + \theta_0 \tag{1}$$

Such a model can be projected to `sleepTime`=100% to yield an estimate of the baseline. Since this estimate can be updated every action interval, it results in a much more responsive system. We have found that using recursive least squares with exponential forgetting provides reasonable results for the model fit. Exponential forgetting allows the estimator to adapt when either the workload changes or the impact of the utility on the workload changes (as for BACKUP). Details on recursive least squares and similar techniques can be found in the literature[9].

### 3.2 Controller

Given the current degradation level, the Throttle Manager must calculate throttling levels for the utilities. Consider these observations

1. *current degradation* $\leq$ *degradation limit*, which is merely the semantics of the throttling policy.
2. However, if *current degradation* $<$ *degradation limit*, it means that resources are not being used maximally since a larger degradation could be tolerated, i. e. , utilities could be throttled less.

Together, they imply that to balance utility degradation and system utilization, we want *current degradation* = *degradation limit*. As shown in Fig. 3, we define the **error** as *degradation limit* − *current degradation*.

Because of the relatively straightforward effects of `sleepTime` on performance, we use a standard Proportional-Integral (PI) controller from linear control theory[10] to drive this error quantity to zero, thereby enforcing the throttling policy. A PI control structure is proven to be very stable and robust and is guaranteed to eliminate any error in steady-state. It is used in nearly 90% of all controller applications in the real world. A new throttling value at time $k+1$ is computed as follows:

$$throttling(k+1) = K_P * error(k) + K_I * \sum_{i=0}^{k} error(i) \tag{2}$$

In our implementation, this value is posted to shared memory, which is then accessed by SIS implementation using the `GetThrottlingLevel()` call. This interface allows the maximum flexibility to implement the Throttle Manager either as an OS service, as an asynchronous thread within the target application, or as a separate application.

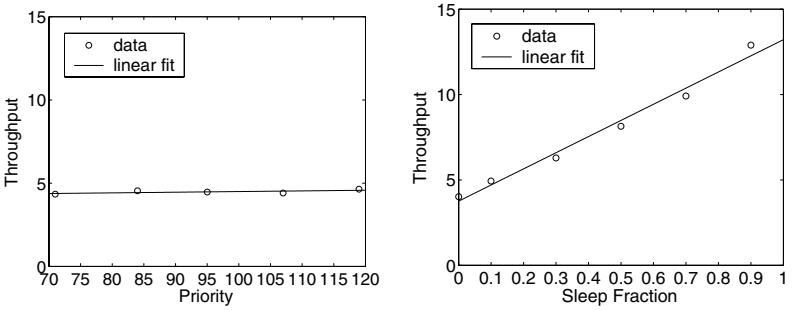
This controller requires that we calibrate the two parameters $K_P$ and $K_I$, which affect the aggressiveness of the controller. We choose a fixed $K_P$ and $K_I$ for all utilities for the experiments in this paper, and we use a control interval of 20 seconds. Using standard control-theoretic analysis, we can show that the values we have used result in a theoretically stable system under the workloads we have studied. We omit this discussion due to space constraints. In principle, the values of these constants could also be determined at runtime. For example, the policy may be augmented by including a desired reliability or variability parameter, which can be combined with online system identification to determine appropriate values for $K_P$, $K_I$ and the control interval. These auto-tuning issues will be explored in future work.

## 4 Empirical Assessments

### 4.1 Testbed Description

Our target system is a modified version of the IBM DB2 Universal Database v8.1 running on a 4-CPU RS/6000 with 2GB RAM, with the AIX 4.3.2 operating system. To emulate client activity, we apply an artificial transaction processing workload which is similar to the industry-standard TPC-C database benchmark. This workload is considered our "production" load. The database is striped over 8 physical disks connected via an SSA disk subsystem. The utility we focus on is an online BACKUP of this database. This backup is parallelized, consisting of multiple processes that read from multiple tablespaces, and multiple other processes that write to separate disks.

For most of the measurements shown here, the workload is run for an initial warm-up period of 10 minutes to populate the buffer pools and other system




(a) OS Priorities (b) Sleep fraction

**Fig. 4.** Average performance at different throttling levels

structures. After this, the utility is invoked under various conditions. The number of emulated users is kept constant for the duration of the run. We measure performance metrics such as throughput, average transaction times, and system utilizations for the entire run.

### 4.2 Comparing OS Priorities and SIS

Two throttling mechanisms are considered: OS priorities and SIS. To study their effectiveness, we altered several database utilities. The details required to implement SIS are described in Sect. 2. To evaluate OS priorities, we use the same sleep point concept as for SIS, except that instead of using `workTime` and `sleepTime`, the process priority is set to the desired value.

In Fig. 4, we show the average throughput for the same workload (25 emulated users) while BACKUP is run at different throttling levels. Each datapoint represents the average performance over a 20-minute run where the throttling level was kept constant at the indicated level. In Fig. 4(a), we study the effect of using different OS priorities for the utility processes. On AIX, processes with smaller priority values receive higher preference for scheduling. Accordingly, the range of priorities for the utilities is varied between the priority of the production processes (70) up to the system maximum (120). Thus, the utility is always given less preference than the production work. In Fig. 4(b), we use the SIS mechanism, varying the sleep fraction from 0.0 to 1.0 across runs.

We see that OS priorities do not have much of an effect on throughput. To understand why, we look at other metrics. On average, the system spends around 80% of its CPU cycles waiting for I/O, indicating that the system is not starved for CPU cycles, and therefore lowering CPU priorities of I/O-bound processes does not help.

On the other hand, SIS has a nearly linear effect in reducing the degradation of the utility on the workload. Note further that at a sleep fraction of 1, the

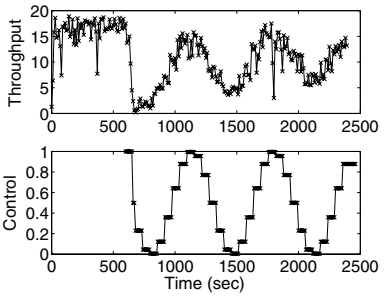



**Fig. 5.** Effect of dynamically varying sleep fraction settings

utility is maximally throttled, hence the workload performance is close to the no-utility case of Fig. 1. This justifies the extrapolation carried out in the Baseline Estimator.

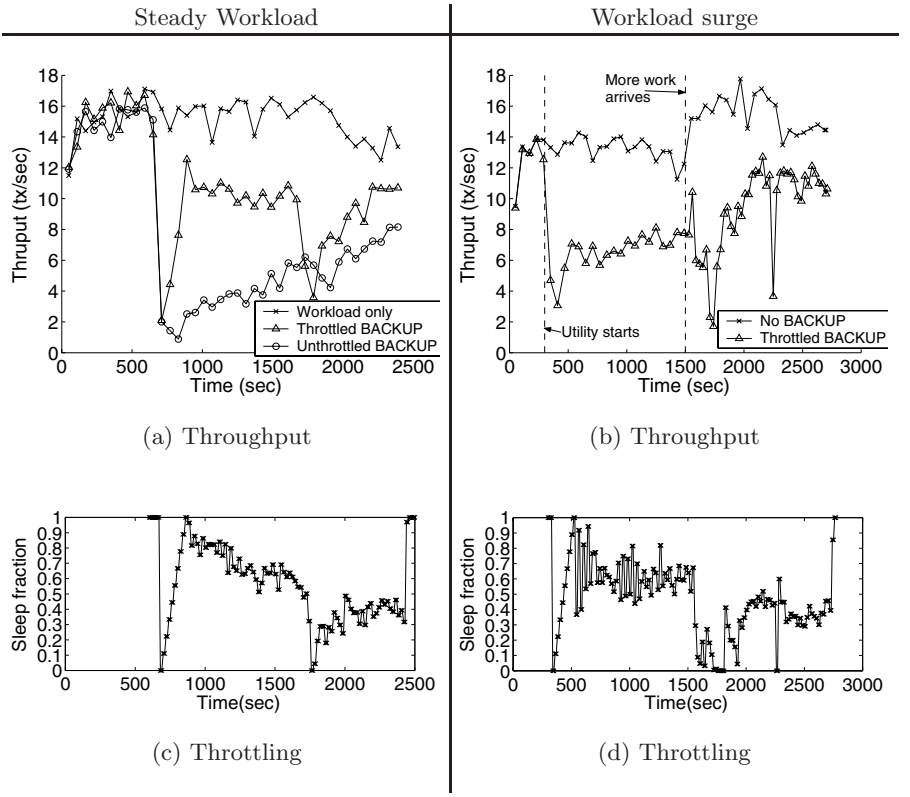
In Fig. 5, we study the dynamic effect of the control mechanism. This allows us to understand factors like delays in effecting a new control value, and transient behaviors like overshoot. The utility is started at 600sec, after which we vary the throttling level in a sinusoid pattern, where each throttling level is maintained for 60 seconds. We see that the sleep fraction is a nice effector for throttling since it has an effect on the utility impact with almost no delay.

From an implementation standpoint, the SIS mechanism requires the administrative utility to be modified. Thus, it is usable mainly by the developers of the utility or software system. In order to insert the SIS points, the main work phase of the utility should be identifiable. This may prove problematic in cases of some legacy systems where the source code is not available or not well understood. However, for current or new software, the SIS mechanism gives developers the ability to build a general and effective throttling capability into their systems. The runtime overhead of this scheme (in terms of its effect on the workload) is not detectable, especially since any amount of throttling is better than no throttling at all.

### 4.3 Effectiveness of Feedback Control

We now evaluate whether the feedback control approach can effectively translate an administrative degradation policy of 30% into appropriate settings for SIS.

We first show in Fig. 6(a) that the throttling system follows the policy limit in the case of a steady workload generated by 25 emulated users. For comparison, the workload performance as well as the effect of an unthrottled utility (from Fig. 1) are also shown. While the average throughput without the BACKUP running is 15.1 tps, the throughput with a throttled BACKUP is 9.4 tps – a degradation of 38%, which is close to the desired 30%. Note how the throttling system compensates for the decreasing resource demands of the utility by lowering the sleep fraction (Fig. 6(c)), resulting in a throughput profile that is more parallel to the no-utility case.




(a) Throughput (b) Throughput

(c) Throttling (d) Throttling

**Fig. 6.** Effect of throttling a utility under a steady workload and a workload surge with a 30% impact policy. The throughput data shown is averages over 1 minute intervals

To highlight the adaptive nature of this system, we consider a scenario where there is a surge in the number of users accessing the database system while the BACKUP utility is executing. We start with a nominal workload consisting of 10 emulated users, and start the utility at 300 sec. At time 1500 sec, an additional 15 users are added (thus, resulting in a total of 25 users). Fig. 6(b) shows the throughput data for the surge, with the no-utility case (in the same scenario) shown for reference. We see that the throttling system adapts when the workload increases, reaching a new throttling level within 600sec. For this case, the pre-surge average throughputs are 13.1 (workload only) and 8.37 (throttled BACKUP) – a degradation of 36%. Analogously, the post-surge degradation is 19%. Note that the sleep fraction used (and the resultant throughput) towards the latter half of the run is similar to the value seen for the steady-workload case, indicating that the models learned by the Baseline Estimator are similar.

The responsiveness offered by the projection-based Baseline Estimator comes at a cost of a small error (typically within 10% in our tests) in exactly satis-

fying the degradation policy. This error arises to a large degree from the errors due to the real-time estimation in the Baseline Estimator. First, there is some inherent inaccuracy in the projection method, as is evident from the projected value of 13.2 tps seen in Fig. 4(b), compared to the actual unthrottled throughput of 15.1 tps. Second, the system stochastics cause the estimation of degradation to be incorrect. As seen by the temporary drop in the sleep fraction near $t$=1800sec in Fig. 6(c), this can cause the controller to violate the policy requirements for a short time window. However, the adaptive estimation corrects itself fairly quickly, so the longer-term behavior is still correct. If such short-term violations are not desirable, we can adjust the forgetting factor in the online estimator to increase the robustness at the expense of its adaptation speed.

## 5 Conclusions and Future Work

Running utility functions against a production system can prove to be an administrative nightmare. In this paper, we have provided one example of the dramatic performance degradation from performing system maintenance tasks while users are active on the system, a situation which is increasingly prevalent in today's 24x7 operations. We argue that the management burden can be eased by a *policy-based* execution of utilities, based on limiting their performance impact on the production workload. Our proposed SIS throttling mechanism proves to be a convenient, portable and effective mechanism for throttling utilities. We also demonstrate how a feedback control loop can be used to translate the policy specification into actions in terms of the throttling mechanism. This throttling system is autonomous and adaptive and thus allows the system to self-manage its utilities to limit their performance impact, with only high-level policy input from the administrator. Our prototype system implemented for utilities running in the DB2 Universal Database achieves within 10% of the desired degradation policy, both when workloads are steady and when they change. This is quite reasonable given the stochastics in the system.

The architecture shown here can be easily adapted for use in other systems; it is not specific to database management systems. The main requirement is that the core of the work phase of the utility should be identifiable, so that the sleep point can be inserted there. A secondary requirement is that the performance metric of interest should be available to be measured; ideally it should be a server-side metric which can be collected at frequent intervals without much overhead.

The results in this paper have focused on a single utility. While we cannot claim that the specific controller proposed here would apply across all instances of all utilities in all systems, we plan to investigate this generality further. The solution we have described here computes a single throttling value for all utilities, which may not be the most efficient. In the case of multiple utilities, it may be advantageous to throttle utilities separately according to their individual impacts on the workload. Our future work will address this issue as well. Finally, while we have shown particular instantiations for the individual components of

our architecture (e. g. , a PI algorithm for the controller, recursive least squares estimator, etc) which work well in combination, the exact choices of algorithms and their parameterizations may need to be adjusted based on the target system. We plan to investigate procedures to automate these steps as well, in particular the determination of the controller parameters (since they can be dependent on the target system).

## References

[1] Bruno, J., Gabber, E., Özden, B., Silberschatz, A.: The Eclipse operating system: Providing quality of service via reservation domains. In: Proceedings of the USENIX 1998 Annual Technical Conference, New Orleans, LA (1998) 235–246 132

[2] Banga, G., Druschel, P., Mogul, J.C.: Resource containers: A new facility for resource management in server systems. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, LA (1999) 45–58 132

[3] Aman, J., Eilert, C.K., Emmes, D., Yocom, P., Dillenberger, D.: Adaptive algorithms for managing a distributed data processing workload. IBM Systems Journal 36 (1997) 132

[4] Ryu, K.D., Hollingsworth, J.K.: Exploiting fine-grained idle periods in networks of workstations. IEEE Transactions on Parallel and Distributed Systems **11** (2000) 683–698 133

[5] Wilson, P.R.: Uniprocessor garbage collection techniques. In: Proceedings of the International Workshop on Memory Management, Springer-Verlag (1992) 1–42 133, 134

[6] Bacon, D.F., Cheng, P., Rajan, V.T.: A real-time garbage collector with low overhead and consistent utilization. In: Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press (2003) 285–298 133

[7] Lu, C., Abdelzaher, T.F., Stankovic, J.A., Son, S.H.: A feedback control architecture and design methodology for service delay guarantees in web servers. Technical Report CS2001-06, University of Virginia, Department of Computer Science (2001) 133

[8] Diao, Y, Gandhi, N., Hellerstein, J.L., Parekh, S., Tilbury, D.M.: Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server. In: Proceedings of Network Operations and Management. (2002) 133

[9] Astrom, K.J., Wittenmark, B.: Adaptive Control. 2nd edn. Addison-Wesley Publishing Company (1994) 136

[10] Ogata, K.: Modern Control Engineering. 3rd edn. Prentice Hall (1997) 137

# Policy-Based Autonomic Storage Allocation

Murthy Devarakonda, David Chess, Ian Whalley, Alla Segal, Pawan Goyal, Aamer Sachedina, Keri Romanufa, Ed Lassettre, William Tetzlaff, and Bill Arnold

IBM Corporation

**Abstract.** The goal of autonomic storage allocation is to achieve management of storage resources, including allocation, performance monitoring, and hotspot elimination, by specifying comparatively high-level goals, rather than by means of low-level manual steps. The process of automation should allow specification of policies as administrator specified constraints under which the resources are managed. This paper describes the system design and implementation experiences from a prototype autonomic storage manager being developed in IBM Research. The prototype is being developed for a storage network that includes a SAN switch, an IBM Enterprise Storage Subsystem, and AIX servers. Our early experience from this prototype implementation is that there are a large number of mundane manual steps in storage management and it is feasible to automate them such that the automation is driven by higher-level goals under policy control. However, to manage heterogeneous storage a standard ontology is needed for specification of goals and how to achieve them.

## 1 Introduction

Today storage allocation and management of allocated storage is an incredibly manual process. This is especially true in networked storage, where many choices exist and several elements need to be configured and managed. In allocating network storage, administrators have to determine resource availability, use a methodology for allocation of resources, and execute several commands or use graphical user interfaces to implement the allocation decisions. Once storage is allocated it is necessary to monitor for performance hotspots and capacity, determine how to alleviate these problems, and then carry out appropriate actions to alleviate the problems. The monitoring needs to be an ongoing process to assure acceptable levels of service.

This need for manual involvement in storage allocation and management significantly increases the real cost of storage, is a source of serious errors, often results in significant delays in deployment of applications, and is the root cause of general dissatisfaction with the state of the art of storage management. Therefore, storage vendors are working on automating these processes. At IBM Research, we have been exploring a management automation methodology based on autonomic computing principles [1] and policy-based management [2]. Specifically, the goal of policy-

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 143-154, 2003.

based, autonomic storage allocation is to achieve allocation of storage resources, their performance monitoring, and hotspot elimination by specifying comparatively high-level goals, rather than by means of low-level manual steps. This process of automation should allow specification of policies as constraints under which the resources are managed.

In this paper, we describe the design and implementation of a policy-based autonomic storage allocation manager prototype, called ALOMS-Tango, being built at IBM Research. ALOMS-Tango allows administrators to define classes of service for storage in terms of performance and space metrics, set up alerts to be generated if the actual performance of the allocated storage comes within a given fraction of violating the requirements of the class of service, and visualize the configuration of the storage system for the allocated storage and identify performance bottlenecks.

We modified IBM's database management system (DB2) to enable a user to request a database tablespace container in terms of the class of storage service it requires, forward such requests to ALOMS-Tango, and use the storage allocated as policy-managed autonomic storage.

In response to a request for the creation of policy-managed storage in a particular service class, ALOMS-Tango automatically performs computations and low-level configuration operations necessary to create required storage, monitors the performance of the allocated storage, and produces alerts if the trigger conditions (as specified through the user interface) are met.

The prototype implementation demonstrates the potential that autonomic systems have for reducing the workload on administrators by allowing them to specify their requirements in comparatively high-level terms and by automating routine low-level allocation and monitoring operations. One of the key lessons we learned from the present prototype is the need for a standard ontology to support different types of storage systems. Storage Management Initiative-Specification (SMI-S) [3] is a great start in this direction but our preliminary analysis of SMI-S indicates that further standardization is needed for autonomic storage allocation.

Previous work in the area of storage management dates back to IBM DFSMS on mainframes [4], which provided policy-based allocation of storage and life cycle management of data. More recently, several efforts from the HP Labs have developed tools for automated storage allocation [5][6]. Our work is the same spirit as DFSMS but it extends the ideas to open systems as well as incorporating a feedback loop into the management process (which is absent in DFSMS). Unlike the HP labs work, our allocation management system has been designed for handling dynamic environments that require on-demand storage allocation and management of changes in the storage system status or its usage.

The ALOMS-Tango prototype fits into the larger vision of Autonomic Computing first articulated by Paul Horn [15] and later elaborated upon by Kephart and Chess [1]. Kephart and Chess see new autonomic managers of traditional computing elements such as data base systems and storage systems. These autonomic managers take on tasks that were previously done by administrators. Policy is key to this process as a way to codify the actions and goals of the administrator so that autonomic managers can independently carry them out. The ALOMS-Tango prototype has been careful to

create functions that are well defined with respect to whether they would be in the autonomic data base manager or the autonomic storage manager.

The rest of the paper is organized as follows. Section 2 gives a functional description of our prototype. Section 3 describes design and operation of ALOMS-Tango. Section 4 provides prototype implementation specifics. Section 5 concludes the paper with an overview of what we have learned from the prototype development.

## 2 Functional Description

With the current storage management tools, when a storage-critical application such as a database requires storage, the process that administrators go through can be time consuming, tedious, and error prone, and can thus result in significant deployment delay. A methodology for allocating and managing storage for database applications is described in this tutorial from IBM [7]. First, administrators determine the performance and availability needs of the application, based on previous experience, a prototype installation, or even using mere guesswork. Next, they determine capabilities of the storage systems they have at their disposal, how the storage systems are connected to application host systems, present storage resource commitments, and the available capacity. Tools available for these purposes are at best inconsistent and often non-uniform across multiple vendor products. Next, administrators use a resource allocation strategy to decide on how to configure the storage systems for the application. The strategy can range from *ad hoc* to the best current practice depending on the experience of the administrative staff and the time available for this purpose. Next, the administrators have to configure the storage infrastructure according to the decisions made in the previous step. Usually, different graphical user interfaces are provided for various elements of the infrastructure as well as for products from different vendors. Therefore, this step is prone to serious human errors. Once these steps are successfully carried out, the administrators are ready to use the allocated storage in an application, such as the database. For example, a database administrator may type a command such as,

```
create tablespace fast1 managed by database using (de-
vice '/dev/rlv23' 20000),
```

to make use of storage that has been created as a logical volume named */dev/rlv23* in an AIX server. Note that many installations use 100s and even 1000s of tablespace containers. From this description the tedious, error prone, and time-consuming aspects of storage allocation should be clear.

Furthermore, regular monitoring of storage performance is critical to assure that the allocated storage meets application requirements. If the initial assumptions about the application requirements are incorrect or the requirements change, the process of reconfiguration is as daunting as or even worse than the initial configuration.

The ALOMS-Tango prototype supports performance goal specification as a named service class that can be later used in a storage allocation request. When a storage consumer, such as the database application, requires storage, it can obtain that storage simply by making a request to ALOMS-Tango, specifying an initial size and a service

class name. ALOMS-Tango also allows setting monitoring policies such that it can automatically detect if a certain storage allocation is missing its service goals by a margin observed over a certain time period. As an example, the following service class can be defined in ALOMS-Tango:

```
Service Class "Gold"
   Maximum size = 100Gbytes
   Throughput = 20 Mbytes/Sec
   Response Time = 5 ms/4K block
   Seq/rand access ratio = 100%
```

The attributes used in the service class definition represent requirements from an application point of view rather than the capabilities of storage hardware. This distinction is quite important and has been discussed in the context of a conceptual framework for policy-based storage management, in earlier publications [4][8]. DB2 can request a storage container of class "Gold" using the following SQL statement:

```
create tablespace fast1 managed by database using
(device '/policy/Gold' 20000)
```

In general, we have considered two interfaces between the database and the storage system; one used when a table space is created and the other when all of the storage backing a table space has been filled. Today, both the Data Base Administrator (DBA) and the Storage Administrator (SA) do part of the storage administration in both of these cases. In order to do automatic storage allocation and administration, the system must be primed with policies that will cause the system to act as the SA and the DBA would have. To accomplish this we have the storage administrator create a number of named storage service classes. The Storage Administrator conveys the available storage service classes to the DBA. When the DBA creates a table space, the policy name is associated with the table space. This allows a DBA to create a table space by only specifying the name of the policy that is to be used when storage space is needed.

The other key moment is when allocated storage has been completely filled. Further data insertion normally results in an out of space error condition, which would impact applications using the database (causing them to roll back). We have defined an interface that the database can use to request more storage. The data base system keeps certain metadata about the storage objects that are being used to back table spaces. This metadata mainly consists of the names of policy-managed logical volumes (in the terminology used in AIX, for example) or files that back a table space. If this association between table spaces and storage objects is kept in both the database and storage there would be potential problems of consistency between them, which would have to be resolved with two-phase commits and logs. In order to avoid this, the interface chosen calls for the data base to hold the association, and present the necessary information across the interface when storage is needed. Thus when storage is needed the database presents the storage policy name and the storage objects currently in use. It is left up to the storage manager to provide more storage either in the form of new storage objects or by extending the size of existing storage objects. This is done in a way that is consistent with both the policy and past allocations.

Note that ALOMS-Tango storage allocation requests can be made without DB2, by using its graphical user interface, command line interface, or the C/XML based pro-

gramming interface. In other words, while we cite DB2 as an important user of this prototype, ALOMS-Tango is neither dependent on DB2 nor limited to it.

An alert policy can be set to monitor throughput on allocated storage, as in:

```
generate an alert, if [throughput] for [a container]
falls below [95%] of the value specified in its service
class definition, in a 10-minute period.
```

In the future, ALOMS-Tango will be extended to support hotspot detection, remediation of the hotspot problems through re-allocation of storage, and a richer set of policies that will automatically select service classes based on requester (i.e. customer, applications, and workload) and usage patterns.

One can clearly see the difference between the present manual process of storage allocation and the automation provided in ALOMS-Tango. The administrators are freed from the tedious and error-prone tasks of determining resource capabilities, bookkeeping of the present usage, resource allocation strategies, and execution of configuration operations. Instead the administrators are given the tools to specify high-level goals in the form of a comparatively small number of service classes, request storage by specifying service classes, and monitor deviations from service delivery through policies. As mentioned earlier, the future plan is to extend this framework with richer policies that include automatic assignment of service classes to storage requests based on the usage patterns and/or requester characteristics.

## 3 ALOMS-Tango System Operation

Figure 1 shows the system view of the ALOMS-Tango prototype. An application or an administrative user can invoke ALOMS-Tango functions. In our prototype, as stated earlier, we are using a modified version of IBM DB2 as the user of storage. The modified DB2 makes storage allocation requests via a shared library developed for this purpose, and the shared library sends these requests to the ALOMS-Tango Management Unit (ATMU), the box in the right, lower quadrant of Figure 1. The current prototype supplies allocated storage as raw logical volumes, for use as DMS (database-managed storage) tablespace containers in DB2; we also have a variation of the prototype that provides files or file systems for use as SMS (system-managed storage).

While the control flows via the shared library to the ATMU for storage allocation, actual I/O requests go directly from the application using storage to the relevant component of the storage infrastructure, just as they do in the absence of the autonomic storage allocation manager. DB2 has also been modified to send a signal to an extender agent when a tablespace becomes full, which in turn sends a request to the ATMU to enlarge the corresponding container.

The ATMU is responsible for resource provisioning and re-provisioning, collection of configuration information and performance metrics, policy management and enforcement, and user interface support. The ATMU interfaces with the storage infrastructure via sensors and effectors. Sensors help the ATMU in collecting configuration information about the storage infrastructure. They also help in obtaining performance metrics.

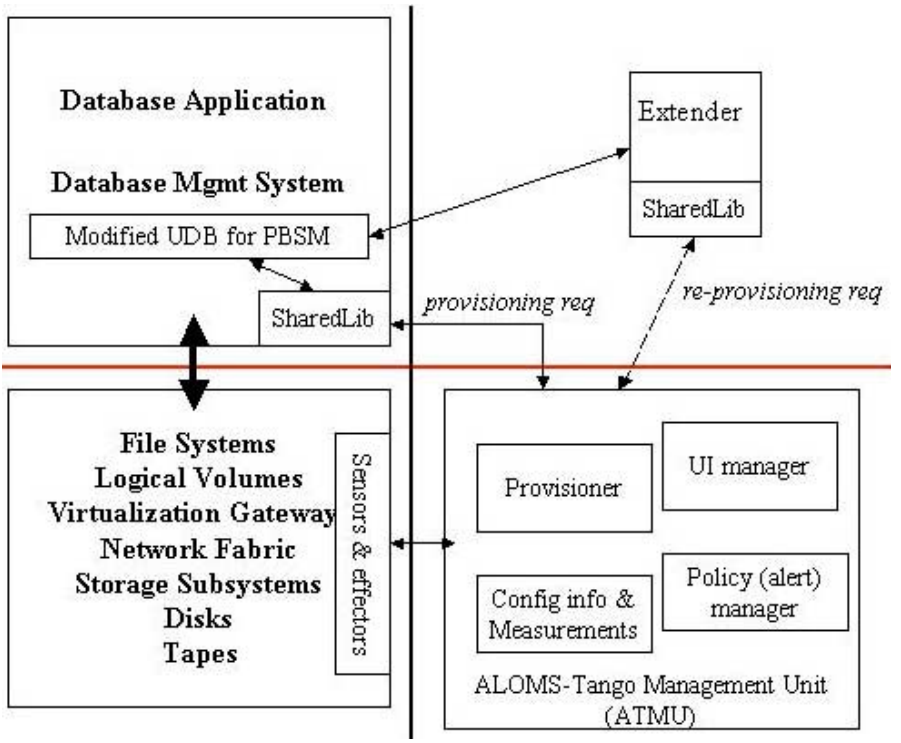



**Fig. 1.** A system view of the ALOMS-Tango design

The effectors, on the hand, carry out configuration commands as instructed by the provisioning logic in the ATMU. The storage infrastructure includes all elements that enable storing and retrieving of data, such as the operating system, file systems, volume managers, storage subsystems, disks, and tape. Specific infrastructure supported in the prototype is described later.

The UI (user interface) manager component of the ATMU allows the administrator to define the service classes that were introduced in Section 2. A storage administrator can define several service classes, each with a descriptive name. In most environments, service class definitions will be done infrequently relative to the frequency of storage allocation. While these service classes are currently defined in terms of maximum size, response time, throughput, and the ratio of sequential and random accesses in the current prototype, it will be possible in the future to specify additional attributes, such as an availability metric. The UI manager enables alerts to be created, such that they will be generated if the actual performance of a storage container comes within a given fraction of violating the requirements of the class of service. Lastly, the UI manager also helps to visualize the configuration of the storage system and identify performance bottlenecks.

The provisioner component of the ATMU embodies a large portion of the management intelligence and often orchestrates the overall functioning of the management software. At the system initialization time, it builds a logical model of the storage

infrastructure based on the configuration information collected via sensors. This logical model includes information such as the storage space available, whether disks are configured in a RAID format, how many physical paths exist from storage subsystem to the host, and how the physical storage is mapped into the operating system supported data access abstractions (i.e. logical volumes).

The provisioner accepts requests for creation and enlarging of policy-based storage containers in particular service classes, and automatically performs computations and low-level configuration operations necessary to create and enlarge the required containers. The logical model of the storage infrastructure coupled with the knowledge of how to configure each element of the model enables the provisioner to automatically perform the low-level configuration operations. The logical model, capabilities of the elements in the model, and capacity management algorithms enable the provisioner to determine the "best" allocation strategy to meet a given creation or enlarging request.

The policy manager makes use of both a policy repository and a policy execution environment. The policy repository validates and stores policies and policy schemas. The policy execution environment has been carefully designed as an extensible system so that when new policies are introduced into this prototype in the future they can be supported with relative ease. In the current prototype, the policy manager monitors the performance of the storage containers based on the measurements available in the configuration information and measurements component, and produces alerts if the trigger conditions (as specified through the user interface) are met. The policy execution environment is such a key element for extensibility of our system in order to support richer policies; the next section will describe it greater detail.

The configuration information and measurements component is a local repository of the static and dynamic information collected from the storage infrastructure using the sensors. The provisioner as described uses the configuration information earlier. The configuration information also includes information about storage allocations that have already been made. The UI manager component uses this information to present a visualization of the configuration of the storage system, and then in combination with the aggregated measurements it is also used to identify performance bottlenecks. The dynamic information, which is a time series of measurements from the storage infrastructure, is aggregated for the use in the policy-based alert management.

The ATMU is designed and implemented as an autonomic computing element, as outlined in [9]; it presents itself to other systems as a Grid Service conforming to the Open Grid Services Architecture [10] for the purposes of management requests (such as setting a policy or requesting a new data container be allocated). All inter-module and external communication in ALOMS-Tango is via the exchange of XML documents through standard transport protocols.

**Policy Execution Environment**

The policy execution environment in ALOMS-Tango has been built with concepts and the framework developed in IETF/DMTF policy work [11][12]. It has three subcomponents: a policy agent, a translator, and a rule engine. The policy agent retrieves relevant policies from a policy repository (in an XML schema) and uses the translator

to convert them into a form that is suitable for the rule engine. Rules are executed either in a periodic mode or in a request-response mode, as appropriate.

When the policy manager in the present prototype is initialized, its policy agent sends a request to the policy repository, and in response obtains alert policies that are valid, in-force, and applicable. In addition, the policy agent subscribes to receive updates for future changes to the alert policies.

The policy agent then submits the retrieved policies to the policy-to-rule translator, which performs translation-time checks on the policies. If there are no errors, it creates a set of rules that can be executed by the rule engine. Translation-time checks typically involve range checks and checks to ensure that the policies expressed are ones that the rule engine can execute.

With the generated rules, the policy manager, through the policy agent, invokes the rule engine to evaluate the rules at regular intervals or in response to events, using a new set of measurements for each invocation. If the conditional part of a rule evaluates to true, then the action indicated by the action part is carried out. The rule engine used in ALOMS-Tango is from an intelligent agent construction framework called ABLE [13].

The rule engine references certain variables and functions in evaluating the condition part of a rule, and (when necessary) in carrying out the action part. These typically represent input values received from sensors, and actions such as creating an entry in a log. For example, the alert policy shown in Section 2 may result in the following rule:

```
if (observedValue("pmdo1", "throughput", "minutes", 10)
<  95% of expectedValue("pmdo1", "throughput")) then
(createAlert("logEntry", "pmdo1 throughput is below 95%
of specified value…"))
```

This rule states that if the measured throughput on a policy managed storage object called `pmdo1`, measured in ten-minute intervals, falls below 95% of its expected value, then an entry should be created in the alert log.

In this rule, the functions `observedValue` and `expectedValue` provide access to measurements of the storage infrastructure and service class definitions respectively. The function `createAlert` produces the alert.

Since we wish to use a general-purpose rule engine and to design the policy manager for extensibility, we cleanly separate the rule engine from the rest of system by providing a well-defined interface for resolving rule engine references to variables and functions. This interface consists of callbacks and mapping functions.

For example, for the alert policy above, `observedValue`, `expectedValue`, and `createAlert` calls become callbacks into the alert management part of ATMU, where they are mapped to the appropriate methods in the sensors, the service class definition repository, and the alerting system. This clean separation of policies, rule execution, and variable/function mapping from one another, easily extend the policy execution environment to support new policies, as described in [14].

## 4 The Prototype

The ALOMS-Tango prototype has been implemented to support several storage infrastructure configurations including IBM SSA drives attached to an AIX server and IBM Enterprise Storage Subsystem (ESS) connected to an AIX server via a SAN switch. Here we will describe the support for the SAN-based configuration.

Figure 2 shows the SAN-based configuration we used for the prototype. An AIX server hosting DB2 is connected to a SAN switch and the IBM ESS is also connected to the SAN switch. The ALOMS-Tango management unit (ATMU) runs in a separate Linux server.

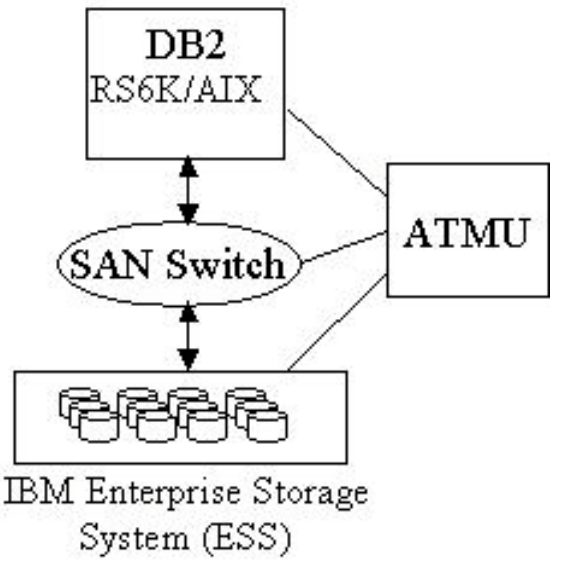



**Fig. 2.** The SAN-based ESS configuration

The provisioner needs to build a logical model of this SAN-based ESS infrastructure configuration. In the current prototype this building of a logical model is implemented statically by writing a customized "SAN-based ESS" module in the provisioner component of the ATMU that understands this configuration. In the future we hope to develop a generic module that can discover and build a logical model based on standardized management interfaces such as the SMI-S.

For this configuration there are two sets of sensors and effectors, one set for the storage infrastructure in the AIX server, and another set for the IBM ESS. We did not manage the SAN switch hence we did not require sensors and effectors for it. The logical model built for this configuration can be best seen in a screen shot of the configuration of a policy managed data object allocated in this configuration (Figure 3).

Figure 3 shows how a policy managed data object ("PMDO_2") is built from the elements of the storage infrastructure, including the lowest level disk resources. At the bottom of the tree are these physical disks, which are aggregated into a disk group

called dg8, from which a RAID5 array called vs0 is configured. From this RAID5 array a logical disk ("LUN") called 1000 was created and was provided to the AIX server as a physical volume named hdisk2. This physical disk is configured under a pseudo-device called vpath0 to mask multiple paths to the logical disk (Figure 3 shows only one data path). A volume group called tVG_0 is created on this pseudo-device, from which an AIX logical volume named tLV_3 is created as the realization of the policy-managed data object pmdo_2. The dashed line shows the separation between the ESS and the AIX server. The construction of this logical model is performed in a module in the provisioner that is specialized for this configuration.

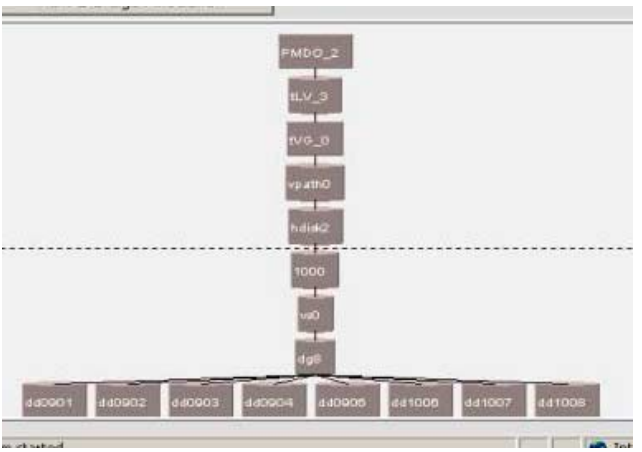



**Fig. 3.** A logical model of the storage infrastructure for the SAN-based ESS configuration

Having built this logical model, the provisioner then develops a capacity management strategy using this model. In the present prototype, the provisioner creates a fixed number of storage pools (or "bins"), from which it allocates resources to meet incoming creation requests.

To broadly apply this strategy, in the present prototype we create these storage pools at the AIX volume group level (i.e. at the tVG_0 level in Figure 3). Each volume group is assumed to have certain performance and space capacity based on how they are built from the infrastructure elements below it. Figure 4 shows creation of four volume groups each corresponding to a RAID5 array in the IBM ESS.

Logical volumes are created out of these volume groups so as to meet service class goals specified in storage allocation request. These logical volumes are the actual data handles using which the requesting applications can make use of the storage resources. How different requested allocations are assigned to different bins is at the heart of an allocation algorithm. We used a simple "worst-fit" algorithm by which each of the incoming requests is distributed as well as possible across the four bins. In Figure 4, we show how two "Fast" (premium) class storage allocations and five "Slow" (ordinary) class allocations are distributed among the storage pools. Note that the "size" of each allocation, as shown in Figure 4, represents performance requirements, not the more traditional size requirement.

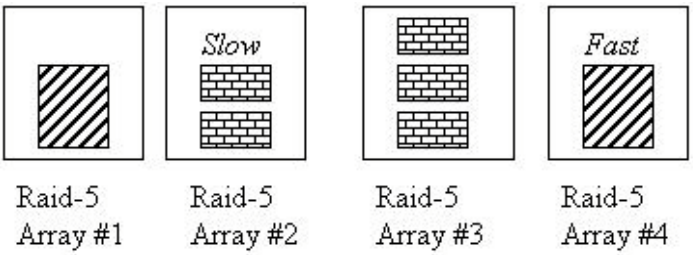



**Fig. 4.** A capacity allocation strategy used in the prototype

Both the capacity planning strategy and the request allocation algorithm could be improved significantly. For example, the "worst-fit" algorithm is prone to local minima, which may be sub optimal.

## 5 Conclusion

From this early but complete prototype we make the following observations:

1. The tedious, error prone, and mundane tasks in the storage allocation process can be automated;
2. The ability to build a common logical model of storage for heterogeneous storage subsystems requires a common way of describing many elements in a typical storage infrastructure, including the operating system support;
3. There is a need to characterize important architectural characteristics of a storage subsystem using a standard model so that it can be properly configured, using management software, to achieve desired quality of service. Alternatively, storage subsystems may directly support quality-of-service, by virtue of having internal policy-based autonomic systems. The latter only changes the location of where the autonomic manager runs (i.e. inside rather than the outside of a storage subsystem), and therefore the need for an explicit description of the storage subsystem architecture remains.

## Acknowledgements

The authors wish to thank Norm Pass and Jai Menon (both from IBM Almaden Research Center), Lorraine Herger, Steve White, Dinesh Verma, and Hoi Chan (all from IBM Watson Research Center); Jack Gelb and Jimmy Strickland (both from IBM Systems Group) for their invaluable guidance and technical help. Special thanks are due to Al Stuart (IBM Systems Group) for support and for facilitating acquisition and set up of our test environment.

## References

[1] Jeffrey O. Kephart and David M. Chess, "The Vision of Autonomic Computing," Computer Magazine, IEEE, Jan 2003.

[2] Dinesh C. Verma, "Policy-Based Networking: Architecture and Algorithms," New Riders Publishing, 2001.

[3] SNIA (Storage Networking Industry Association), "SMI-S Specification, Public Review Draft (v. 1615)," 15 Apr 2003, available at http://www.snia.org/tech_activities/SMI/bluefin

[4] Jack P. Gelb, "System-Managed Storage," IBM Systems Journal, Vol 28, No 1, 1989.

[5] Guillermo A. Alvarez, John Wilkes, Elizabeth Borowsky, Susie Go, Theodore H. Romer, Ralph Becker-Szendy, Richard Golding, Arif Merchant, Mirjana Spasojevic, and Alistair Veitch, "Minerva: An automated resource provisioning tool for large-scale storage systems," ACM Transactions on Computer Systems, Vol. 19, No. 4, pp 483-518, November 2001.

[6] Eric Anderson, et al, "Hippodrome: running circles around storage administration," Proc. of USENIX FAST '02 conference, June 2002.

[7] Barry Mellish, John Aschoff, Bryan Cox, and Dawn Seymour, IBM ESS and IBM DB2 UDB Working Together, IBM Redbook SG24-6262-00, IBM International Technical Support Organization, October 2001 (available on the Internet at ibm.com/redbooks).

[8] Murthy Devarakonda, Jack Gelb, Avi Saha, and Jimmy Strickland, "A Framework for Policy-Based Storage Management," in Proceedings of Policy 2002 (Intl Workshop on Policies for Distributed Systems and Networks), Monterrey, CA, June 2002.

[9] David W. Levine *et al*, "A Toolkit for Autonomic Computing," IBM Developerworks Live, 2003.

[10] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman; "Grid Service Specification," Open Grid Service Infrastructure WG, Global Grid Forum, Draft 2, 7/17/2002.

[11] DMTF, "IETF/DMTF policy framework," http://www.dmtf.org/download/presentations/junedev01/track/0613-01_policy.pdf

[12] DMTF, "CIM Core Policy Model white paper," DSP0108, February 2001, http://www.dmtf.org/education/whitepapers.php

[13] Joseph P. Bigus, Jennifer Bigus, "Constructing Intelligent Agents Using Java," Second Edition, John Wiley & Sons, Inc., 2001.

[14] Murthy Devarakonda, Alla Segal, and David Chess "A Toolkit-Based Approach to Policy-Managed Storage," in Proceedings of Policy 2003 workshop (Intl Workshop on Policies for Distributed Systems and Networks), Lake Como, Italy, June 2003.

[15] Paul Horn, "Autonomic Computing: IBM's Perspective on The State of Information Technology", IBM Corporation, http://www.research.ibm.com/autonomic/manifesto, October 2001.

# Visual-Based Anomaly Detection for BGP Origin AS Change (OASC) Events

Soon-Tee Teoh[1], Kwan-Liu Ma[1], S. Felix Wu[1], Dan Massey[2], Xiao-Liang Zhao[2], Dan Pei[3], Lan Wang[3], Lixia Zhang[3], and Randy Bush[4]

[1] Computer Science Department, University of California
Davis, CA 95616, USA
{teoh,ma,wu}@cs.ucdavis.edu

[2] Networking Group – East (NGE), USC/ISI
Arlington, VA, 22203, USA
{masseyd, xzhao}@isi.edu

[3] Computer Science Department, UCLA
Los Angelos, CA, 90095, USA
{peidan,lanw,lixia}@cs.ucla.edu

[4] IIJ, Bainbridge Island, WA 98110, USA
randy@psg.com

**Abstract.** To complement machine intelligence in anomaly event analysis and correlation, in this paper, we investigate the possibility of a human-interactive visual-based anomaly detection system for faults and security attacks related to the BGP (Border Gateway Protocol) routing protocol. In particular, we have built and tested a program, based on fairly simple information visualization techniques, to navigate interactively real-life BGP OASC (Origin AS Change) events. Our initial experience demonstrates that the integration of mechanical analysis and human intelligence can effectively improve the performance of anomaly detection and alert correlation. Furthermore, while a traditional representation of OASC events provides either little or no valuable information, our program can accurately identify, correlate previously unknown BGP/OASC problems, and provide network operators with a valuable high-level abstraction about the dynamics of BGP.

## 1 Introduction

A statistic-based anomaly detection system is designed to detect any significant differences ***statistically*** between a long-term historical profile and a short-term behavior. We need to apply some mathematical models both to produce the long-term profiles and to compare them with short-term behaviors. On the other hand, a "visual-based" anomaly detection system is to utilize human's cognitive visualization capability to detect any significant differences ***visually*** between the particular human user's mental model [1] of the long-term behavior and his/her short-term visual observation. In

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 155-168, 2003.

short, a visual anomaly is something that catches his/her eyes. While the idea of visual-based anomaly detection is certainly not new – we have been doing it in our daily life for thousands of years, the focus here is to report our results in applying this concept to detect potential security problems due to OASC (Origin AS Change) events under the BGP routing protocol.

With the popularity of the Internet technology, unintentional faults and intentional intrusions directly on network protocols, such as routing protocols, have become a serious threat to our Internet-connected society. In 1997, a buggy BGP (Border Gateway Protocol) [2] router falsely de-aggregated thousands of network addresses which disabled the Internet in the entire US east coast for up to 12 hours. In 2001, worm attacks such as CodeRed and Nimda were spread around the Internet, while the Renesys report [3] discussed a possible correlation between worm attacks and routing protocol stability. In [4], a different interpretation of the Renesys observation indicated that the large increase in the number of BGP messages was mainly due to the behavior of the measurement points, and probably not related to the BGP routing stability.

Driven by these faulty or intrusive instances on the Internet infrastructure, many research teams have studied how to either design new protocols and/or enhance existing protocols such that the Internet can be more robust and fault/intrusion-tolerant. For instance, the Secure BGP (S-BGP) [5] protocol utilizes the PKI infrastructure to authenticate and authorize the route update messages in an inter-domain environment. In [6], a program is developed to statistically profile the "stable" BGP paths leading to critical DNS servers and to filter out "unusual" routes potentially being falsely injected by attackers or simply misbehaving BGP routers.

Another complementary approach to handle these Internet vulnerabilities is via ***an interactive process*** between network administrators/operators and network management systems with visualized network/security information. We believe that, at least in the short term, a network system with machine intelligence alone will have certain limitations in detecting and responding to novel attacks/faults targeting on the Internet infrastructure itself. For instance, given a colorful image of some BGP routing data, an experienced human operator might discover numerous facts about the Internet instantly, while an analysis program must already have the mechanisms built in to achieve the same results. One of the most difficult tasks in intrusion detection is "event correlation," but via human visualization, this task may be much more plausible. Furthermore, due to the complexity and size of the Internet, it is already a very difficult task to evaluate Internet's "health" and clearly identify the root causes of some observed symptoms. Without a comprehensive understanding of the Internet, it is not certain that some new Internet protocols, architecture, or enhancements will be effective in responding to the problems we have today.

This paper describes our design, implement, and evaluate an experimental visual anomaly detection system for BGP OASC (Origin AS Change) events. In the following section, we will describe our visualization design for the BGP/OASC events in Sections 2, 3, and 4. Then, in Section 5, we will report our results in using our visual-based anomaly system to detect non-trivial BGP/OASC problems.

## 2 Visualization Design

Information visualization [7] has emerged as one of the most active areas of computer science research in recent years due to the explosive growth of the World Wide Web. In contrast to scientific visualization, which is primarily about transforming numerical data defining physical structures or phenomena in three spatial dimensions into pictures, information visualization generally maps very large amount of textual, symbolic, or relational data into spatial forms that can be displayed graphically. Data visualization exploits the human vision system to help us explore and better communicate with others particular aspects of the data under study. The process of visualization is an inherently iterative one consisting of multiple steps. A standard visualization process is depicted in the following figure:

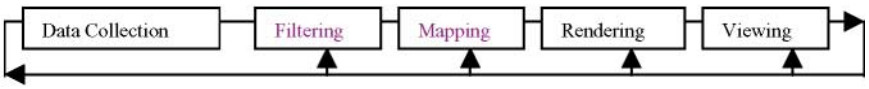



**Fig. 1.** a standard visualization process

In this research work, we pay special attention to the filtering and mapping steps, which prepare the collected data for rendering and viewing. Generally, the filtering step: removes "noise" from the raw data, reduces the data to a more manageable size, or enhances particular aspects of the data.

For visual anomaly detection, it extracts and organizes particular aspects of the data (e.g., the Origin AS changes in BGP routing) for the subsequent steps. The mapping step transforms the filtered data into a collection of graphics entities with appropriate properties (e.g., colors, transparency, and texture) for rendering.

To summarize, our visualization design has the following four advantages. First, our system integrates the capability of cognitive pattern matching (i.e., utilizing human's capability in recognizing special "possibly previously unknown" patterns). Second, the visual/graphical information from our program may trigger the human's intelligence and memory to reason and analyze the observed situation. A human operator will obtain experience via this process and be trained to more accurately identify the problems quickly. Third, the interactive monitoring and analysis process provides a feedback loop from the human back into our program. Finally, the human expert can provide an in depth explanation about the potential problems using the annotated images and/or animations derived. In the following section, we will demonstrate these advantages via the example of BGP/OASC.

## 3 OASC (Origin AS Change) Events in BGP

In this section, we very briefly describe the BGP routing protocol and the definition of OASC events.

The Internet is made of thousands of Autonomous Systems (ASes), loosely defined as a connected group of one or more IP prefixes which have a single and clearly defined routing policy. BGP (Border Gateway Protocol) [2] is the standard inter-AS

routing protocol. A BGP route lists a particular prefix (destination) and the path of ASes used to reach that prefix. The last AS in an AS path is the origin of the BGP routes (or the origin AS). The concept of origin AS is critical for the consideration of routing protocol security as it implies the ownership of the IP address prefix (destination). An OASC (Origin AS Change) event [9,10] occurs if we observe any change in the IP address ownership.

One example of OASC events, called MOAS (Multiple Origin AS), is when suddenly we observe multiple ASes simultaneously claiming the ownership of the same address prefix. More precisely, suppose prefix $d$ is associated with AS paths $asp_1 = (p_1, p_2, \ldots, p_n)$ and $asp_2 = (q_1, q_2, \ldots, q_m)$, and $p_n$ is not equal to $q_m$.

An OASC event can be either valid or invalid. The MOAS example described above is legitimate if each originating AS can directly reach the prefix. On the other hand, if one of the origin ASes cannot reach the prefix, then it is an invalid MOAS conflict and may be due to some malicious attacks. The problem of detecting invalid OASC events is further complicated by BGP operational practices. RFC1930 [8] only recommends (not requires) that each prefix originate from a single AS. In general, we have no way to determine whether an OASC event is the result of a fault, an attack, or a legitimate operational policy.

Faulty aggregation or de-aggregation may cause invalid OASC events. For instance, an AS advertises an aggregated prefix, even though some of more specific prefixes are not reachable by the AS, while there are no other more specific routes available to reach those more specific prefixes. On the other hand, as an example of de-aggregation, on April 25th, 1997, a severe Internet outage occurred when one ISP falsely de-aggregated most of the Internet routing table and advertised the prefixes as if they originated from the faulty ISP. The falsely originated prefixes resulted in many invalid OASC events, which had a serious impact on Internet routing.

For producing OASC events, we used the raw data from the Oregon Route Views server (peering with 54 BGP routers in 43 different ASes) to obtain the BGP routes and AS paths. The Oregon Route Views data is particularly attractive because it provides data from a number of different vantage points. Overall 38225 MOAS events were observed over 1279 days, and there is a significant increase from 683 events (in average) in 1998 to 1294 events (in average) in 2001 as shown in the following "traditional 2D" figure.

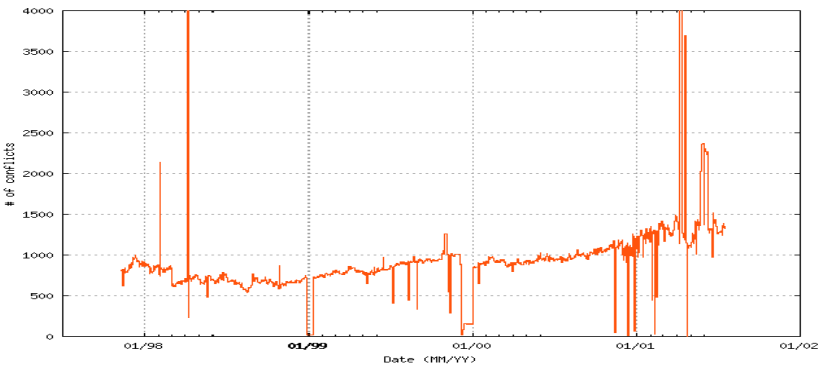



**Fig. 2.** MOAS events in the Internet from 1998 to 2001

Via the above 2D figure about OASC events, we can spot some event spikes/anomalies in the past. For instance, in early 2001, one single AS caused 9177 OASC events in one day. However, from the figure itself (or the raw data), it is not easy to derive more valuable information about the problem. Neither could we tell what was the response or reaction from this or other ASes after this particular BGP problem instance.

# 4 The Design of Our Visual-Based Anomaly Detection System

We have designed and prototyped a visualization program to support an interactive process for analyzing BGP OASC events. Via our initial experiments, we have clearly observed some great advantages in using information visualization techniques to analyze and correlate a large number of BGP/OASC events. In this section, we will first briefly describe our design. Then, in the next section, we will show a few scenarios of using our program to not only detect the problem but also quickly nail down the root cause to the detected problem.

## 4.1 Types of Origin AS Changes in BGP

As mentioned before, from the raw BGP data collected, we can produce a set of "Origin AS Change (OASC)" events. Each Origin AS Change (OASC) event contains the following five attributes:

| | |
|---|---|
| *Prefix* | is the IP prefix whose Origin AS has changed. |
| *AS-before* | is a list of the associated AS(es) before the change. |
| *AS-after* | is a list of the associated AS(es) after the change. |
| *Date* | is the date on which the change occurred. |
| *Type* | is the type of a OASC change event. |

Furthermore, OASC events are classified into 4 main types and then further classified into 8 types in total. The 4 main classes are:

| | |
|---|---|
| *B*-type: | An AS announces a more specific prefix out of a larger block it already owns. |
| *H*-type: | An AS announces a more specific prefix out of a larger block belonging to another AS. In other words, this AS "punches a hole" on prefix addresses of others. |
| *C*-type: | An AS announces a prefix previously owned by another AS. |
| *O*-type: | An AS announces a prefix previously not owned (and therefore owned by ICANN by default). |

The C-type and O-type changes are further classified by whether they involve Single Origin AS (SOAS) or Multiple Origin ASes (MOAS):

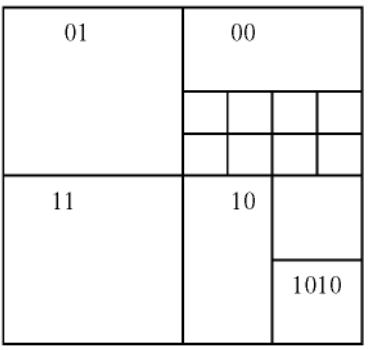
| | | | |
|---|---|---|---|
| CSM: | *C*-type change from | SOAS | to MOAS |
| CSS: | *C*-type change from | SOAS | to SOAS |
| CMS: | *C*-type change from | MOAS | to SOAS |

CMM: *C*-type change from MOAS to MOAS
OS: *O*-type change involving SOAS
OM: *O*-type change involving MOAS
H: *H*-type change always involving another AS (being punched a hole)
B: *B*-type change always involving itself only

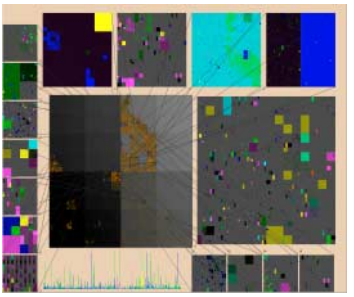
In the visualization, different colors are associated with each of the eight types. While this paper is black and white only, a colorful version of this paper can be downloaded from our web site.

### 4.2 Representing IP Address Prefixes

In representing BGP/OASC events, 2 key concepts are IP address prefix and Autonomous systems. We will first describe our quad-tree representation of IP address prefixes.



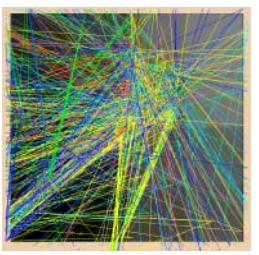

In our prototype, each IP prefix maps to one pixel on a square. The mapping is done in a traditional quad-tree manner as shown on the left. In a quad-tree, a square is repeatedly subdivided into 4 equal squares. In mapping a 32-bit prefix to a square, we start with the first two most significant bits of the address to place the IP address in one of the 4 squares in the second level of the quad-tree. We then use the next two most significant bits to place the IP prefix in the appropriate third level square within this square. We do this repeatedly until we can place the prefix in a square the size of a single pixel. The prefix is mapped to that pixel.



As an example, the following is the visualization of data for 416 days up till February 19, 2001. The main window shows the quad-tree mapping of the entire space of 32-bit IP address. A pixel is colored yellow if an Origin AS Change occurred on the current day (February 19, 2001), and colored brown to green if a change occurred on a previous day (January 1, 2000 through February 18, 2001). In the windows showing detail, a square is used to depict each change, with hue determined by the type of the change, brightness determined by how long ago the change occurred (present day data shown the brightest), and size determined by the mask of the prefix. The background of the main window is shaded according to the IP prefix the pixel represents. The brighter the pixel, the larger the IP prefix represented.

Due to the limitations of a computer screen space, we use a 512x512 pixel square to represent the entire 32-bit IP prefix space. With only 512x512 pixels, even though many IP prefixes map to the same pixel, we found that this is sufficient in spreading out the IP addresses in BGP/OASC events. IP prefixes sharing similar more significant bits would be in close proximity on the screen. With an additional level of zooming into a portion of the data, we can view individual IP prefixes as shown above. In the detail windows, each IP prefix is shown as a square or a rectangle. The size of the rectangle indicates the size of the block of IP addresses; a prefix with a smaller mask gets mapped to a larger rectangle.

### 4.3 Relationship Between Prefix and AS



To represent the relationship between IP address prefix and different ASes, we place 4 lines surrounding the IP square, and an AS number is mapped to a pixel on one of the 4 lines. A line is then drawn from an IP address to an AS number if there is an Origin AS change involving that IP address and that AS number. This mapping takes advantage of the user's acute ability to recognize position, orientation and length. This figure shows visually the IPAddrPrefix-AS relationship of Origin AS Changes of

“a typical day” (April 5, 2001). The color of each line represents one of the eight different OASC types.

Since there are more AS numbers than pixels, more than one AS numbers map to a single pixel. Again, we provide zooming features for the user to differentiate between AS numbers, which map to the same pixel in the main display. The lines representing changes for the AS in focus is shown with brighter and more saturated colors than other changes. This effectively highlights the AS, fading the other changes into the background.

### 4.4 Animation and Other Features

For the time dimension, our program shows one day's data at a time, and allows the user to animate the visualization (each frame showing consecutive day's data). With this ``movie'' display, the user can build up a mental long-term profile in his/her mind, and detect temporal patterns. To assist our memory of patterns from previous days, we allow a user-defined window of a certain number of days prior to the currently shown date. Data from these previous days are displayed, but with darker, less saturated colors, so that the current day's data stands out.

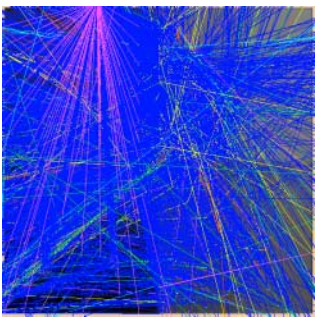
For the convenience of the user, we also provide textual display of the IP address or AS number represented by the pixel clicked by the user. Another feature for convenience is a slider bar to tell the date of the current data shown. The user can click on the bar to choose the date to show. A simple plot of the total number of changes of each type on each day is shown with the bar.

By choosing parameters like what IP prefixes to zoom in on, which AS numbers to focus on, which type of changes to view etc., the user follows an interactive process to navigate abstract information in different levels of details. Depending on the combination of chosen parameters, the user can see the overall pattern of the data, or the user can focus attention on very specific parts of the data. Different choices would reveal different anomalies and information.

## 5 Detecting and Analyzing OASC Anomalies Visually

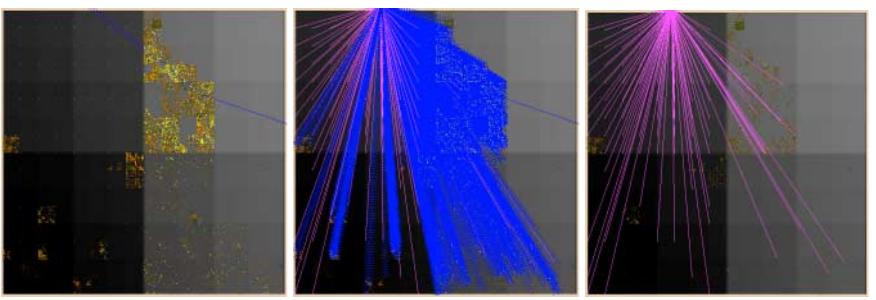
A simple “counter-based” 2D figure for historical OASC events, as shown earlier, can represent some simple OASC related anomalies. But, the value of the information is relatively inadequate to the system administrators dealing with the network instances. In this section, we present two examples of using our visualization program interactively to identify and analyze OASC problems. With our program, the system administrators can easily move deep into the data and discover critical and abstractive facts above a large set of low-level network events. In reading this part of paper, please use a color viewer or printer.

### 5.1 Interactive Visual Analysis: "What Went Wrong on August 14, 2000?"



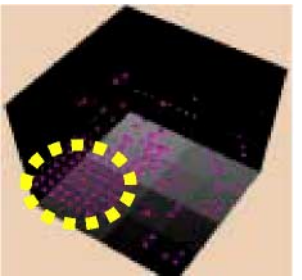
On August 14, 2000, when we turned on all eight OASC types and monitored on all ASes, we visually observed an abnormally large number of blue-colored lines (H-type OASC events). Since the amount of H type OASC events was large to catch our attention, our initial hypothesis was that, due to configuration errors possibly, one or more ASes punched a large number of holes on the IP address prefixes belonging to other innocent ASes.

Immediately, to verify our hypothesis, we used the "AS detail" feature in our program to select only one single H type OASC event and displayed the ASes being involved. The rationale is that, if a very small number of ASes are the root causes for the aggressive Hole-punching problem, selecting one of such event would lead us to one of the "trouble makers."



The figure above on the left shows that after we selected one AS randomly (AS-11724 in our example). In the same figure, the solid line connects the victim AS (AS-11724) and the prefix address (207.50.48.0/21) being hole-punched, while the dash line connects the attacking AS (AS-7777) and a subset of the prefix address (207.50.53.251/32). In other words, AS-7777 would attract all the traffic toward 207.50.53.251 from AS-11724, which supposed to be the true owner. Immediately, we know that the potential attacker (or faulty BGP router) was from AS-7777. Now, we can use the features in our program to only select the OASC events related to AS-7777, and we have the middle snapshot. In fact, after focusing on AS-7777, we can easily validate that this AS was the only AS causing H type events on August 14, 2000. However, in the rightmost picture above, we also isolated the pink-colored lines (OS type OASC events), and interestingly, it seems to us that the pattern of OS type

events was regularly distributed across a region of IP address prefixes that has never been used or allocated in Internet as in the right figure.

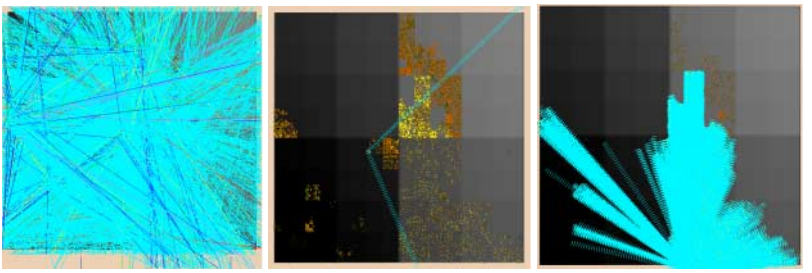


To validate further about what exactly is going on, we used our 3D representation to analyze all the OS type OASC events. With the left figure, from the left middle part (circled by a thick dash line), it is very clear and interesting that AS-7777 announced prefix addresses forming a grid in the unused IP address space. Based on the location and shape of the grid and the raw events, we concluded that AS-7777 falsely announced from 65.0.0.0/8 to 126.0.0.0/8 plus many others.

Please note that the discovery of the OS type OASC event grid is trivial by human, if the visual orientation is right. However, the same task would be very difficult for a fully automated intrusion detection system to reveal this type of facts unless the pattern matching mechanism for grids has been included in advance. Certainly, this case shows the limitation of the traditional intrusion detection systems in detecting "unknown/new/novel" attacks, while our visual-based anomaly detection system has a very good chance to catch them. In the case of August 14, 2000, with a few clicks interactively, our system not only helped detect the problem, but also, quickly nailed down the trouble source, AS-7777. Furthermore, via visualization, it even can tell the details about the errors from AS-7777.

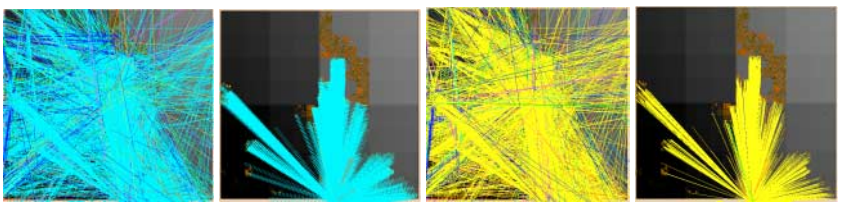
### 5.2 Interactive Visual Correlation: "What Was AS-15412 Doing in April 2001?"

Earlier we showed that on April 5, 2001, things looked "normal". But, on April 6, the next day, we observed an unusual amount of skyblue-colored lines (i.e., CSM-type OASC events). A CSM event indicates that a particular IP address prefix was originated by one AS the previous day, but more than one ASes are claiming it on the current day. With the possibility of multi-homing and private AS numbers, a small amount of CSM events is probably acceptable, but the snapshot on the left below is visually abnormal.



Again with a simple interactive analysis process, very quickly we identified that AS-15412 is the only AS injecting all the CSM OASC events on April 6, 2001. In the middle snapshot above, it shows that AS-15412 is conflicting with a victim AS-10132, which is the Eastar Technology Center in Hongkong. (We randomly picked one of the victims and we used the "whois" program to find out more information from the whois server, whois.radb.net.) Furthermore, after we animated the data only related to AS-15412, we found something interesting: AS-15412 not only injected thousands of CSM-type OASC events on April 6, but also, from April 7 to 12, it introduced thousands of yellow-colored lines, which are CMS-type OASC events. This indicates that, right after the CSM mistakes on April 6, 2001, the system administrator responsible for the AS-15412 problem started to "correct" the problems by withdrawing the bad announcements, which caused a storm of CMS events during next 6 days. The trouble AS was a small ISP (Internet Service Provider) called FLAG Telecom Global Internet in London, UK.

However, a few days later, on April 18, 2001 (4 days later), AS-15412 caused (at least, it seems to us visually) exactly the same mistake again, and the "shape" is exactly the same as the one on April 6 (possibly reloaded an old copy of BGP configuration file). The difference though is that this time it only took them one day to fix all the problems. The two pictures on the left below show the CSM-type OASC events on April 18, 2001, while the right two pictures are CMS-types events on April 19, 2001.



Please note that, via this example, we again observe the big advantage of integrating machine and human intelligence. From human's point of view, the correlation between CSM and CMS events is very clear. After watching the animation and interactively identifying the AS causing the problem, we will not consider, for example, the large number of CMS events on both April 12 and 19 as errors probably because we know AS-15412 over at UK was trying to fix the problems they created. On those two days (April 12 and 19 in 2001), without the right correlation as we shown here, thousands of false alarms might have been reported.

### 5.3 How about France Telecom?

When we first studied the instance in April 6-19, 2001, we believed, visually, that AS-15412 made "exactly" the same mistake on both April 6 and 18 of 2001 because the shapes visually look exactly the same. We reached this conclusion based on purely visual correlation. Earlier this year (2003), one of our colleagues from France Telecom came to us with their own AS numbers (AS-3215 and AS-5511). They would like to find out how many OASC events were related to their own ASes. While we quickly realized, using our Elisha BGP visualization tool, that France Telecom's ASes have been affected in relatively small number of OASC events, we also found out that France Telecom's ASes were not affected on April 6, but they were indeed part of some OASC events on April 18. This implies that, although the graphs on 6$^{th}$ and 18$^{th}$ look the same, they are different in a minor ways. It turned out that only a very small portion of ASes behave differently between 6$^{th}$ and 18$^{th}$.

## 6 Remarks

In a large complex system, it is impossible to rely on any single mechanism, however powerful may it be, to detect all possible attacks or faults. It is also very difficult to pre-design and pre-implement a set of mechanisms to detect and respond to problems not being seriously considered before. However, while human intelligence (such as the security instance response team) can certainly complement an intrusion detection system, we need to have an effective interaction process to follow in order to resolve problems correctly and quickly.

With a traditional 2D representation (i.e., counting BGP/OASC events), relatively little information can be derived and we need to dig into the raw BGP data to analyze the problem instances. For the instance of August 14, we might be able to marginally spot the anomaly. But, with our visualization program, we can not only detect the H-type OASC anomaly but also go deep into the information using different representation methods with only a few clicks to identify the OS-type OASC anomaly as well. As a result, we identify the possible error made by the AS-7777. Please note that our system was designed and built BEFORE we were aware of the AS-7777 instance in August 14, 2000 or other similar instances.

In handling millions of events from a large complex distributed system such as Internet, "false alarms" and "event correlation" become two most critical issues (or technical bottlenecks). Our visual representation for the OASC events here provides a global and abstract picture about the BGP/OASC activities in the Internet. Therefore, the human operator will be given not only a huge set of events but also the context and the relations among the events graphically. An experienced human operator can then use our system to justify the validity of a reported attack instance based on his/her comprehensive awareness of the target system. On the other hand, if he/she is not certain about the situation, then the interactive process should guide him/her to navigate more information to reduce the potential false positive. Second, as demonstrated in Section 6.2, our system provides the capability of visual event correlation such that

a human operator can quickly correlate a set of reported events and provide a valid explanation about what is going on.

In the 2D counter-type figure (Figure 2 in Section 3), we can see big spikes in April 2001, but to completely understand what was going on is a very difficult task. If not correlating events correctly, the system administrator would have to digest more than tens of thousands of events over a two-week period. And, hopefully some correct abstraction of these events can be discovered. But, again, with our program's animation features, the right, short/compact, and abstractive conclusion can be drawn quickly for all these events.

Via the experience in using our programs to analyze the BGP routing data on the Internet (we have another visualization program for BGP route path stability, which has not been described in this paper), we have demonstrated the great potential in applying information visualization techniques to critical problems in fault and intrusion detection on network routing protocols such as BGP. We believe that the integration of human and machine intelligence via the technique of information visualization may provide yet another important avenue to enhance the performance, security, and fault tolerance of the Internet.

## Acknowledgements

This research is supported in part by DARPA (under the FTN program) and NSF under Grant No. 0220147. We appreciate valuable information and comments regarding the OASC problem from Herve Debar (France Telecom), Jason Coit (UC Davis), and anonymous reviewers of this paper.

## References

[1] Dedre Gentner and Albert L. Stevens (editors), “Mental Models”, Cognitive Science, 1983.

[2] Y. Rekher and T. Li, “A Border Gateway Protocol 4 (BGP-4)”, rfc1771, IETF.

[3] James Cowie, Andy Ogielski, BJ Premore and Yougu Yuan, “Global Routing Instabilities during Code Red II and Nimda Worm Propagation” NANOG, 09/19/2001.

[4] Lan Wang, Xiaoliang Zhao, Dan Pei, Randy Bush, Daniel Massey, Allison Mankin, S. Felix Wu, Lixia Zhang, “Observation and Analysis of BGP Behavior under Stress”, by in ACM SIGCOMM IMW (Internet Measurement Workshop), Marseille, France, November 2002.

[5] Stephen Kent, Charles Lynn, and Karen Seo, “Secure Border Gateway Protocol (Secure-BGP)” in IEEE Journal on Selected Areas in Communications Vol. 18, No. 4, April 2000, pp. 582-592.

[6] Dan Massey, Lan Wang, Xiaoliang Zhao, Dan Pei, Randy Bush, Allison Mankin, Felix Wu, Lixia Zhang, “Protecting the BGP Routes to Top Level DNS Servers” in NANOG 25, June, 2002, Toronto, Canada.

[7] Ivan Herman, Guy Melançon, M. Scott Marshall, "Graph Visualization and Navigation in Information Visualization: a Survey" in IEEE Transactions on Visualization and Computer Graphics, Vol. 6, No. 1, pp. 24-43, 2000.

[8] John Hawkinson and Tony Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)" rfc1930, IETF.

[9] X. Zhao, D. Pei, L. Wang, L. Zhang, D. Massey, A. Mankin, S. F. Wu, "Detection of Invalid Route Announcement in the Internet" in International Conference on Dependable Systems & Networks, 2002.

[10] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S.F.Wu, L. Zhang, "An Analysis of BGP Multiple Origin AS (MOAS) Conflict" in ACM SIGCOMM Internet Measurement Workshop, pp.31-35, November 1-2, 2001, San Francisco.

# Context Driven Access Control to SNMP MIB Objects in Multi-homed Environments

R. State, O. Festor, and I. Chrisment

INRIA-LORIA
615, Rue du Jardin Botanique, 54600 Villers-les-Nancy, France
{state,festor,ichris}@loria.fr

**Abstract.** The advent of multi-technology networks offering the service continuum over multiple network infrastructures implies new challenges to integrated management. One of these challenges is the auto-configuration of the management plane needed to allow dynamic relationships among several managers and one management agent. This paper proposes the use of provisional policies in order to dynamically auto-configure the access control part of a management agent. This allows simple management based on agent location and time as well as the cooperative behavior of several managers.

## 1 Introduction

Recent advances in providing multiple services over heterogeneous networks demand a new approach for constructing the management plane. The management paradigms used in today networks are based on the manager agent model (designed in the mid 80') which defines a protocol for exchanging management information, specified according to an information model and stored in a conceptual repository, called Management Information Base (MIB).

In this approach and for the last decade, the management plane was configured in a static manner. For instance a basic configuration of the management protocol established at agent boot time is always assumed to exist. Such an assumption does not hold any more when we are trying to manage nomadic equipment, multi-homed sites and dynamic service infrastructures. In this paper we address the issue of providing a management approach which extends the SNMPv3 framework allowing the dynamic configuration of the access control to MIB objects using a context specific access control. The automatic configuration of the management plane is an essential step towards fully integrated management. Including self-management features within the management plane is important for plug and play type of management, where minimal human interactions are requested. On the other hand, existing management frameworks should be easily extended/integrated without demanding a total conceptual and implementation change. Our paper is structured as follows: section 2 describes the business case and motivates our work. An introduction to provisional authorizations and their usage for the configuration of the management stack is given

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 169-180, 2003.

in section 3. The management framework is described in section 4, while pointers to related work are given in section 5. Section 6 concludes the paper by highlighting future work.

## 2 The Business Case for Self-Configuration of MIB Access

This section presents two simplified business cases (see figure 1) motivating the potential of the self-configuration of the management stack. In this first case, Bob owning a WIFI enabled laptop uses his laptop both at home and work. While Bob is at the office, his laptop is under the management responsibility of the enterprise management platform (EMP). For instance, Bob might not even be allowed to auto-administer his laptop. However, as soon as Bob leaves the office and gets home, the same laptop which is connected to the home network should be considered under the management responsibility of the home management platform (HMP). At this moment, the enterprise management platform should not be allowed to perform any management operations on Bob's laptop. An extension to the previous case consists in adding some additional constraints. Bob's laptop is owned by the enterprise. If Bob is working at the office then his laptop is under the total control of the enterprise management applications. However, when Bob connects his laptop to his home network, the home network management application might perform management operations if and only if the enterprise network management platform approves it.

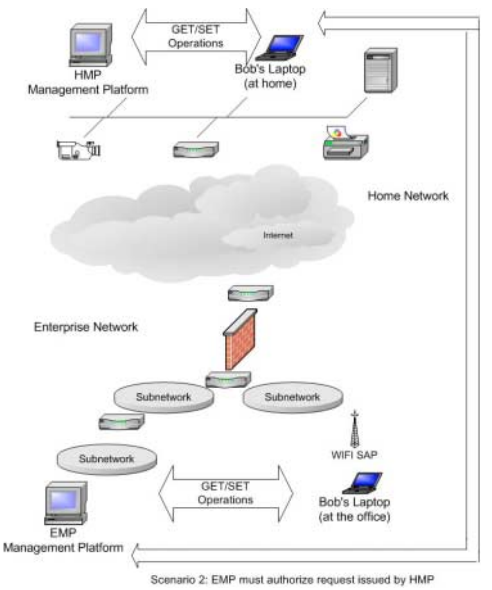



**Fig. 1.** Simplified Business Case

Looking at the two scenarios in order to abstract the fundamental requirements we can claim the following requirements:

1. Management needs to be context driven. A context can be defined as the overall ensemble of parameters characterizing the instant connectivity. For instance the context can be the collection of network interfaces, their IP addresses, netmask, and the DNS used.
2. Dynamic interactions among several managers need to be supported. It should be possible to a manager to approve or deny operations performed between an agent and another manager, without requiring direct manager to manager communication.

Does the current management SNMP framework meet these requirements? The answer is definitely not. For instance, the most advanced SNMP version (SNMP v3) [6] [17] allows to offer several views and access rights to the MIB, but these ones are defined statically independent of the context. This is done using an enhanced access control module, also known as the View Based Access Control Model (VACM) [6]. In the case of the simple scenario presented above, using the current VACM model defined by SNMPv3, would imply that the EMP could perform some management operations even when Bob is connected from home. This is based on the assumption that Bob will not manually configure the VACM every time he connects to a network. The extended scenario (where the EMP must authorize management requests performed by HMP) is definitely more complicated and can not be implemented with current management paradigms. In the following section we will introduce provisional policies and show how an extension of the current VACM can be based on provisional authorizations in order to meet the requirements. Two more additional requirements must be addressed:

1. Backward compatibility with SNMP [5].
2. Whatever works right now, should also work in the future. That is, existing management applications should be valid and fully working.

## 3 Provisional Authorizations for Dynamic Access Control to MIB Objects

Provisional authorizations have been introduced in [7] and [8] as a solution towards additional semantics for controlling access requests. Traditional access control systems considered that access requests can be modeled as a demand to authorize a particular action (read/write/delete) made by a subject within a given context on a particular object. A provisional authorization is a generalization of this scheme, modeling a conditional authorization, ie, an access request is granted if and only if additional conditions hold. This approach can be applied at the agent side. MIB objects are entities to which read/write requests are made. Provisional authorizations are stored on the agent and regulate the access to the MIB objects. The provisional framework for management can be described as follows:

1. The set S represents all possible subjects requesting access to objects. In our case, these are all managers: For instance in terms of the previously described simple scenario $S = \{EMP, HMP\}$
2. The set of all possible objects is O. This corresponds to all OIDs in the MIB.

3. A is the set of access modes permitted on objects. $A = \{get, set\}$
4. There is the notion of context, c, representing time and location. The set of contexts is C. A particular context is given by the collection of ipaddress/netmask/DNS used on each network interface. In this paper, we will use the term context having this definition in mind. The VACM aware reader should not confuse this term with the one used to specify a MIB view in SNMPv3.
5. A permission is either to grant or to deny access. The set of permissions is Perm = {grant, denied}
6. A set FM of formulas. An individual formula f is a logical conjunction of equalities and/or inequalities. For instance t>18 means current time after 18h. An equality or inequality is constituted of constant terms (string or numbers) as well system accessible variables. System accessible variables are all variables in the MIB. Other system specific variables might also be presented. If a formula holds, then the authorization policy is active (see in the following for the definition of an authorization policy). Examples of formulas could be :

   - "`1.3.6.1.2.1.ip.ipInAddrErrors>1000`", allowing a manager to get/set values in a subtree, whenever the number of datagrams errors due to address errors is higher than a threshold. While visiting a foreign network, Bob could allow management access to his machine, if and only if it's starting to have functioning errors.
   - "`1.3.6.1.2.1.interfaces.ifTable.ifEntry.ifIndex.1.ifOperStatus==down`". This formula expresses the fact that the first network interface of an agent's node does not work. For a multihomed node, one can use this formula to allow a manager connecting via the second interface to fully manage the agent if the first network interface is not working. As a typical business case, let us consider an access router with two interfaces, used by a home network. Two managers do exist. The first one belongs to the Internet Service Provider, while the second one is used on the home network. Such a formula could be used to allow the home manager full control over the access router whenever no connection to the ISP's manager is possible.

7. A set of provisional actions PRV. A provisional action is used to add new semantics on an authorization policy. We provide one single extension *verify*, meaning that another subject agrees with this operation. For instance *verify(HMP)* means that HMP has to agree with a particular request in order to authorize this access. We can model the agreement of several managers using a list of provisional actions. Several comma separated verify clauses are equivalent to a conjunction of authorizations. All subjects must authorize the request. Semicolon separated list of provisional actions represent the fact that at least one entity must authorize the request. These types of actions are used in situations when a subset of managers must all agree on a set of actions. The second case is used when any manager out of a subset has to authorize a request.

The configuration of the access control scheme is based on providing a set of authorization policies : $Auth\,\mathrm{Pr} \subset O \times S \times A \times C \times Perm \times FM \times PRV^*$

An authorization policy, for instance is:

$$(1.3.6.1.*, EMP, get,$$
$$context(address = 194.224.3.23, netmask = 255.255.255.0, dns = 195.224.3.1),$$
$$grant, time > 18, verify(HMP)) \in AuthPR$$

modeling the fact that : Manager HMP is allowed to read all elements under the subtree 1.3.6.1, if the agent is connected with IP address 194.2234.3.23, netmask 255.255.255.0 and DNS server 194.224.3.1, and current time is past 18h, ***provided that EMP agrees***.

An authorization request is a 4 tuple : $ar = (o, s, a, c) \in O \times S \times A \times C$ meaning that action a in context c is to be performed by s on object o. For the previous example, such a request might look like:

$$ar = (1.3.6.1.*, EMP, get,$$
$$context(address = 194.224.3.23, netmask = 255.255.255.0, dns = 195.224.3.1))$$

This request models the read operation on OID starting with 1.3.6.1, performed by manager EMP, where the current connectivity configuration of the laptop is given by the address, netmask and DNS. For every authorization request, an authorization decision is computed based on the set of authorization policies. Basically, a decision is to either grant or deny an action. Allowing an action can be either unconditionally or conditionally. A conditional allow is associated to a list of provisional actions. These actions must be performed in order to fully allow the operation. An example based on the previous scenario is the following decision:

$$decision = (1.3.6.1.*, EMP, get,$$
$$context(address = 194.224.3.23, netmask = 255.255.255.0, dns = 195.224.3.1),$$
$$grant, verify(HMP))$$

This decision allows the access for EMP to read all objects in the subtree 1.3.6.1, if the agent is on a network (ip address of the agent=194.224.3.23 with netmask 255.255.255.0) if and only if HMP agrees.

The provisional action can include several actions. "*verify(HMP), verify(Bob)*" is equivalent to both Bob and HMP must agree with this decision, while "*verify(HMP);verify(Bob)*" means that at least one of them must agree. This framework is a generalization of the SNMPv3 VACM architecture. However, this generalized approach allows us to model context driven and conditional management, as well as more advanced interactions between managers as described in the two business cases. Another applicability of this framework lies in providing a potential solution towards autonomous management.

For the moment, we postpone the presentation of the management architecture needed to support this authorization framework, and focus on the semantic expressiveness of this model.

Let us first see if the two described scenarios can be implemented by the model. Obviously, for the fist scenario, we can use the following four authorization policies:

1. $(*, EMP, get, context(address = 152.224.3.23, netmask = 255.255.255.255, dns = 152.224.3.1), grant, null)$
2. $(*, EMP, set, context(address = 152.224.3.23, netmask = 255.255.255.255, dns = 152.224.3.1), grant, null)$
3. $(*, HMP, get, context(address = 194.224.3.23, netmask = 255.255.255.255, dns = 195.224.3.1), grant, null)$
4. $(*, HMP, set, context(address = 194.224.3.23, netmask = 255.255.255.255, dns = 195.224.3.1), grant, null)$

Policies 1 and 2 state that the enterprise management platform in allowed to get/set any object in the MIB as long as Bob's laptop is on the enterprise network. Policies 3 and 4 model the equivalent for the home network. Obviously, dealing with dynamic allocated addresses is done by using wildcards in the specifications. For this scenario, no formulas are requested. It's also easy to see that Bob's terminal cannot be managed by his office manager, when Bob is at home.

The second scenario, in which Bob's enterprise manager must agree with the home management application, is already presented when we have introduced the provisional authorizations.

Without detailing an implementation specific verification mechanism, let us consider the following authorization policy:

$$(1.3.6.1.4.1.2, *, read, context(address = *, netmask = *, dns = *), grant, null, verify(Bob))$$

This policy models the fact that Bob must be asked by the management agent whenever a get request is made on the subtree 1.3.6.1.4.1.2. Such flexibility is interesting in the management of mobile devices, in which users could be prompted to explicitly authorize operations performed on their terminals (like for instance reading their configuration/agenda by a foreign management application). It could be considered as an user-interactive SNMP agent.

Finally, it's natural to ask if such an authorization policy based approach can be used for total self management. Without pretending to have a clear definition of total self management, we argue that at least it might give you illusion of self management. Imagine the very simple case of one authorization policy:

$$(*, *, *, context(address = *, netmask = *, dns = *), grant, null, verify(manager))$$

The intelligence or auto-configuration feature is actually hidden by the existence of a manager who authorizes every request. An agent using such a policy can be easily plugged into any network and provide the illusion of being auto-configurable. The auto-configuration in this case is actually outsourced and delegated. This is obviously an extreme case, having a lot of overhead in terms of management communication, but it shows that provided the existence of an authorization manager and a convenient set of authorization policies, a relative degree of autonomy can be achieved. Such a

mechanism is very useful for the management of nomadic equipment. The management platform to which the equipment belongs can specify limited management operations allowed whenever the equipment is not on the home network and might agree to extended management provided it is consulted. As far as we know, no previous management framework is capable of similar features.

## 4 Implementation Framework

This section describes the management framework based on the authorization policies. One of the requirements was not to depend on a new management protocol. Taking SNMPv3 as a major building block, our approach consists in providing a new Access Control subsystem and an optional extension within an SNMP agent. Figure 2 (adapted from [6], [19]) shows where the new access control subsystem and the optional extension fit into the block functional architecture of an SNMP agent. One issue that was not addressed in this paper yet relates to the dynamic interactions among an agent and one or several managers. The issue is how do managers and agents discover reciprocally, and how to provide a security model for such a framework.

The existing security model defined in SNMPv3 is implemented in the USM (User Security Model) [18] allowing the authentication of the manager as well as the optional privacy of the communication. For the authentication and privacy, two secret keys are shared between the agent and the manager. The first one is used to authenticate the manager, while the second one is used to encrypt/decrypt the SNMP operations. While the existence of these two keys can be assumed by an off-line exchange among the managers, we consider that full autonomy for the management plane is based on a dynamical manager to agent secret key exchange. Figure 2 shows two additional components and their interactions. The first one is a Manager Discoverer. Its main functionality is to discover network managers. Several choices are possible:

1. The manager learns through topology monitoring/DHCP the existence of a new node, and checking UDP port 161 on the node, discovers the agent.
2. The address of the manager is included in an extension of the DHCP configuration 10. An IETF proposition [15] suggests the use of DHCP to detect a list of IP addresses to which notifications have to be sent.
3. The agent uses the DNS to look for a manager located in the same domain.
4. A management service containing the description of the manager location is advertised over SAP (Session Announcement Protocol) [12].
5. The agent discovers via the Service Location Protocol (SLP) [11] the manager.

The information about the manager contains also a public key *Public(M)* on which the latter is listening. The manager uses the RSA (Rivest, Shamir, Adleman algorithm) [13] with the public key *Public(M)* and an associated private key *Private(M)*.

As soon as a new manager is discovered, two secret keys *authk, privk* are generated by the agent and stored in the MIB of the agent. Although, the existing SNMPv3 standard discourages the storing of secret keys in the MIB, we consider that proper access control performed by the agent can assure the required security. The Manager obtains the two keys by issuing a Get Request. This Request must be encapsulated in

a SNMPv3 packet having *msgSecurityModel* equal to 4. This field describes the security model to be used by the agent. Normally, 1 is used for SNMPv1, 2, SNMPv3 uses the value of 3 and SNMPv2c uses 2 [6].

The value 4 corresponds to our PK security model. This PK security model checks the authenticity of the issued requester using the manager's public key *Public(M)*. In fact, our extension provides a possible public key exchange facility to a SNMPv3 agent. The assumption is here, that the manager discovery process detects a genuine trusted service.

If authentication is valid (this process is based on the fact that only the legitimate manager has the key used for encryption: *Private(M)*), the two keys (*authk, privk*) are sent to the manager. The response is encrypted using the public key of the manager: *Public(M)*.

The next following interactions between the manager and the agent are performed using the traditional security model (*msgSecurityMode*l=3), which is based on symmetric keys (*authk, privk*) and therefore more efficient.

The second extension consists in a new Access control subsystem, which is used by a SNMP engine to check that particular request is authorized.

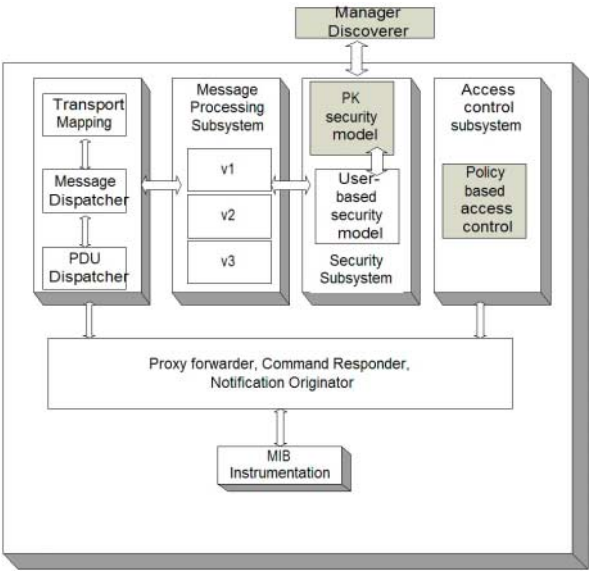



**Fig. 2.** SNMP agent architecture

A functional block decomposition of the Authorization based Access control is shown in figure 3.

Authorization Policies are stored in a Policy store. This store is implementation specific. A policy management console pushes authorization on this store. This configuration can be either based on SNMP, similar to the VACM configuration in SNMP3. In this case an extended MIB containing tables corresponding to the formulas and the provisional actions are needed. Note, that for the address and netmask variables used in the context, the existing MIB2 entries can be used.

Another configuration of these policies can be done using COPS-PR [4]. One of the advantages of COPS-PR lies in its working over TCP. If the self-management capable agent is connected to a distinct network then the one containing the policy console, most firewalls will drop SNMP traffic, leaving however TCP connections originating from the agent.

The Policy Validator is responsible to check the coherence of deployed policies. Its main objective is to resolve conflicting authorizations.

A Request Processor is the entity where an access request is made. The request processor uses the policy store and computes the authorization decision. If access is either granted without any provisional actions, or denied, this decision is returned to the SNMP engine.

To summarize, authorization policies are configured in the Policy Access Control module of the SNMP engine. When such a policy is triggered, an authorization might be requested from an authorization manager. This request can be implemented via COPS, in which a explicit accept decision is requested from a PDP. Therefore, the concept of policy is twofold. We have authorization policies regulating the access rights to the SNMP agent and we have authorization requests and replies triggered by an authorization policy.

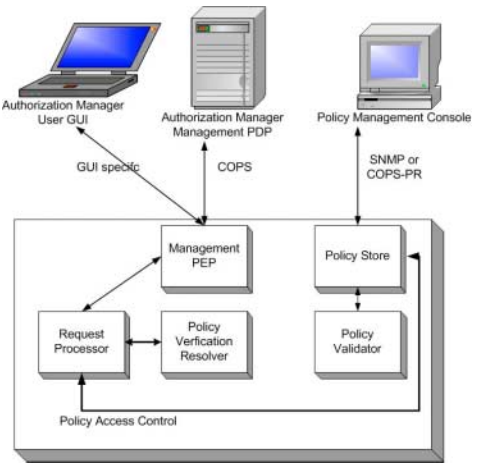



**Fig. 3.** Functional Architecture of the Access Control Subsystem

One interesting example consists in a management approach for a multi-homed host. This case, (shown in figure 4) assumes that both managers trust a third manager T. The latter is responsible to assure that requests issued by one manager do not alter/expose sensible management information. For requests issued to particular OIDs, T must explicitly approve them. The assumption that a common trusted manager exists might be relatively optimistic. If this assumption does not hold, both A and B should be asked to allow these requests. This can be expressed in a provisional action might be "*verify(A), verify(B*)". This means, that both manager have to approve this request. Obviously, this implies an overhead, but provides a solution for either not disclosing A's confidential monitoring information to B, or allowing B to configure A's related entities.

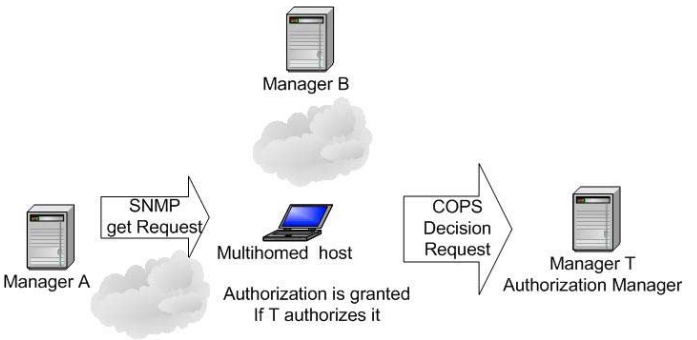



**Fig. 4.** Management of a Multi-homed host

To see the usage of COPS for authorization checking, assume that A issues a Set on a OID and that the Control Subsystem decides to grant it, provided T confirms.

The Control subsystem issues a COPS Request Message [3] to T. The COPS-context in this message is set to "Set", (it would have been "Get" for a Get Request). Additional COPS-type objects in this message, called *ClientSI* (client specific information object) represent the OIDs and the *snmpEngineId* of the requester.

If T decides that the request is granted, it replies with a COPS-Decision having a command INSTALL. From now one, whenever A issues the same request, (within the same context and formulas still holding), this is granted by the agent. If after some time, T issues a Remove Decision, the grant decision of the agent, must be re-confirmed as previously explained.

## 5 Related Work

We started our work based on the general management agent security configuration proposed in the SNMPv3 [6], [19] specifications. The View Access Control Model [17] allows the definition of view and allowed operations for a set of managers. Its user security model USM [18] provides the supporting mechanisms for manager-agent privacy and authentication. These mechanisms must be configured statically and are difficult if management of nomadic or multi-homed equipments has to be performed. These approaches are extended by our work with context driven and conditional management as well as a security model configurable dynamically. One of the first initiative towards simplified and automatic host configuration was started by the zero-configuration group at the IETF [9], where individual (private network) addresses are automatically assigned without requesting static configuration or DHCP [10] support. Policy based network management has been applied for QoS management in both Diffserv as well as Intserv [1] types of networks, without however addressing the self-management issue. A policy based approach for the auto-configuration of networks is proposed in the Nestor [14] platform used to manage an active network platform. Change operations are managed via policy rules and integrated within a larger network self configurability architecture. The research community working in access control mechanisms proposed a large variety of security ar-

chitectures and policy specification methods. However, no direct applications towards extending the management functionality based on these concepts have been proposed. Our approach is different with respect to the previously mentioned work in several aspects. We start with the main objective to use SNMPv3 the standard management protocol in an autonomic way. We propose a novel architecture for an SNMP agent capable to integrate within existing deployed networks and without requiring a complex additional infrastructure at the network site. We have not yet decided to use a specific policy specification framework. Our current prototype uses a proprietary solution. We are looking into applying the PONDER specification framework [16] for this purpose. PONDER provides an extremely powerful language which could be capable to express management authorizations as well as context related information.

## 6 Conclusions

We addressed in this paper the issue of flexible network management, respectively self-management. We started with the observation that current standardized network management frameworks do not offer enough support for enhanced agent autonomy. The simple observation is that an already configured SNMPv3 agent, taken out from his home network and put under the management control of one or several foreign management applications without any additional human interaction is not functional. This is due to several factors. Firstly, a fixed security Model (USM) and a fixed Access Module (VACM) are incapable to configure on their own. A second factor, which is more conceptual, is related to the existing management paradigm, in which a manager interacts directly with an agent. We extend this paradigm by allowing other parties (managers) to express their agreement within such an interaction. We provide a framework allowing context/location driven management and conditional management. We argue that current SNMPv3 specifications provide an excellent authorization and access control mechanism for a fixed environment. This is the case with most existing target environments. However, nomadic environments where more and more users are mobile require new management frameworks. We addressed this issue by assuming the standard management protocol SNMPv3 and we proposed an extension of the SNMPv3 management framework. A new security model and an access control are proposed. The proposed security model allows the exchange of the authentication and privacy keys. These keys are used by the SNMPv3 user security model. Our management framework is based on authorization policies and provisional actions in which context driven management can be performed. The context is defined by the network connectivity properties used on the managed equipment site. Our approach generalizes the View based Access Control subsystem proposed in the SNMPv3 architecture, without requiring changes at the SNMP protocol level. Thus, our extension is transparent for already existing management applications. Existing management agents should be easily modified in order to enable dynamic access control to MIB objects. We consider that self-management is strongly related to the auto-configuration of the management plane. Management of mobile devices as well as environments where multiple management applications interact dynamically are the primary immediate business targets. Our approach is based on managing the management stack within a policy based solution. This integration shows also the com-

plementary nature of these two types of management and motivates the necessity of having both of them within a single management stack. We are currently implementing the proposed architecture within a Net-SNMP framework. It will be validated within our IPv6 testbed.

## References


[1] D. Verma. Policy-Based Networking. New Riders Publishing.2000.
[2] RFC 3159. Structure of Provisioning Information (SPPI). IETF. 2001
[3] RFC 2748. The COPS (Common Open Policy Service). IETF. 2000
[4] RFC 3084. COPS Usage for Policy Provisioning (COPS-PR). IETF.2001.
[5] W. Stallings. SNMP, SNMPv2, SNMPv3 and RMON1 and 2 Addison-Wesley Pub Co; 3rd edition 1998)
[6] W. Stallings. Network Security Essentials, Prentice Hall, 2nd edition 2002.
[7] M. Kuda , S.Hata. XML Document security based on provisional authorization Proc. 7th ACM Conference on Computer and Communication Security (CCS 2000), Nov. 2000.
[8] S. Jajodia, M. Kuda, V.S. Subrahmanian. Provisional authorizations. Workshop on Security and Privacy in E-Commerce (WSPEC), Nov. 2000, Recent Advances in Secure and Private E-Commerce, published by Kluwer Academic Publishers in 2001.
[9] E. Guttman. Autoconfiguration for IP Networking: Enabling Local Communication, IEEE Internet computing. 2001.
[10] R. Droms. "The DHCP Handbook". Sams. 2nd edition. 2002.
[11] RFC 2608. Service Location Protocol , Version 2. IETF 1999.
[12] RFC 2974. Session Announcement Protocol. IETF 2000.
[13] B. Schneier. Applied Cryptography: Protocols, Algorithms and Source Code in C, Second Edition. John Wiley and Sons. 1995
[14] V. Konstantinou, Y. Yemini, and D. Florissi. Towards Self-Configuring Networks. DARPA Active Networks Conference and Exposition (DANCE), May 2002, San Franscisco, CA.
[15] M. Bakke. DHCP Option for SNMP Notifications. draft-bakke-dhc-snmp-trap-01.txt. Internet draft IETF. Work in progress 2003.
[16] N. Damianou. A Policy Framework for Management of Distributed Systems. Ph.D thesis. Faculty of Engineering of the University of London and Diploma of the Imperial College of London. London. December 2002.
[17] RFC 3415. View Based Access Control Module (VACM) for the Simple Network Management Protocol. IETF 2002
[18] RFC 2274. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). IETF 2002
[19] RFC 3411. An architecture for describing simple network management protocol (SNMP) Management Frameworks. IETF 2002

# A Policy-Based Framework for RBAC

Ricardo Nabhen, Edgard Jamhour, and Carlos Maziero

Pontifícia Universidade Católica do Paraná, PUCPR, PPGIA
Rua Imaculada Conceição 1155, CEP 80215-901, Curitiba, Brazil
{rcnabhen,jamhour,maziero}@ppgia.pucpr.br

**Abstract.** This paper presents a PCIM-based framework for storing and enforcing RBAC (Role Based Access Control) policies in distributed heterogeneous systems. PCIM (Policy Core Information Model) is an information model proposed by IETF. It defines a vendor independent model for storing network policies that control how to share network resources. PCIM is a generic core model. Application-specific areas must be addressed by extending the policy classes and associations proposed by PCIM. In this context, this paper proposes a PCIM extension, called RBPIM (Role-Based Policy Information Model), in order to represent network access policies based on the RBAC model. A RBPIM implementation framework based on the PDP/PEP (Policy Decision Point/Policy Enforcement Point) approach is also presented and evaluated.

## 1 Introduction

The Policy Core Information Model (PCIM) is an information model proposed by IETF and DMTF [5]. PCIM is based on the Common Information Model (CIM), proposed by DMTF (Distributed Management Task Force) [4]. The CIM information model permits to represent both, network resources and policies definitions for managing these resources. PCIM can be considered as a sub-set of the CIM model, which comprises only the policy information. PCIM is a generic model. Application-specific areas must be addressed by extending the policy classes and associations proposed by PCIM. For example, QPIM (QoS Policy Information Model) is a PCIM extension for describing quality of service polices [12].

This paper describes a PCIM extension for role-based access control, called RBPIM (Role Based Policy Information Model), which permits to represent network access control policies based on roles, as well as static and dynamic constraints, as defined by the proposed NIST RBAC standard [1]. RBPIM is an open framework for supporting the development of applications that shares a common set of RBAC policies. In the RBPIM approach, the RBAC policies refers to objects represented in a CIM repository. The RBPIM framework is implemented using a PDP/PEP approach [10]. The PDP (Policy Decision Point) is a network policy server responsible for supplying policy information (or policy decisions) for network devices and applications. The PEP (Policy Enforcement Point) is the policy client (usually, a component of the

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 181-193, 2003.

network device/application) responsible for enforcing the policy. The motivation for defining RBAC in PCIM terms can be summarized as follows. First, there are several situations where the same set of access control policies should be available for heterogeneous applications in a distributed environment. This feature can be achieved by adopting the PDP/PEP framework. Second, an access control framework requires having access to information about users, services and applications already described in a CIM/PCIM repository. Implementing access control in PCIM terms permits to leverage the existing information in the CIM repository, simplifying the task of keeping a unique source of network information in a distributed environment.

The remaining of this paper is organized as follows: Section 2 reviews some related works. Section 3 presents the RBPIM information model. Section 4 presents the RBPIM framework implemented using the outsourcing model, as defined by the COPS standard [11]. Section 5 presents the performance evaluation results of a prototype of the RBPIM framework under various load conditions. Finally, the conclusion summarizes the main aspects in this project and points to future works.

## 2 Related Works

Recent works explore the advantages of the PDP/PEP approach for implementing an authorization service that could be shared across a heterogeneous system in a company. An interesting work in this field is the XACML (eXtensible Access Control Markup Language), proposed by the OASIS consortium [13]. XACML is a XML based language that describes both an access control policy language and a request/response language. The policy language is used to express access control policies. Policies are written in XACML by policy administrators and made available for PDP servers. The request/response language is used for supporting the communication between PEP clients and PDP servers. A PEP queries a PDP whether a particular access should be allowed using XACML, and the PDP describes answers to those queries also using XACML. The RBPIM framework described in this paper also uses the PDP/PEP approach. However, our approach differs from XACML from several points. First, the RBPIM uses a standard COPS protocol for supporting the PEP/PDP communication. Second, the information model used for describing policies is based on a PCIM extension. Third, RBPIM has been implemented to support a specific access control method, the RBAC. That permits to define a complete framework that includes the algorithms in the PDP especially conceived for evaluating policies that includes hierarchy of roles and both, dynamic and static separation of duties.

Most of the research efforts found in the literature refer to the use of the PCIM model and its extensions for developing policy management tools for QoS support [12]. However, a pioneer work for defining a PCIM extension for supporting RBAC, called CADS-2, has been proposed by BARTZ, L.S. [3]. The CADS-2 is a review of a previous work, called *hyperDRIVE*, also proposed by BARTZ [2]. The *hyperDRIVE* is a LDAP schema for representing RBAC. This schema can be considered as a first step for implement RBAC using the PDP/PEP approach. However, *hyperDRIVE* was elaborated before the PCIM standard, and has been discontinued by the author. As hyperDRIVE, CADS-2 defines classes suitable to be implemented in a directory-based repository, such as LDAP. In the CADS-2 approach, the permissions are ex-

pressed in terms of "having access to services". A service has subordinated objects called *ServiceAccessPoint* (SAP), which are, in fact, the protected objects in the RBAC model.

The RBIM model described in the section 3 uses the idea of mapping roles to users using Boolean expressions, proposed by the CADS-2 model. Note that this approach offers an additional degree of freedom for creating RBAC policies because the relationship between user and roles can be expressed through Boolean expressions instead of a direct mapping. However, the RBPIM work differs from the CADS-2 model from several points. Many features have been introduced in order to support the other elements of the RBAC model, such as hierarchy of roles, DSD (Dynamic Separation of Duty) and SSD (Static Separation of Duty), defined in the proposed NIST standard [1], but not present in the CADS-2 model. Also, in the RBPIM, Boolean expressions are built using the PCIM extensions proposed in RFC 3460, not available in the original release of the CADS-2 model.

## 3 RBPIM: The Role-Based Policy Information Model

Fig. 1 shows the PCIM model, and the proposed RBPIM extensions for supporting RBAC policies. In the PCIM approach, a policy is defined as a set of policy rules (*PolicyRule* class). Each policy rule consists of a set of conditions (*PolicyCondition* class) and a set of actions (*PolicyAction* class). If the set of conditions described by the class *PolicyCondition* evaluates to true, then a set of actions described by the class *PolicyAction* must be executed. A policy rule may also be associated with one or more policy time periods (*PolicyTimePeriodCondition* class), indicating the schedule according to which the policy rule is active and inactive. Policy rules may be aggregated into policy groups (*PolicyGroup* class) and these groups may be nested, to represent a hierarchy of policies.

In a *PoliceRule*, rule conditions can be grouped by two different ways: DNF (Disjunctive Normal Form) or CNF (Conjunctive Normal Form). The way of grouping policy conditions is defined by the attribute *ConditionListType* in the *PolicyRule* class. Additionally, the attributes *GroupNumber* and *ConditionNegated*, in the association class *PolicyConditionInPolicyRule* (the attributes are not shown in the Fig. 1) helps to create condition expressions. In DNF, conditions within the same group number are ANDed and groups are ORed. In CNF, conditions within the same group are ORed and groups are ANDed.

The RFC 3460 proposes several modifications in the original PCIM standard. These modifications are called PCIMe (Policy Core Information Model Extensions) [6]. PCIMe solves many practical issues raised after the original PCIM publication. In the PCIMe, *PolicyCondition* have been extended in order to support a straightforward way for representing conditions by combining variables and values. This extension is called *SimplePolicyCondition*. The strategy defined by *SimplePolicyCondition* is to build a condition as a Boolean expression evaluated as: does <variable> MATCH <value>? Variables are created as instances of specializations of *PolicyVariable*. The values are defined by instances of specializations of *PolicyValue*. The MATCH element is implicit in the model. PCIMe defines two types of variables: explicit (*PolicyExplicitVariable*) and implicit (*PolicyImplicitVariable*). Explicit variables are used to

build conditions that refer to objects stored in a CIM repository. Implicit variables are used to represent objects that are not stored in a CIM repository. They are especially useful for defining filtering rules with conditions based on protocol headers, such as source and destination addresses or protocol types. For supporting filtering rules, PCIMe defines several specializations of *PolicyImplicitVariable*, such as *PolicySourceIPv4Variable*, *PolicySourcePortVariable*, etc. Please, refer to the RFC 3460 for more details about this approach.

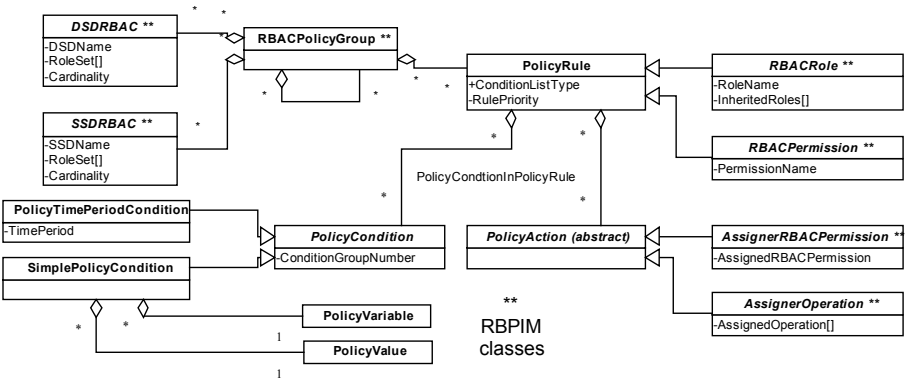



**Fig. 1.** PCIM/PCIMe class hierarchy and RBPIM extensions

The RBPIM class hierarchy is also shown in the Fig. 1. The following classes have been introduced: *RBACPermission* and *RBACRole* (specializations of *PolicyRule*), *AssignerPermission* and *AssignerOperation* (specializations of *PolicyAction*), *DSDRBAC* and *SSDRBAC* (specializations of *Policy* , not shown in the figure). The *RBACPolicyGroup* class (specialization of *PolicyGroup*) is used to group the information of the constrained RBAC model. As shown in Fig. 1, the approach in the RBPIM model consists in using two specializations of *PoliceRule* for building the RBAC model: *RBACRole* (for representing RBAC roles) and *RBACPermission* (for representing RBAC permissions). *RBACRole* and *RBACPermission* follows the same semantic proposed by the standard class *PoliceRule*, i.e., if the set of the conditions associated to the class are evaluated true, than the corresponding set of actions are executed. This approach offers a flexible method for representing the relationship between user and roles and between roles and permissions using DNF or CNF expressions. *RBACRole* can be associated to lists of *SimplePolicyCondition*, *AssignerRBACPermission* and *PolicyTimePeriodCondition* instances. The instances of *SimplePolicyCondition* are used to express the conditions for a user to be assigned to a role (UA relationship). The instances of *AssignerRBACPermission* are used to express the permissions associated to a role (PA relationship). The instances of *PolicyTimePeriodCondition* define the periods of time a user can activate a role. *RBACPermission* can be associated to a list of *SimplePolicyCondition* and *AssignerOperation* instances. The instances of *SimplePolicyCondition* are used to describe the protected RBAC objects and the instances of *AssignerOperation* are used to describe approved operation on these objects. The SSDRBAC and DSDRBAC classes permit to describe static and dynamic separation of duty constraints. The RBPIM model defines SSD and DSD with two attributes: a *roleSet[]* that includes two or more roles, and a *cardi-*

*nality* greater than one indicating the maximum combination of roles in the set a user can be assigned (SSD) or activate within a session (DSD), e.g., for constraining a user to assume the roles "r1" and "r2", one must define a set {r1, r2} with cardinality 2 (the user can assume cardinality-1 roles in the set).

The example in Fig. 2 to illustrates the use of the RBPIM model. The *RBACRole* in the figure was called "role1". The attribute *InheritedRoles* is used for expressing the Hierarchical RBAC, i.e., the role "role 1" inherits the permissions of roles "role2" and "role3". The UA relationship for "role1" is defined as:

IF "*PolicySourceIPv4Variable* MATH 192.168.10.0/24" AND "*Person.BusinessCategory* MATCH CT*" AND "*PolicyTimePeriodCondition* MATCH [20020701,20020831]".

The PA relationship is defined by the reference to the permission object "*App_Directory*", shown in the Fig. 2. This permission defines the operations {*R,W*} are approved when *Directory.Name* MATH "*/etc/application*". Observe how the use of explicit variables permits leveraging the information of existing CIM repositories. As well as PCIM, the RBPIM model is implementation neutral. RBPIM mapping to LDAP schema has been implemented according to the IETF standard PCLS [9]. Please, refer to [9] for a detailed description of the PCIM and LDAP mapping.

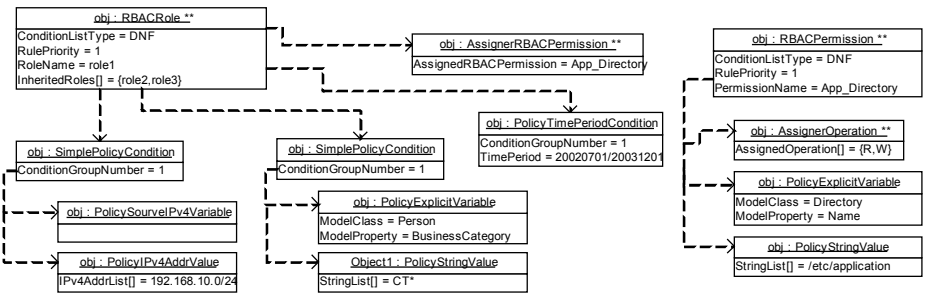



**Fig. 2.** Object instances of the RBPIM model

## 4 RBPIM Framework

The RBPIM Framework follows the PDP/PEP approach [10]. The IETF defines that the PEP and the PDP communicates using the COPS (Common Open Policy Service) protocol [11]. The COPS protocol defines two models of operation: outsourcing and provisioning. In the outsourcing model, the PDP receives policy requests from a network node (PEP), and determines whether or not to grant these requests. By the other hand, in the provisioning model, rather than responding to PEP events, the PDP prepares and "pushes" configuration information to the PEP. This takes place as a result of external events (unrelated to the PEP) such as change of applicable policy, time of day, expiration of account quota, or information from third party (non-PEP) signaling. In the provisioning approach, a PEP can answer to events based on the locally stored policy information. Fig. 3 illustrates the main elements in the RBPIM framework. RBPIM framework adopts the PDP/PEP model using a "pure" outsourcing approach, i.e., the PDP carries most of the complexity and the PEP is comparatively light.

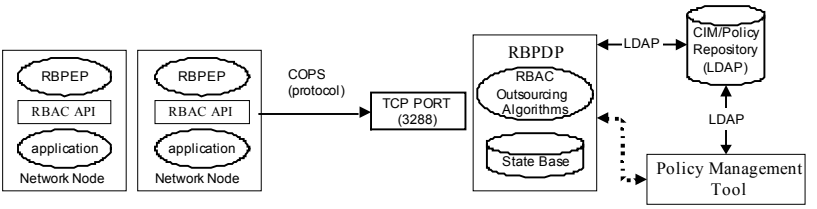



**Fig. 3.** RBPIM Framework Overview.

In the RBPIM framework, the PEP is called Role-Based PEP (RBPEP). The Role-Based PDP (RBPDP) is a specialized PDP responsible for answering the RBPEP questions. The RBPDP has an internal database (called State DataBase) used for storing the state information of the RBPEP. The CIM/Policy Repository is a LDAP server that stores both: objects that represent network information such as users, services and network nodes and objects that represents policies (including the RBPIM model described in the section 3). The Policy Management Tool is the interface for updating CIM/Policy repository information and for administrating the PDP service.

The RBPEP is basically a software library that simplifies the task of building "RBAC-aware" applications. It offers a high level programming interface for mapping the RBAC APIs to COPS messages addressed to the RBPDP. The RBAC API's used in the RBPIM framework are based on the RBAC functional specifications described in the proposed NIST standard [1]. The NIST standard defined five supporting system functions[1]: *CreateSession* (*user, session*): creates the *user* session and provides the user with a default set of roles. The *session* identifier is supposed to be generated by the underlying system. *AddActiveRole*(*user, session, role*): adds a *role* as an active role for the current *session*. *DropActiveRole* (*user*, *session*, *role*): deletes a role from the active role set for the current session. *CheckAccess* (*session, operation, object, out: result*): determines if the *session* subject has permission to perform the requested *operation* on an *object*. The *result* is BOOLEAN. *DeleteSession*(*user, session*): deletes a given session with a given owner user. Based on the NIST supporting system functions proposed by NIST, the RBPIM framework defines a set of five API's:

- RBPEP_Open ()
- RBPEP_CreateSession(userdn:string; out session:string, roleset[]:string, usessions:int)
- RBPEP_SelectRoles(session: string, roleset[]:string; out result:BOOLEAN)
- RBPEP_CheckAccess(session: string, operation:string, objectfilter[ ]:string; out result:BOOLEAN)
- RBPEP_CloseSession(session:string)

As mentioned before, RBPEP maps the RBPEP calls to COPS messages. The COPS for outsourcing model defines several messages, but the most important are REQ (Request) and DEC (Decision), which are used to encapsulate the

[1] In the NIST standard, supporting system functions refer to the process of creating a session, activating roles and checking access permissions.

RBPEP_CreateSession, RBPEP_SelectRoles and RBPEP_CheckAccess API's. The REQ message encapsulates the parameters passed by the PEP to the PDP, and the DEC message encapsulates the messages returned by the PDP to the PEP.

The RBPDP module implements a set of algorithms triggered by the COPS messages sent by the RBPEPs. These algorithms interpret the RBAC policies stored in the CIM/Policy repository and the state information of the RBPEP sessions (stored in a relational state-database), and answer the RBPEP using the COPS protocol. Note that the state-database is a database internal to the *RBPDP* and its information is not described in the RBPIM model. The RBPEP API and the respective PDP algorithms are described next in this section. The most important algorithms implemented by the *RBPDP* are those related to the *RBPEP_CreateSession*, *RBPEP_SelectRoles* and *RBPEP_CheckAccess.* Some obvious error treatment have been omitted in order to simplify the presentation of the algorithms.

***RBPEP_Open.*** The *Open* is the only API not related to RBAC. It establishes the connection between the PEP and the PDP. The API could be used by an application to ask the RBPEP to initiate the RBAC service. The RBPEP will process the API only if it is not already connected to the PDP.

***RBPEP_CreateSession.*** The *CreateSession* API establishes a user session for the user and returns the set of roles assigned to the user that satisfies the SSD constraints (*roleset*[]). This approach differs from the standard *CreateSession()* function because it does not activate a default set of roles for the user. Instead, the user must explicitly activate the desired roles in a subsequent call to the *RBPEP_SelectRoles* API. This modification avoids the need of the user to drop unnecessarily activated roles in order to satisfy DSD constraints. In order to call the *CreateSession* API, an application must specify the user through a DN (distinguish name) reference to a CIM Person object that represents the user (*userdn*). The RBPIM framework does not interfere in the authentication process. It supposes the application have already authenticated the user and mapped the user login to the corresponding entry in the CIM repository.

Because the DSD constraints are imposed only within a session, the *CreateSession* API also returns the number of sessions already opened by the user (*usessions*). The application can abort the *CreateSession* process by calling *DeleteSession,* if it does not desire to serve a user with sessions already open. Finally, the *session* parameter is a unique value generated by the RBPEP and returned to the application in order to be used in the subsequent calls. Presently, the approach defined by the RBPIM framework consists in using a *RBACPolicyGroup* object for grouping the RBAC objects. In the CIM/Policy repository, the *RBACPolicyGroup* objects are associated to "organization units" by DIT containment. By using the attribute organizational unit ("OU") in the CIM *Person* object, the algorithm determines the corresponding *RBACPolicyGroup* object associated to the user. The algorithm for the *RBPEP_CreateSession* API is defined as follows:

1. If the session already exists in the state database then returns a <Error> object in the DEC message. Otherwise, go to Step 2.
2. Determine **Ra** as the list of "candidate" roles (*RBACRoles* objects) associated to the *RBACPolicyGroup* object in the organization unit of the user.

3. Determine **Rb** as the list of RBAC roles, subset of **Ra**, which conditions are satisfied by the CIM Person object pointed by *userdn*.
4. Determine **Rc** as the list of RBAC roles, subset of **Rc**, that satisfy the time constraints defined by the *PolicyTimePeriodCondition*.
5. Determine **Rd** as the list of inherited roles indicated by the attribute *InheritedRoles* of all RBACRole objects $\in$ **Rb**.
6. Determine **Re** as the disjoint union: **Rc** $\cup$* **Rd**.
7. Determine **SSD** as the list of SSDRBAC objects associated to the *RBACPolicyGroup* object in the organization unit of the user.
8. Determine **Rf** by removing from **Re** the roles that are constrained by **SSD**. The roles with lowest priority (*RulePriority* attribute inherited by *RBACRole* from *PolicyRule*) are removed first, until the Cardinality attribute of all **SSD** constraints is satisfied.
9. Create in the state database a record with the session, *userdn*, the *roleset[]* defined by **Rf** and *status*=Phase1 and sends a DEC message with the parameters *roleset* and *usessions* encapsulated in <Decision> objects.

***RBPEP_SelectRoles.*** The SelectRoles API activates the set of roles defined by the *roleset[]* parameter. This API evaluates the SSD constraints in order to determine whether the set of roles can be activated or not. If all roles in the set *roleset[]* can be activated, the function returns *result*=TRUE.

The *SelectRoles* API, differently from the standard *AddActiveRole* function, can be evocated only once in a session. Also, in the RBPIM approach, the standard function *DropActiveRole* was not implemented. We have evaluated that allowing a user to drop a role within a session would offer too many possibilities for violating SSD constraints. The *RBPEP_SelectRoles* API activate in a session the set of roles defined by the *roleset[]* argument. The *SelectRoles* API will activate the roles only if all roles in *roleset[]* are presented in the session database and all of them are free of DSD constraints. The algorithm for the *RBPEP_SelectRoles* API is defined as follows:

1. If the session already exists in the state database with status=Phase1 go to Step 2. If it doesn't, then returns a <Error> object in the DEC message.
2. Determine **R** as the list of references to roles (RBACRoles) objects associated to the session in the state database.
3. If *roleset*[] $\not\subset$ **R** then sends a DEC message indicating the operation has been denied. Otherwise, go to Step 4.
4. Determine **DSD** as the list of DSDRBAC objects associated to the RBACPolicyGroup object in the organization unit of the user.
5. If *roleset[]* violates the **DSD** constraints then sends a DEC message indicating the operation has been denied. Otherwise, go to Step 6.
6. Update the state database by storing *roleset[]* as the list of active roles in the session and define status=Phase2. Then, sends a DEC message with *result=true* encapsulated in a <*Decision*> object.

***RBPEP_CheckAccess.*** The CheckAccess API is similar to the standard *CheckAccess* function proposed by the NIST. This API evaluates if the user has the permission for executing the *operation* on the set of objects specified by the filter *objectfilter[]*.

The *objectfilter*[] is a vector of expressions of type "*PolicyImplicitVariable=PolicyValue*" or "*PolicyExplictyVariable=PolicyValue*" used for discriminating one or more objects. In the current RBPIM version, the expressions in *objectfilter[]* are ANDed, i.e., only the objects that simultaneously satisfy all the conditions in the vector are considered for authorization checking. The explicit variables expressions are evaluated independently, and must belong to the same object class in order to avoid an empty set of objects. To consider association between the CIM classes is a complex issue let for future studies. As an alternative, a condition "*DN=value*", based on the distinguished-name of an object, can be passed in the object filter to uniquely identify a CIM object, leaving to the application the responsibility of querying the CIM repository. Explicit variable conditions may define one or more CIM objects. For example, {"*DataFile.Readable=true*", "*DataFile.Name=*.doc*"} will probably define a set of objects instead of a single object. Say $\Phi$ as the set of objects defined by the *objectfilter[]* in the *RBPEP_CheckAccess* API. The CIM objects in the $\Phi$ can be retrieved by a single LDAP query which filter is based on the *objectfilter[]* conditions. By the other hand, the *RBACPermission* objects associated to the roles activated by the user may also contain conditions based on implicit and explicit variables and, therefore, define another set of CIM objects, say $\psi$, also retrieved by a single LDAP query. The *RBPEB_CheckAccess* API will return true if $\Phi \subseteq \psi$. Because $\psi$ can be very large, the condition $\Phi \subseteq \psi$ is replaced by the equivalent expression $\Phi \subseteq \theta$, where $\theta = \psi \cap \Phi$. The $\theta$ set can also be determined by a single LDAP query, by defining a LDAP filter that combines the conditions presented in the *objectfilter[]* and the *RBACPermission* associated conditions. The implicit variables conditions such as {*"PolicyDestinationIPv4Variable=192.168.2.3*"} are not used for creating the LDAP queries, because implicit variables doest no correspond to objects in the CIM repository. Instead, they are used for eliminating the *RBACPermission* objects that does not satisfy the implicit variables in the *objectfilter[]* vector. The algorithm for the *RBPEP_CheckAccess* API is defined as follows:

1. *Verify* if the session exists in the state database with *status=Phase2*. If it doesn't than returns an <*Error*>. Otherwise, go to Step 2.
2. *Determine* **Ra** as the list of active roles in the session (references to objects *RBACRoles*).
3. *Determine* **Rb** as the subset of **Ra** where the roles satisfy the time constraints (*PolicyTimePeriodCondition* objects*)*.
4. *Determine* **Pa** as the list of permission (*RBACPermission*) objects associated to the roles $\in$ **Rb**.
5. *Determine* **Pb** as the subset of **Pa** that includes only the permission objects which implicit variables conditions satisfy the implicit variables conditions in the *objectfilter[]* vector passed by the RBPEP API.
6. *Determine* **O** as the list of operations (*AssignerOperation*) objects associated to the permission object $\in$ **Pb** .
7. *Determine* **Pc** as the subset of **Pb** that include only the permission objects where the *operation* passed by the RBPEP API $\in$ **O**.

8. *Determine* $\theta$ as the list of CIM objects that simultaneously satisfy the conditions defined by the explicit variables in the *objectfilter[]* and the list of conditions (*SimplePolicyCondition*) associated to the permission objects $\in$ **Pc**.
9. *Determine* $\Phi$ as the list of CIM objects that satisfy the explicit variables in the *objectfilter[]*.
10. S*ends* a DEC message with *result* = true if $\Phi \subseteq \theta$, otherwise, sends *result*=false.

***RBPEP_CloseSession.*** The *CloseSession* API terminates the user session, and informs to the PDP that the information about the session in the "state database" is no longer needed.

The RBPEP_APIs are currently implemented in Java, and throws exceptions for informing the applications about the errors returned by the PDP. Examples of exceptions are: "RBPEP_client not supported", "non-existent session", "*userdn* not valid", etc.

## 5 Evaluation

There are some important questions to evaluate in the RBPIM model. First, the strategy adopted for representing UA (User Assignment) and PA (Permission Assignment) relationship, based on implicit and explicit variables, is very flexible, but can lead to a long policy decision response time. Other important issue is determining if the outsourcing model is capable of a reasonable response time.

In order to evaluate the performance the RBPIM framework, a Java based RPPDP and a RBPEP scenario simulator was implemented. This prototype is available for download in [14]. In the evaluation scenario, twenty RBPEP clients request the RBPIM policy service provided by a single PDP. Each RBPEP keep a distinct COPS/TCP connection with the PDP. Several user sessions were created in the context of each RBPEP connection. In order to simulate different load scenarios, we have introduced a random delay between each API call evocated by the RBPEP client. By varying the range of the random delay, we have created six load scenarios as shown in Fig. 4. The figure also presents the results obtained using a Pentium IV 1.5 Ghz 256 Mb RAM for hosting the PDP, and other identical machine for hosting the 20 RBPEP clients. Initially, we defined a small set with five role objects hierarchically related and six permission objects, corresponding to a small set of departmental policies grouped in a single *RBACPolicyGroup* object. Each role and permission object has been defined considering a small set of three or four conditions combining implicit and explicit variables. Also, three SSD constraints and one DSD constraint were considered. The result of this simple scenario is presented in Fig. 4. One observes from the results that the RBPEP_CreateSession API correspond to the longest decision time. This is justified by the fact that this API prepares the state database by retrieving the list of the roles assigned to the user, free of SSD constrains.

After this initial test, the number of RBPIM objects has been increased. Each RBPIM object affects differently the response time of the RBPEP_APIs. Because of the flexibility introduced in the UA relationship by the RBPIM approach, the number of roles objects significantly affects the *RBPEP_CreateSession* API. Increasing the

number of roles from five to twenty has almost doubled the average response time. By the other hand, the effect of increasing the number of SSD objects is not important. The response time of other APIs are not affected because the roles assigned to the user are saved in the state database for subsequent calls. The RBPEP_SelectRoles is almost imperceptible affected by the number of DSD objects and the other RBPIM objects do not affect it. By analyzing the algorithms described in section 4, one could suppose the number of permission objects associated to the roles should affect the *RBPEP_CheckAccess*. However, our tests shown that increasing the average number of permissions per role from two to ten has no significant effect in the response time. The justification for this result is that steps 8 and 9 in the *CheckAccess* algorithm are solved with a single LDAP query that creates a complex filter combining the conditions of all permission objects. Similarly, in all APIs, increasing the number of conditions associated to a role or permission object has no significant effect, because the DNF or CNF conditions are transformed in a single LDAP query.

After this initial test, the number of RBPIM objects has been increased. Each RBPIM object affects differently the response time of the RBPEP_APIs. Because of the flexibility introduced in the UA relationship by the RBPIM approach, the number of roles objects significantly affects the *RBPEP_CreateSession* API. Increasing the number of roles from five to twenty has almost doubled the average response time. By the other hand, the effect of increasing the number of SSD objects is not important. The response time of other APIs are not affected because the roles assigned to the user are saved in the state database for subsequent calls. The RBPEP_SelectRoles is almost imperceptible affected by the number of DSD objects and the other RBPIM objects do not affect it. By analyzing the algorithms described in section 4, one could suppose the number of permission objects associated to the roles should affect the *RBPEP_CheckAccess*. However, our tests shown that increasing the average number of permissions per role from two to ten has no significant effect in the response time. The justification for this result is that steps 8 and 9 in the *CheckAccess* algorithm are solved with a single LDAP query that creates a complex filter combining the conditions of all permission objects. Similarly, in all APIs, increasing the number of conditions associated to a role or permission object has no significant effect, because the DNF or CNF conditions are transformed in a single LDAP query.

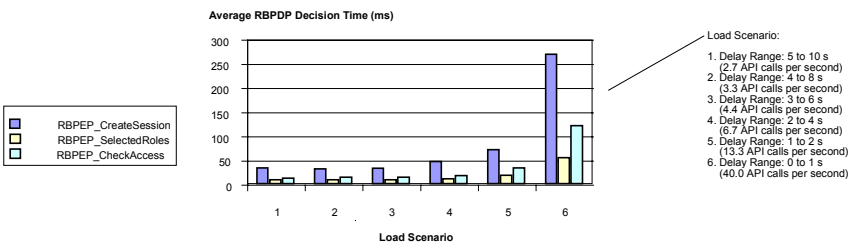



**Fig. 4.** Average Time RBPDP decision x API calls

## 6 Conclusion

This paper has presented a complete policy based framework for implementing RBAC policies in heterogeneous and distributed systems. This framework, called RBPIM, has been implementing in accordance with the IETF standards PCIM and COPS, and also, the proposed NIST RBAC standard. The framework proposes a flexible RBAC model, which permits specifying the relationship between users, roles, permissions and objects by combining Boolean expressions. The performance evaluation of the outsourcing model indicates that this approach is suitable for supporting RBAC applications that requires decisions based on user events. This paper does not discuss the problems that could rise if the PDP breaks. Future works must evaluate alternative solutions for introducing redundancy in the PDP service. Also, additional specifications are required for assuring a secure COPS connection between the PDP and the RBPEPs. These studies will be carried out in parallel with the evaluation of provisioning and hybrid approaches for implementing the RBPIM framework. Finally, some studies are being developed for evaluating the use of the RBPIM framework for QoS management based on RBAC rules.

## References

[1] D.F. Ferraiolo, R.S. Sandhu, G. Serban: A Proposed Standard for Role-Based Access Control. ACM Transactions on Information System Security, Vol. 4, No. 3, (2001) 224-274.
[2] L.S. Bartz: LDAP Schema for Role Based Access Control, IETF Internet Draft, expired, (1997).
[3] L.S. Bartz: CADS-2 Information Model, not published. IRS: Internal Revenue Service (2001).
[4] Distributed Management Task Force (DMTF), Common Information Model (CIM) Specification, URL: http://www.dmtf.org (2003).
[5] B. Moore, E. Elleson, J. Strasser, A. Weterinen: Policy Core Information Model. IETF RFC 3060, February 2001.
[6] B. Moore, E. Elleson, J. Strasser, A. Weterinen: Policy Core Information Model Extensions. IETF RFC 3460, February 2001.
[7] W. Yeong, T. Howes, S. Killie: LightWeight Directory Access Protocol. IETF RFC 1777, March, 1995.
[8] Distributed Management Task Force (DMTF): Guidelines for CIM-to-LDAP Directory Mappings. whitepaper, May 8th, 2000, URL: http://www.dmtf.org
[9] J. Strassner, E. Ellesson, B. Moore, R. Moats: Policy Core LDAP Schema. IETF Internet Draft, January 2002.
[10] R. Yavatkar, D. Pendarakis, R. Guerin: A Framework for Policy-based Admission Control. IETF RFC 2753, January 2000.
[11] D. Durham, Ed., J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry: The COPS (Common Open Policy Service) Protocol, IETF RFC 2748, January 2000.
[12] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, B. Moore: Policy QoS Information Model. IETF internet-draft, November 2001.

[13] OASIS: eXtensible Access Control Markup Language (XACML) -Version 1.03. OASIS Standard, February 2003, URL: http://www.oasis-open.org
[14] RBPIM Project WebSite. URL:http://www.ppgia.pucpr.br/~jamhour/RBPIM, (2003).

# Management += Grid

Sven Graupner, Vijay Machiraju, Akhil Sahai, and Aad van Moorsel

HP Laboratories
1501 Page Mill Road, Palo Alto CA 94304, USA
{svgr,vijaym,asahai,aad}@hpl.hp.com

State of the art management products have until now focused on monitoring of resources such as networks, systems, servers and applications. Management usually does not deal with handling the complete resource life-cycle, i.e., of providing resources on-demand to application providers, matching and allocating resources to users so that application requirements are best met, providing guarantees through service level agreements, and monitoring and assuring these SLAs. Grid aims to provide seamless access to computational resources. In Grid based systems and the latest initiative of Open Grid Service Infrastructure (OGSI) [3] concepts related to discovery of network topology, allocation based on farms, monitoring towards certain goal (as specified in SLAs), and control to assure these goals are missing. However, there are parallels between grid and traditional management systems. A grid just like traditional management systems is comprised of grid nodes each of which manages a group of resources. The management+ [1] system that we propose, will be able to manage the complete life-cycle of resources assigned to applications, provided, Grid technologies are used for reserving, allocating, and handing over resources to applications and traditional management functionalities are incorporated that deal with monitoring and control.

A management+ system will be comprised of grid nodes that communicate with each other to coordinate the tasks of resource allocation and management. *The management+ system may be used for managing a resource pool inside the enterprise or spanning multiple enterprises*. In order to do so, the management+ system needs capabilities to discover the resources so as to create a resource pool, maintain & model the resource information, be able to provide resources to application providers on receiving requests, be able to provide guarantees as agreed upon through Service-Level Agreements, and by monitoring and assuring them. This poses the following requirements for the management+ system:

**Resource Discovery:** Under the realm of each grid node are collections of resources that are allocatable and manageable by that grid node, that need to be registered and discovered. The registration information may be maintained in a central repository that may be LDAP based (as in traditional Grid) or in a UDDI based registry. The other approach is to let each of the grid nodes manage their own repository, but to have protocols similar to p-2-p systems for searching resources amongst grid nodes.

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 194-196, 2003.

**Resource Modeling:** Resources that are managed by the grid or those even within a grid node are heterogeneous in nature. They are of different types, potentially distributed in different geographic locations and administrative domains (e.g., consolidated in racks vs. located on desktops). Modeling is therefore an important task for the management+ system. A relevant resource model is CIM.

**Requesting Resources and Guarantees: A**pplication providers need to specify their requests to the management+ system in some form. The specification of the request may be done in terms of application level metrics (e.g. throughput, transactions/sec) or at a low-level, an enhanced RSL may be used. The guarantees may be provided in terms of reliability, availability, security, timeliness through SLA[5].

**Resource Allocation and Deployment:** Resource requests are sent to the management+ system that then undertakes resource reservation, allocation and deployment. Resource reservation is done when the resources are not immediately required. The management+ system has to keep a note of all the competing reservations and of ensuring that the resource pool is being properly utilized. As the reservations become current, requests for resources are satisfied through match-making [3].

**Monitoring Guarantees:** Traditionally, applications are installed and operated on a fixed set of associated, physical hardware. In management+ system, resources virtualize the physical hardware. This virtualization provides the capability of transparently switching the physical hardware (in case of degradations/failures) while maintaining the transparency of a continually running application. The metrics specified on the resources in SLAs have to be monitored and aggregated into higher-level metrics that business manager(s) may relate to.

**Assuring Guarantees:** Once the SLA violations are detected, it is important for the management+ system to take corrective actions. Analysis tools are employed to analyze the violation data so as to decide on corrective action(s). The corrective action may range from taking one of the control actions like fail-over, reboot/rejuvenation of the resources to transparently switching the failed resources with new set of resources out of the resource pool. These new resources are obtained through the same resource allocation and deployment mechanisms as discussed earlier. This may enable closed-loop management.

## Conclusion

Management+ system combines traditional management and grid technologies.

## References

[1] Sahai A, Machiraju v, van Moorsel A. A System that combines Grid and Management technologies for Closed Loop Enterprise IT Management. HPL Invention disclosure #200310038 (patent pending).

[2] Platform Computing . http://www.platform.org

[3] Raman R, Livny M, Sloman. M . Match-Making: Distributed Resource Management for High Throughput Computing. *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28-31, 1998, Chicago, IL.

[4] Foster I., Kesselman C, Nick J.M., Tuecke S. The Physiology of the Grid. http://www.globus.org/research/papers/ogsa.pdf

[5] Sahai A, Graupner S, Machiraju V, van Moorsel A. Specifying and Monitoring Commercial Grids through SLA. CCGrid, May 2003.

# A Web Services Signaling Approach over Optical Networks for SAN Applications

Omar Cherkaoui[1], Nathalie Rico[2], T. Dieu Linh Truong[1], Halima Elbiaze[1] and Viet Minh Nhat Vo[1]

[1]Universit du Qu bec Montr al, C.P. 8888 succ. A., Montr al, H3C-3P8, Canada
[2] Universit de Montr al

**Abstract.** Storage Area Network (SAN) promise both a single source of data and improved performance and availability. SAN are extended to a WAN infrastructure to offer easier sharing over multiple locations. This created a need for bandwidth which can be provided by the underlying optical transport network. In this paper, we propose a policy-based lightpath signaling approach based on the OpenGrid architecture allowing SAN applications to dynamically establish and provision lightpaths over an optical network.

## 1 SAN Management Architecture

The storage industry is undergoing a revolution. The emerging technologies such as the SAN promise both a single source of data and improved performance and availability. A SAN is a means to allow multiple servers to have direct access to common storage devices or storage pool. Today, SAN are deployed over a LAN infrastructure. The increasing need to share information across organizations led companies to extend the SAN over WAN. IETF and other organizations proposed different solutions for realizing a SAN over the IP network. SAN applications need to have access to the required bandwidth provided by the underlying optical transport network to obtain adequate performance. We propose a lightpath signaling approach based on the OpenGrid architecture allowing SAN applications to establish and control lightpaths over an optical network. To address the admission control and scalability issues, we use a policy-based approach providing a flexible means to control the set-up of optical lightpaths.

VSAN (Virtual SAN) is like a virtual private network dedicated to exchange storage traffic. The network architecture for VSAN is composed of SANs interconnecting through the Internet WANs. To facilitate the management and control, a policy server in each SAN store the service policies. It is used to activate a service, to reserve the storage units and to configure the FC switches to build routes from the hosts to the storage units. A service provider who has available VSAN services advertises them on a web site. The web site may be located locally or on a common web server such as an UDDI (Universal Description, Discovery and Integration) database server.

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 197-199, 2003.

The proposed signaling approach is based on the Open Grid Services Architecture (OGSA) [2]. The architecture incorporates a policy framework to control the establishment of lightpaths on high-speed networks (see Figure 1). The OGSA integrates Grid technologies with Web service mechanisms to create a distributed system framework based around the Grid Service. Grid services conform to WSDL interfaces for such purpose as discovery of characteristics, notification, etc. OGSA introduces registration interfaces for creating and discovering services. The availability of lightpaths and cross connects is advertised through standards based registry services such as UDDI and WSIL (WSIL (Web Services Inspection Language). LPO (Light-Path Object) registries contain all the services for creating the end-to-end lightpath. The LPO Factories are independent abstract components, which discover and register all provisioned end-to-end lightpaths for a given domain with dedicated Grid service registry. Network reachability information is exchanged between systems to construct the paths, allowing the creation and control of lightpaths. The solution allows communication with source and destination routes to create a new forwarding path along established lightpath and provides the capability to control the state of the lightpath. Interfaces have been developed to offer services for SAN application, which will query the lightpath registry services and establish an end-to-end lightpath. A policy manager is used to control user access. SAN needs can be translated in a policy condition. End-to-end lightpath are created according to a specified policy rule or defined network management policies. The policy server provides service for specifying and saving policies. The policy registry allows advertising the policy services. The signaling approach uses WSDL-based policy services for admission control and other policy services. The Grid resource management model is based on CIM (Common Information Model) [3] and its policy extension PCIM, allowing an efficient exchange of policy information. The preliminary implementation experiments demonstrate the Web-based signaling approach for creating and controlling the lightpath according to different policy conditions. The policy approach helps achieve an optimal utilization of network resources required by SAN applications.

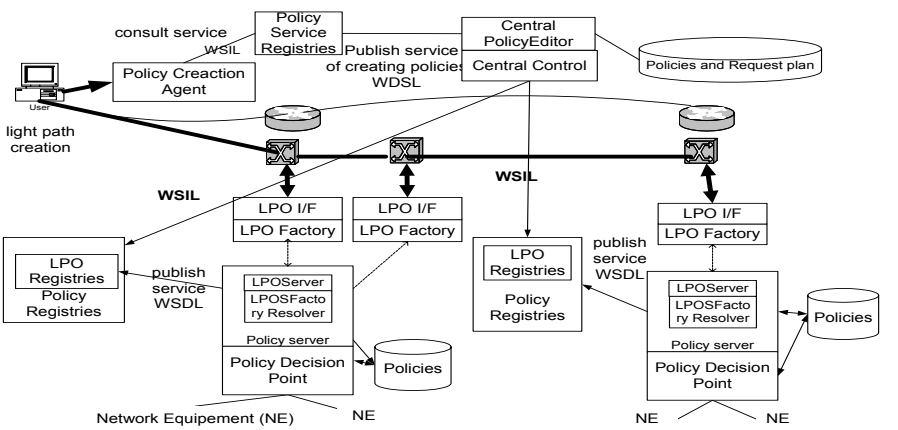



**Fig. 1.** Architecture

We proposed a signaling approach based on the Open Grid architecture to dynamically provision lightpaths for end-to-end high volume SAN data transfers. SAN applications that need large high-speed bandwidths can use the policy services for controlling the access, set-up and tearing down the lightpaths based on policy conditions.

## References

[1] Ravi Kumar Khattar et al, Introduction to Storage Area Network (SAN), Redbooks.

[2] I. Foster and al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", OGSI WG, Global Grid Forum, June 22, 2002.

[3] CIM Schema, version 2.x. DMTF, http://www.dmtf.org/standards/standard_cim.php

# A Self-Configuring Sensing System for Data Centers

Malena Mesarina, Cyril Brignone, Tim Connors, Mehrban Jam, Geoff Lyon, Salil Pradhan, and Bill Serra

Hewlett-Packard Laboratories
Palo-Alto, CA 94304, USA

**Abstract.** Wiring sensors in a data center is extremely expensive in comparison to the wiring of computing equipment. This is due to the central architecture of traditional sensing systems, which requires long wires to be connected between racks and a central box. In addition, a re-layout of the racks after the sensor wires are deployed is practically impossible. We propose using a wireless self-configuring network of smart sensing nodes to alleviate these problems. We explore how to design the sensor control software to be self-reconfiguring when nodes relocate. The software is divided in three layers: network organization, data aggregation and visualization. In this paper, we identify several insights into the thermal monitoring requirements, design issues and initial design solutions for these layers.

## 1 Introduction

We propose a self-configuring, wireless sensing network to eliminate the labor cost of wiring a sensor[1] and facilitate the re-layout of racks. The nodes in the network are able to locate themselves, using a time-of-flight method, with a precision of less than 5 cm [1]. The thermal monitoring software is structured in layers of network, data aggregation and visualization functions. Self-configuration means automatic adaptation of these layers to node moves or failures. The network layer adapts by updating data routes; the aggregation layer by reselecting clusters of sensors and the visualization layer by relocating objects in its model. The adaptive algorithms are distributed for the network and centralized for the aggregation and visualization layers. In this report on a first experimental implementation, we identify requirements, design problems, tradeoffs, and provide insights into initial algorithmic solutions.

## 2 Software Layers

The layers mentioned above should meet the application's requirements: to avoid IP address assignment to sensors, the wireless and IP networks should be separate and bridged by access points. Sensor reading is periodic or event triggered.

[1] $1000 according to customers

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 200–201, 2003.

The data flows from sensors to access points, and between immediate neighbor sensors. Data bit rate is low (5-10 kbps) and sensor density is high (5-7/$m^3$). The following assumptions are made: a node knows its location and its neighbors' locations and ids; nodes report their locations to access points, and the radio has a fixed transmission power.

### 2.1 Network Configuration, Data Aggregation and Visualization

The network layer solves three problems: determining the number of access points, finding routes from sensors to access points, and updating routes when nodes move or fail. The number of access points depends on the maximum number of sensors that talk to an access point, bandwidth, and cost. We use a source routing approach for route discovery. The access points broadcast discovery packets to nodes within a delineated area. A receiving node appends its IDs and location and forwards the packet. Nodes construct multipaths using information from the discovery packets. Updating routes consists of deleting paths from memory when a neigbor sensor moves or fails, and adding a new path when a new neighbor node appears. The underlying localization algorithms in the node detect when a node has moved or failed and notifies the network layer. A node computes a new path with the new node as an intermediate node based on the new node's location and existing knowledge of paths.

The network aggregates packets to reduce the number of packet transmissions, saving power and bandwidth. We use a 1-hop clustering scheme, where a cluster-head acts as the data aggregator. The problems to solve are: cluster formation, cluster-head selection, de-synchronization of data collection between channel sharing clusters, and selection of non-colliding paths from cluster-heads to access points. Cluster formation should be based on proximity, line of sight between sensors, and historical data correlation. The cluster formation and cluster algorithms are centralized and controlled by the access points, since they require global knowledge of nodes locations and failures.

Given the location of sensors, our visualization tool, Geoview, creates a geometric model of the room in 3D, allowing the user to program the sensors by clicking on the display.

### 2.2 Conclusions

We propose a self-configuring sensing infrastructure and control software that adapts to sensor mobility. We identify several system requirements, design problems, tradeoffs and initial solution approaches. We plan to deploy our system in a data center at HP Labs in Palo Alto in the fourth quarter of 2003. Low implementation costs and reonfiguration latency will validate the efficiency of our design in comparison to traditional methods.

## References

[1] Brignone, C., Lyon, G.: Smartlocus: A location aware system for adaptive environments, hpl-2003-41. Technical report, Hewlett-Packard (2003) 200

# Towards Autonomic Business Activity Management

Jun-Jang (JJ) Jeng and Henry Chang

IBM T.J. Watson Research Center
Yorktown Heights, New York

Business activities can be highly complicated and dynamic. This complexity lies in the number of activities and organization, and many business relationships between the things that make up a business activity and its business environment. The dynamism of business activities comes from the needs of continuously measuring, monitoring, controlling and improving business activities. The systems providing this type of functions and services are coined as Business Activity Management (BAM) systems. A BAM system enables an enterprise to respond to emerging problems at right time. Right time does not equal to real time though, in many cases, real-timeliness is one of the requirements. Traditional techniques for building BAM systems are proven to be inadequate in dealing with the aforementioned complexity. It is not uncommon for some minor perturbation in some remote corner of the system to have unforeseen, and at times catastrophic, global repercussions. In addition to being fragile, many situations arising from the extreme dynamism of these systems require manual intervention to keep distributed applications functioning correctly. In our opinion, BAM systems should possess the characteristics of confronting the high degree of complexity and enabling the construction of robust, configurable, scalable, self-organizing and self-managing distributed systems.

A typical electronics value chain consists of retailers, manufacturers and parts/capacity suppliers. These value chain partners collaborate in the context of business operations to deal effectively with dynamic changes in the marketplace. These collaborative relationships enable synchronization of the orders, production schedule, and parts supply plan. Traditional workflow management can help automate the business activities that make up this scenario. The state-of-the-art involves modeling the business processes and executing these models in an appropriate workflow engine. The behavior of the resulting system is static in the sense that it cannot accommodate any business situations not explicitly modeled at build-time. Therefore, a workflow approach tend to create an infrastructure that will not be capable of handling business related deviations or exceptions or evolutionary changes.

We propose using an adaptive entity named *BABot* (Business Activity Bot) as the cornerstone of the BAM systems. Since the relationships among business activities and entities are dynamic, the bindings between BABots at run time are dynamic as well. A BABot consists of seven perspectives. The *context* perspective describes the contextual information for the existence and behavior of a BABot. Being aware of business contexts is essential for driving effective development of BABots. The *intent* perspective describes the goal of a BABot and relates the goal to other goals, capa-

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 202-203, 2003.

bilities, and values. The *value* perspective is the value-exchange view of the concerned business activities. This uses the monitoring process heavily to sample and analyze the operations of target business activities. The *capability* perspective specifies and links what a BABot can do, from the strategy level to operation, to execution level, and to the resource level. The *constraint* perspective regulates the logical relationships among values, capabilities, intents, and contexts. Constraints impose a set of configurable commitments and rules upon designated BABots. The *resource* perspective specifies the resources governed by the enclosing BABot. Resources can be shared among multiple BABots. The *processes* perspective describes how BABots cooperate with one another and harness passive resources in order to execute capabilities and to desired outcomes. Events trigger the commencement of processes. Five processes are identified in BABot-based BAM systems, i.e., metric calculation, situation detection, situation analysis, decision making, and decision enforcement.

Consider the supply chain example. A conventional management system for supply chain often consists of two layers: a planning layer and an execution layer, where the planning layer allows business condition to be periodically analyzed in order to make adjustment of work scheduling, supplier contract, or crucial resource allocations. The execution layer is the daily operational process to carry out business functions by following the preset rules and policies set up by the planning layer. Our approach inserts a reactive layer between the planning and execution. Since the planning layer has to be done periodically to avoid thrashing and destabilizing the execution system, there will be process exceptions, special situations not responded well in the execution layer. To provide real-time adjustment, the reactive management layer would predict, localize and classify the situation. If the situation warrants a big adjustment of planning, an alert would be sent to the planning layer to ask for urgent rep-planning, such as the disruption of goods supply during the event of war or market crash. The following diagram shows the sense-and-response support of a supply chain through the BAM system. If the situation is repairable, the BAM system may resort to automatic analytic algorithms to assess the damage and trigger an appropriate autonomic recovery process.

Autonomic BAM system represents a natural evolution of the policy-based real-time monitoring and reactive recovery functions of a complex system. A percentage of the detected situations may be amendable for self-healing, self-management, and self-optimization with minimal human interventions. Others may definitely need human involvement. Such being the case, the mathematical modeling techniques required for a autonomic business process is not far away from that needed for a IT component such as storage subsystem. In the BAM domain, we believe that the autonomic functions will emerge by adopting better prediction and optimization algorithms for specific business domains, such as inventory replenishment in supply chain, risk containment in finance. We are developing BABot-based BAM system as the endeavor towards an enabling infrastructure for self-managing BAM systems. We will investigate cloning, versioning, and mobility of BABots with an eye towards non-functional requirements such as performance, availability, fail-over, and migration that are critical to the success of BABot in practice. This infrastructure is being applied to real-world scenarios including supply chain, logistics, finance, insurance, and manufacturing.

# Idiosyncratic Signatures for Authenticated Execution of Management Code

Mario Baldi[1], Yoram Ofek[2], and Moti Yung[3]

[1] Torino Polytechnic, Computer Engineering Department
Torino, Italy
mario.baldi@polito.it
www.polito.it/~baldi
[2] Synchrodyne Networks, Inc.
New York, NY
ofek@synchrodyne.com
[3] Columbia University, Computer Science Department
New York, NY
moti@cs.columbia.edu

**Abstract.** TrustedFlow™ is a software solution to the problem of remotely authenticating code during execution. A continuous flow of idiosyncratic signatures assures that the software from which they have emanated is not changed prior to and during execution. TrustedFlow™ can be used to create a run-time trust relationship between the components of a distributed management system.

Software, especially in the context of data networks, suffers from some inherent problems. These include modifications by an either malicious or inadvertent attacker, malware distribution (e.g., viruses and "Trojan horses"), and the use of malicious software remotely for penetration, intrusion, denial-of-service (DoS), and distributed DoS (DDoS). Management software is particularly critical from this point of view since it is used to *monitor* and, especially, to *control* network devices. Hence, malicious modification and use of management code can be particularly harmful to the network and advantageous for the attacker. Moreover, distributed, possibly self-, management software, which is spread on various systems and possibly dynamically downloaded, is particularly exposed to manipulations.

The presented software solution aims at overcoming the above mentioned problems and at assuring (in many typical scenarios) that management operations are executed by a trusted software source. The solution is based on continuous authentication, ensuring at run-time that the correct software has been employed. This unique method for continuous authentication is based on a continuous flow of idiosyncratic signatures that are constantly being generated and emanated during execution. With reference to the architecture shown in Figure 1, the idiosyncratic signatures are generated by a secret function called Trusted Flow Generator (TTG) that is hidden (e.g., obfuscated) in the software and whose execution is subordinated

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 204-206, 2003.

to the proper execution of the software being authenticated. The flow of signatures is validated at a remote trusted component called Trusted Tag Checker (TTC). Consequently, this method guarantees that the correct software modules are used at run time, i.e., the authenticity of the executed software can be trusted. An explanation of the TrustedFlow™ architecture and the basic principles of the the TrustedFlow™ protocol can be found in [1].

The TrustedFlow™ protocol is an *add-on software protection component* intended to be included within other protocols, such as those, for example, of distributed computation (e.g., grid computing), traffic generation (e.g., TCP), and management (e,g., SNMP). In essense, the TrustedFlow™ protocol provides run-time continuous (multi-factor) authentication, certifying the authenticity of software modules that were used to compute, generate, and send messages. As such, it becomes evident that the TrustedFlow™ protocol has broad applications in both networking and computing for military and commercial environments including, besides trusted network management.

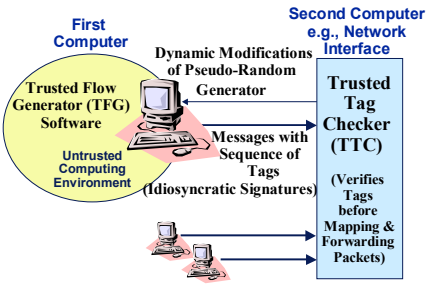



**Fig. 1.** TrustedFlow™ architecture

The TrustedFlow™ protocol is complementary to many of the current enhancements for secure computing and networking protocols, including the security extensions to the Internet management model defined in SNMPv3. In fact, (to the best of our knowledge) no other authentication method certifies the software continuously during run-time by emanating idiosyncratic signatures. In other words, while other approaches provide privacy and authentication protecting from the attacks of a man in the middle, TrustedFlow™ protects from the attacks of *a man at the edge*. The TrustedFlow™ protocol has broad *synergistic implications* on various computing and networking protection means, as discussed in more detail the related work section of [1]. A prototype of the TrustedFlow™ architecture sponsored by Microsoft Research is under development at Torino Polytechnic.

In general, network management involves actions that are critical for proper network operation. SNMPv3 defines security mechanisms that enable authentication of management messages. However, these mechanisms do not protect from modified, possibly malicious, code that has access to proper authentication information. For example, by running on a network management station, the malicious code could get a hold of the certificates used for authentication with the management agent. The TrustedFlow™ protocol could be used to complement SNMP security mechanisms by including a TFG and a TTC in the management entity and in the management agent.

- A TFG in the management entity and a TTC in the management agent enables the latter to trust the management code requesting to store information in the MIB or to pass along information contained in the MIB.
- A TFG in the management agent and a TTC in the management station enables the latter to trust the agent code generating a trap or sending MIB information previously requested.

Embedding both a TFG and TTC in both the management entity and the management agent enables the creation of *mutual trust* in the execution of the other component. Hence, the TrustedFlow™ protocol can be used, possibly together with SNMP security mechanisms, to create a trusted run-time network management distributed environment. The resulting benefits increase with the degree of distribution and code mobility.

## References

[1] M. Baldi, Y. Ofek, M. Yung, "Idiosyncratic Signatures for Authenticated Execution - The TrustedFlow™™ Protocol and its Application to TCP," IASTED CSN 2003, Benalmadena, Spain, 2003.

# Effects of Wavelength Conversion on Self-healing Optical Networks

Hoyoung Hwang

Digital Media Engineering Dept., Anyang University
Anyang 430-714, South Korea
hyhwang@aycc.anyang.ac.kr

**Abstract.** This paper studies the effects of wavelength conversion on backup routing and spare capacity utilization in optical networks. The efficiency of spare wavelength utilization is proportionally increased as the wavelength conversion capability increases, different from the call blocking probability for which about 30% of wavelength conversion capability shows nearly the same performance as full wavelength conversion capability. The spare resource utilization efficiency can be improved by using alternate routing and wavelength assignment algorithms.

## 1 Motivation

Wavelength division multiplexing (WDM) optical networks can be implemented in all-optical manner without wavelength conversion, or in opaque manner with full wavelength conversion. A hybrid implementation is also possible with limited number of O-E-O conversion switches that provide partial wavelength conversion capability. In all-optical networks with only transparent switches, a single wavelength should be assigned to a lightpath throughout the optical route thus shows inflexibility in routing. The wavelength conversion capability can relieve this inflexibility by assigning different wavelengths to different links of an optical route. It is known that about 25-30% of partial or sparse wavelength conversion capability can provide nearly the same call blocking probability as that of full wavelength conversion networks [1]. However, the effects of wavelength conversion capability on the network protection/restoration aspects and on the spare resource utilization have not been studied enough.

This paper studies the effects of wavelength conversion on the spare resource utilization, and proposes an alternate backup connection provisioning method using semi-lightpath [2]. A set of one or more consecutive lightpaths to connect a source-destination pair is called *semi-lightpath*, and each lightpath segment to build a semi-lightpath is called *sub-lightpath*. An optical path can be divided into several sub-lightpath segments, and those sub-lightpath segments may have different wavelengths from others and can be shared among backup semi-lightpaths, thus wavelength conversion effect can be achieved.

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 207–208, 2003.

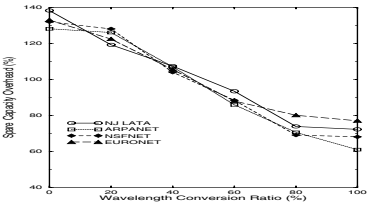



**Fig. 1.** Conversion ratio vs. Capacity overhead for single link failure

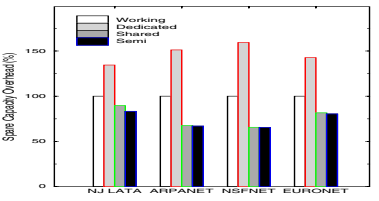



**Fig. 2.** Semi-lightpath vs. Other configurations

## 2 Wavelength Conversion and Semi-lightpath

Simulations were performed to see the effects of wavelength conversion on self-healing optical networks. Four mesh topology networks are used assuming a network link has bidirectional fibers and each fiber had 40 wavelengths. Fig. 1 shows the spare capacity overhead versus wavelength conversion capability using disjoint shortest backup path provisioning. The wavelength overhead decreases almost linearly as the wavelength conversion ratio increases, which is different from the curve of call blocking probability mentioned earlier. This result comes from the probability of spare wavelength sharing according to the wavelength conversion capability.

In Fig. 2, the semi-lightpath based backup configuration was compared with dedicated capacity restoration in all optical networks and shared capacity restoration in opaque networks. The semi-lightpath configuration shows substantially better capacity efficiency compared with the dedicated path restoration, 51.1% to 94.1%, and a little bit better capacity efficiency compared with the shared path restoration, 0.4% to 6.2%. In this simulation, about 30% of network nodes were selected as wavelength conversion nodes based on 'higher degree node first' policy, and then updated manually considering even distribution in the network topology. Sub-lightpaths were configured between those selected nodes and the source/destination nodes of working lightpaths, and backup semi-lightpaths were configured to maximize the sharability of spare wavelengths on sub-lightpaths.

In summary, the semi-lightpath based backup path provisioning provides an improved way of spare wavelength utilization and routing, especially for hybrid optical networks with limited wavelength conversion capability. Analytical approach and detailed design method for semi-lightpath based backup path provisioning are to be studied.

## References

[1] G. Shen et al. Operation of WDM Networks with Different Wavelength Conversion Capabilities. IEEE Communications Letters, 4(7):239–241, Jul. 2000. 207

[2] I. Chlamgtac et al. Lightpath Routing in Large WDM Networks. IEEE Journal on Selected Areas in Communications, 14(5):909–913, Jun. 1996. 207

# Efficient and Transparent Instrumentation of Application Components Using an Aspect-Oriented Approach

Markus Debusmann[1] and Kurt Geihs[2]

[1] Fachhochschule Wiesbaden, University of Applied Sciences
Department of Computer Science, Distributed Systems Lab
Kurt-Schumacher-Ring 18, 65197 Wiesbaden, Germany
debusmann@informatik.fh-wiesbaden.de
[2] Berlin University of Technology
Intelligent Networks and Management of Distributed Systems
Einsteinufer 17, 10587 Berlin, Germany
geihs@ivs.tu-berlin.de

**Abstract.** The increasing significance of Service Level Management (SLM) strongly requires an appropriate instrumentation of application components in order to monitor compliance with the defined Service Level Objectives (SLOs). The manual instrumentation of application components is very costly and error-prone and thus rather inefficient.
This paper presents an approach for using aspect-oriented programming techniques for efficiently and transparently instrumenting application components. The approach is applied to the interference sensitive area of performance monitoring using the Application Response Measurement (ARM) API. Experiments with a prototype have revealed that our aspect-oriented approach fits well to the integration of instrumentation code into application components and that the runtime overhead is only slightly higher than the overhead of a manual instrumentation.

## 1 Motivation and Related Work

Over the past years, economic pressure has forced enterprises to outsource many IT services and purchase them from external service providers. Quality-of-Service (QoS) parameters agreed on by service providers and their customers are laid down in a contract, called service level agreement (SLA) [1, 2, 3]. Typical QoS parameters specified in SLAs define availability criteria and performance-related metrics, e.g. response times. The fulfilment of such an SLA has to be monitored at run-time by both the customer and the service provider. Customers are primarily interested in short end-user response times and high service availability. Providers are interested in a more fine-grained view of the interrelated performance metrics of the components constituting their services, especially when using the same service infrastructure for different customers at the same time.

For computing high-level SLA parameters, such as service response times and availability, the application components within the service provider domain

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 209–220, 2003.

have to be instrumented appropriately for management purposes. Instrumenting applications incurs additional cost since instrumentation is a non functional requirement that is not part of the business logic interface. Often, instrumentation is added on top of existing applications in an ad-hoc manner as described in [4]. In [5] an integrated software development process for applications and their management infrastructure is described. The development of the management functionality runs in parallel to the development of the normal software. For implementing management solutions based on CIM [6], developers are additionally supported by a design patterns catalogue that contains reusable patterns for CIM models.

Alternative approaches aim for realising instrumentation in a transparent manner, e.g., by instrumenting middleware components and platforms. [7] presents an approach for modifying an EJB server in a way that EJBs are instrumented automatically while they are deployed. Response times of method invocations are automatically monitored using the Application Response Measurement (ARM) API defined by the OpenGroup [8]. In addition, EJBs are instrumented via Java Management Extensions (JMX) [9] to expose configuration information. In [10] CORBA applications are instrumented based on CORBA Portable Interceptors which allow to insert any kind of instrumentation code that is executed at certain points of method invocations. The insertion of the Portable Interceptor code into the application causes only minimal effort for the application developer. He only has to add two additional lines of code to the source code of the component which is to be instrumented. [11] presents an approach for an automated instrumentation of component-based applications. The approach is limited to measuring and correlating response times similar to the Application Response Measurement (ARM) API defined by the OpenGroup.

The examples described above show that there is a certain tradeoff between the degree to which the instrumentation can be automated and the level of detail obtained by the instrumentation. The more generic the instrumentation approach the more abstract is the achievable data. On the other hand, the more specific information shall be extracted the higher the instrumentation costs for the application developer. Obviously, if fine-grained monitoring is the goal, then the instrumentation needs to be woven into the code of the application components. This is where aspect-oriented programming (AOP) can help.

This paper presents an aspect-oriented instrumentation approach for application components. Aspect-orientation helps to solve the shortcomings of the existing instrumentation approaches by providing transparency for the application developer and furthermore offering the opportunity to monitor management data at any level of detail. In addition, it is not limited to monitoring a single environment like CORBA or EJB.

The paper is structured as follows: Section 2 gives a brief introduction to the concepts of aspect-oriented programming and introduces AspectJ as an example of an aspect-oriented development environment. In Section 3 we present how the aspect-oriented paradigm can be applied to simplify the monitoring of distributed applications. Section 4 discusses performance measurements that under-

line the efficiency of the aspect-oriented instrumentation compared to a manual instrumentation. Section 5 concludes the paper and gives an outlook to future work.

## 2 Aspect-Oriented Programming (AOP) and AspectJ

The design of large applications is arduous since decisions made in early modelling stages influence later phases, i.e. implementation and maintenance. Hence much effort in the past has been made to employ proper software analysis and design methods. Typically these methods are tied to a distinct programming paradigm, e.g. imperative or object-oriented, and they attempt to structure the model accordingly. The aspect-oriented programming paradigm (AOP) [12] claims that although conventional analysis and design methods try to partition a given problem into self-contained, encapsulated entities, this kind of partitioning is not always feasible. Dependencies between modelled entities break the desired encapsulation and thus make the design hard to deploy and even harder to reuse. AOP aims at a conceptual understanding of the "cross-cutting" of responsibilities through separate entities. For that purpose, the AOP distinguishes aspects and components.

A component is an entity which encapsulates a distinct responsibility. Components can be combined with other components to achieve a distinct behaviour. They interact through well defined interfaces.

In contrast to components, aspects are not encapsulated. They cross-cut each other, preventing clear encapsulation. Aspects typically reflect non-functional issues of a system, e.g., error handling, performance tuning, or synchronisation.

The proposed solution to the coexistence of aspects and components in a system is the introduction of aspect languages and a corresponding aspect weaver. Aspect languages are specialised languages, suitable for modelling and expressing the distinct properties of an aspect and its connection with other aspects. The aspect weaver is a tool which takes all aspect specifications of a system and generates a corresponding program.

So far, most aspect-oriented approaches dealt with distribution and synchronisation aspects in distributed systems. Recently, the integration of QoS management in distributed applications has become a target for AOP-based systems [13, 14].

AspectJ [15], originally developed at Xerox Parc, is an implementation of aspect-oriented programming paradigm for the Java language. In the following, a short overview of the AspectJ's basic concepts and constructs is given; details are provided in [16]. AspectJ defines the concept of a *join point* which represents a well-defined point within the program flow, e.g., a method call, a constructor call, referencing of a field, etc. *Pointcut designators*, or simply *pointcuts*, are used to identify certain join points within the program flow. Pointcuts can also be combined using filters to define more complex expressions. For example the pointcut

```
pointcut foo() : call (void ClassA.methodA (int)) ||
                 call (* ClassB.get* (..));
```

identifies every invocation of `methodA` in `ClassA` as well as every invocation of `ClassB`'s methods that start with `get`, regardless of its return type and its parameter list.

*Advices* define the additional code, typically implementing the cross-cutting concern, that should be executed when a join point is reached. Pointcuts are used within the definition of an advice to identify the join point. Three different advice types are distinguished: *before advice* run when their joint point is reached, *after advice* run after their joint point, and *around advice* run in place of their join point. For example the advice

```
before() : foo
{
  System.out.println ("After pointcut foo");
}
```

prints out a simple message when the pointcut `foo` is reached and before the computation of the original code proceeds.

Pointcuts are able to expose the execution context at their join point. This context information can be used in advices.

```
pointcut setX(ClassA a, int x) :
{
  call (void ClassA.setX(int))
    && target (a)
    && args (x);
  before(ClassA a, int x)
  {
    System.out.println ("New value for x: " + x);
  }
}
```

This pointcut `setX` exposes the two values from calls to method `setX` of `ClassA`: the instance of `ClassA` that receives the call and the new value for `x`. The advice prints out the new value of `x` before the `setX` method is invoked.

*Introductions* are used to modify classes and the hierarchy by adding new members and changing relationships between classes. Introductions change the declarations of classes. Since these changes are inherited they effect the rest of the program. Introductions are static, i.e., they take place at compile time whereas advices operate during runtime.

The definition of *aspects* is very similar to classes. Aspects define the units for implementing cross-cutting concerns using pointcuts, advices, and introductions.

```
aspect Count
{
  private int count = 0;
  pointcut CountSetX () : call (ClassA.setX (int));
  before() : CountSetX ()
  {
    count++;
  }
}
```

The aspect `Count` introduces `count` as a new member of `ClassA` and defines a pointcut for the `setX` method of `ClassA`. The before advice increments the newly introduced `count` variable every time the `setX` method is invoked.

## 3 Monitoring Distributed Applications with AOP

Managing distributed systems requires knowledge about the status of the constituting components. Their status can be deduced from information gained by monitoring the components. This is realised by inserting instrumentation code into the managed system. Here, two basic approaches can be distinguished: First, an intrinsic instrumentation approach where code is inserted into the component under monitoring. Second, an extrinsic instrumentation approach which uses additional components, either hardware or software to monitor components of the system, e.g., by analysing the log files of a Web server. Typically, the intrinsic approach is able to provide more fine-grained data, since component internals can be used. The extrinsic approach typically provides more coarse-grained information since it has to rely on information exported by the component. However, the latter approach is less intrusive.

Our approach based on aspect-oriented programming can be characterised as an intrinsic instrumentation approach since it inserts the instrumentation into the code of the component. In the following sections we describe how our approach helps to make components more manageable.

### 3.1 Measuring Servlet Response Times Using ARM

In the area of Service Level Management, Service Level Objectives (SLOs) defining availability criteria and performance-related metrics, e.g., response times, are the key issues of many Service Level Agreements. The fulfilment of an SLA has to be monitored to prove compliance with the defined SLOs. In the terms of AOP, response time is a non-functional requirement and its measurement a cross-cutting concern since many parts of a component are involved. Even further, several components may be involved and their measurements have to be correlated.

For response time measurements the ARM API is a well accepted approach for instrumenting applications at the source code level. The API supports execution time measurements of source code fragments termed ARM transactions within a distributed application. ARM provides ways of correlating nested measurements, even across host boundaries. For this purpose the API provides correlators that identify ARM transactions. Correlators can be supplied on creating a nested transaction for relating this to the enclosing transaction. However, passing of correlators between application components, which might prove difficult especially in distributed systems, is the task of the application developer.

### 3.2 Scenario

In the future, application service providers will offer a wide range of different services from simple web hosting to complex e-business applications. These complex applications are typically realised in a Web-based e-business environment consisting of a Web server as central entry point and a web container to provide Java server pages and servlets. Most of the business logic is implemented by Enterprise Java Beans (EJB) that live in an EJB container. CORBA is used to implement business logic as well as an integration middleware for legacy components. A relational database ensures the persistent storage of the enterprise data.

Our previous research has revealed that instrumenting such a complex environment using ARM is a difficult task [17]. Since the ARM specification does only specify the format of the ARM correlator which is used for correlating nested transactions, but not the mechanism to transfer them between process boundaries, this task is up to the software developer.

Infrastructure components like a Web server and a Web container can be instrumented transparently for the application developer. However, application components, such as Servlets, should also be instrumented transparently for the application developer in order to ensure cost effectiveness. In addition, an automatic instrumentation guarantees a consistent instrumentation of all application components which is the prerequisite of comparable measurements. The following section describes our aspect-oriented instrumentation for transparently measuring Servlet response times using the ARM API.

### 3.3 Aspect-Oriented Instrumentation Approach

A typical task of a Servlet is the retrieval of records from a database system. Thus, the duration of a database query is an important unit of work to be measured. Listing 3.1 shows the code of a simple Servlet that queries a number of records from a database and subsequently transforms the result set into a Web page. For the clarity of the code all `try` and `catch` clauses are not shown in the listing.

The `init` method of the Servlet, which is executed only once during the initialisation of the Servlet, is responsible for setting up the connection to the database. The `destroy` method closes the database connection when the Servlet is destroyed. The database query is performed as part of the `doGet` method. Finally, the results of the query are transformed into HTML and sent back to the client.

An application developer who has to instrument the Servlet with ARM manually would place code for initialising the ARM environment into the `init` method, and code for shutting down the ARM environment into the `destroy` method. Within the `doGet` method the application developer has to check if a parent correlator was handed over to the Servlet. Afterwards, a new ARM transaction must be created and appropriate start and end points for the measurements have to be identified in the application code. In our example, start and stop commands were placed around the execution of the database query.

**Listing 3.1** Servlet for querying a database

```
public class MyDB extends HttpServlet {
  final String url = "jdbc:mysql://dbhost:3306/mysql";
  final String driver = "com.mysql.jdbc.Driver";
  final String query = "select host, user from user";
  Connection conn;

  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    Class.forName(driver);
    conn = DriverManager.getConnection(url, "dbuser", "");
    conn.setReadOnly(true);
  }

  public void destroy() {
    if(conn != null && !conn.isClosed())
    conn.close();
  }

  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ... {
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    java.lang.Thread.sleep (20);
    response.setContentType("text/html; charset=ISO-8859-1");
    PrintWriter out = response.getWriter();
    out.println("<HTML><BODY>");
    ...
    out.println("</BODY></HTML>");
    rs.close();
    stmt.close();
  }
}
```

In this simple example the manual instrumentation is considerably simple. Nevertheless, the application developer has to handle all the complexity of the ARM environment in order to achieve useful measurements, i.e., the application developer has to understand the business requirements of the application to correctly implement the functionality as well as the management requirements to support the management of the application component later on. In addition, the manual instrumentation leads to enormous costs since many code pieces have to be instrumented. This work is monotonous and distracts from implementing the business logic. Using inheritance does not really solve the problem since it mainly simplifies the initialisation and destruction of the Servlet. Furthermore, it requires a refactoring of the existing code which is again a potential source of errors.

The aspect-oriented approach does not require modifications of the existing code. Here, the instrumentation code is encapsulated as an aspect which may be done by a different developer who is familiar with ARM environments, while the application developer can concentrate on the application logic. The application code is simply recompiled using a special compiler, the aspect weaver, which connects the aspect code with the application code. Thus, instrumentation can also easily be integrated into an existing application.

Listing 3.2 depicts the aspect code for measuring the duration of database queries. (Again, `try` and `catch` clauses are not shown in the listing.) First, a number of local variables are defined that are used within the aspect for the ARM measurements. Thereafter, the first pointcut identifies the `init` method of the Servlet, the second pointcut identifies the `doGet` Servlet method, and the third pointcut identifies all invocations of the `executeQuery` method of the

**Listing 3.2** Aspect for instrumenting JDBC calls with ARM

```
aspect MyDBAspect {
  ArmTransactionFactory tranFactory;
  ArmTransaction dbTransaction;
  HttpServletRequest request;
  byte[] myByteUuid = null;

  pointcut arm_init() : call(void *.init(..));

  pointcut arm_doGet(HttpServletRequest request, HttpServletResponse response)
    : call(void *.doGet(HttpServletRequest, HttpServletResponse))
      && args(request, response);

  pointcut execQuery(String content)
    :  call(ResultSet Statement.executeQuery(String)) && args(content);

  void around() : arm_init() {
    Class tranFactoryClass;
    tranFactoryClass = Class.forName(tranFactoryName);
    tranFactory = (ArmTransactionFactory)tranFactoryClass.newInstance();
    myByteUuid = new byte[] { (byte)0x6c, ..., (byte)0xf1 };
    proceed();
  }

  void around(HttpServletRequest request, HttpServletResponse response)
    : arm_doGet(request, response)  {
    ArmUUID uuidDbTransaction;
    this.request = request;
    uuidDbTransaction = tranFactory.newArmUUID(myByteUuid);
    dbTransaction = tranFactory.newArmTransaction(uuidDbTransaction);
    proceed(request, response);
  }

  before(String content): execQuery(content) {
      dbTransaction.start((ArmCorrelator)request.getAttribute("CORRELATOR"));
      ArmCorrelator corr = dbTransaction.getCorr();
  }

  after(String content) returning():execQuery(content) {
      dbTransaction.stop(ArmConstants.ARM_GOOD);
  }
}
```

`Statement` class. The second and the third pointcut also expose variables from their context.

The instrumentation code of the aspect is defined in its advices. The `arm_init` advice is an around advice that traps the execution of its join point (`init` method of Servlet), i.e., the code of the advice will be executed in place of the original code. By including the `proceed` statement at the end of the advice the original code of the `init` method will also be executed. The advice initialises the ARM environment.

The `arm_doGet` advice traps the execution of `doGet` Servlet method and handles the initialisation of a new ARM transaction for measurement. By using the `proceed` statement the original code is executed.

The ARM measurements are performed by using a `before` and an `after` advice for the `execQuery` pointcut. These advices place ARM start and stop statements around the execution of the database query. When starting a measurement the instrumentation code tries to extract a parent correlator from the `CORRELATOR` attribute of the the request object. This parent correlator will then be used as basis for the measurements within the Servlet and later enables a management application to correlate measurements of different components.

## 4 Performance Evaluation

To evaluate the efficiency of the aspect-oriented instrumentation approach we performed a series of measurements under lab conditions. The goal was to determine the runtime overhead caused by the aspect-oriented instrumentation approach.

Our prototype involved the Tomcat Web container in version 3.21 which was instrumented using the tang-IT ARM library [18]. The Servlet code was developed using the Sun Java Development Kit (JDK) 1.4.0; we used AspectJ Version 1.0.6 as aspect-oriented programming environment. The experiments were performed on a AMD Athlon XP 1880+ with 512 MB RAM running SuSE Linux 7.3 with Kernel version 2.4.16. The Servlet performed a query on a local MySQL database in version 3.23.55.

The measurements consisted of a client sending 1000 consecutive requests to the Servlet. The think time between two requests was 100 ms. Three independent experiments were performed using a uninstrumented Servlet, a manually instrumented Servlet, and a Servlet instrumented using our aspect-oriented approach. The results of the measurements are shown in figure 1. The values are based on measuring the overall processing time of the Servlet within the Tomcat Web container.

At first glance, the overhead of the instrumentation (manual and aspect-oriented) is quite high especially for performance measurements. The overhead is caused by the additional instrumentation code that was inserted into the Servlet. Since the functionality of the Servlet is minimal, as indicated by the mean response time of about 1 ms, the execution time proportion of the instrumentation code is considerably high. Therefore, the measurements can be regarded as representing the worst case. In real world Servlets with more complex application logic, the overhead will be considerably smaller. The overhead of the aspect-oriented instrumentation is slightly higher than the manual instrumentation. However, the aspect-oriented instrumentation offers a number of advantages such as cost effectiveness, no code pollution of application code, and separation of concerns. These advantages outweigh the slightly higher overhead.

We also determined the overhead from an end-user perspective, i.e., the response time as seen by the client (see figure 2). Here, the overhead of the instrumentation code went down to about 7% for manual instrumentation and about 8% for the aspect-oriented instrumentation. This overhead is within acceptable

| | (1) without instrumentation | (2) with manual instrumentation | (3) with aspects |
|---|---|---|---|
| mean (ms) | 1.145 | 1.473 | 1.536 |
| std. dev. (ms) | 0.954 | 1.247 | 1.274 |
| % increase | | 28.646 % | 34.148 % |

**Fig. 1.** Processing times of the Servlet code

| | (1) without instrumentation | (2) with manual instrumentation | (3) with aspects |
|---|---|---|---|
| mean (ms) | 4.059 | 4.176 | 4.426 |
| % increase | | 7.203 % | 8.063 % |

**Fig. 2.** Client side response times

boundaries for performance measurements. Again, these values reflect a worst case scenario, since the Servlet contains only minimal application functionality.

The results of the performance evaluation show that the aspect-oriented approach is very appropriate for performance instrumentation since its overhead is only slightly higher than the overhead caused by a manual instrumentation. The higher overhead of the aspect-oriented approach is due to the fact that it involves more instrumentation code than the manual instrumentation. The AspectJ environment generates a class representing the code of the aspect. In the class that has to be instrumented, hooks to this aspect class are added.

In principle, the number of measurement points within a component should be as minimal as necessary in order to keep the overhead as low as possible. Alternatively, the implementation of the ARM library and the AOP environment could be optimised which would require modifications of their source code. This was not the focus of our work.

## 5 Conclusions and Future Work

The increasing importance of Service Level Management implies a strong demand for management instrumentation of application components.

In this paper, we presented an aspect-oriented approach for instrumenting application components. The instrumentation code is encapsulated as an aspect and woven into the application code during compile time. Except for the use of a different compiler, the aspect weaver, the instrumentation process is completely transparent to the application developer. Thus, he is relieved from the burden of manual instrumentation and can concentrate on the application logic. We achieve separation of concerns by decoupling the application logic from the management logic. Thus, the efficiency of the overall software development process is increased, since separate concerns can be treated in separate tasks performed by different experts. An additional strong advantage of our approach is, that the aspect code can be reused in other applications. This is usually impossible in conventional instrumentation approaches.

For demonstrating our approach we chose the performance instrumentation of a Servlet in a typical e-business application scenario. Concretely, the duration of JDBC database queries was measured using the ARM API. The efficiency of the aspect-oriented instrumentation is demonstrated by a series of measurements. The instrumentation approach has been compared to an uninstrumented Servlet and a manually instrumented Servlet. The results show that in a worst case scenario the overhead of the aspect-oriented approach is only slightly higher

than the overhead of a manual instrumentation. This demonstrates that clarity of program structure and support for code reuse can be achieved without a substantial loss of efficiency.

Since our aspect-oriented instrumentation approach has demonstrated its viability and effectiveness, we are optimistic to apply the approach successfully to other management problems as well. So far, we also used AOP for instrumenting Web Services with ARM. There we chose a dual approach, i.e., we manually instrumented the Web Services platform and used AOP to instrument the Web Service itself. The results are comparable to the results published in this paper. Our future work will concentrate on using AOP to tackle management problems other than ARM instrumentation, e.g., we are working on a transparent integration of JMX instrumentation into EJBs. We expect to gain general insights into the use of AOP in management applications and to understand the limitations of the approach.

## Acknowledgements

The authors like to express their gratitude to Alexander Hoffmann, member of the Distributed Systems Lab at Fachhochschule Wiesbaden - University of Applied Sciences, Germany, for his helpful discussions and implementation work.

## References

[1] Lewis, L.: Managing Business and Service Networks. Kluwer Academic Publishers (2001) 209

[2] Sturm, R., Morris, W., Jander, M.: Foundations of Service Level Management. SAMS Publishing (2000) 209

[3] Verma, D.: Supporting Service Level Agreements on IP Networks. Macmillan Technical Publishing (1999) 209

[4] Katchabaw, M. K., Howard, S. L., Lutfiyya, H. L., Marshall, A. D., Bauer, M. A.: Making Distributed Applications Manageable Through Instrumentation. In: $2^{nd}$Second International Workshop on Software Engineering for Parallel and Distributed Systems (PDSE'97). (1997) 210

[5] Mehl, O., Becker, M., Köppel, A., Paul, P., Zimmermann, D., Abeck, S.: A Management-Aware Software Development Process Using Design Patterns. In: $8^{th}$ IFIP/IEEE International Symposium on Integrated Network Management (IM 03). (2003) 579–592 Colorado Springs, USA 210

[6] Distributed Management Task Force: Common Information Model (CIM) Specification. (1999) Version 2.2. 210

[7] Debusmann, M., Schmid, M., Kröger, R.: Generic Performance Instrumentation of EJB Applications for Service-Level Management. In Stadler, R., Ulema, M., eds.: $8^{th}$ IEEE/IFIP Network Operations and Management Symposium (NOMS). (2002) Florence, Italy 210

[8] The Open Group: Systems Management: Application Response Measurement (ARM). (1998) Open Group Technical Standard, Document Number: C807 210

[9] Sun Microsystems, Inc.: The Java Management Extensions Instrumentation and Agent Specification, v1.0. (2000) 210

[10] Debusmann, M., Schmid, M., Kröger, R.: Measuring End-to-End Performance of CORBA Applications using a Generic Instrumentation Approach. In: $7^{th}$ IEEE Symposium on Computers and Communications. (2002) Taormina/Giardini Naxos, Italy 210
[11] Hauck, R.: Architecture for an Automated Management Instrumentation of Component Based Applications. In: $12^{th}$ International Workshop on Distributed Systems: Operations & Management (DSOM'2001). (2001) 210
[12] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J. M., Irwin, J.: Aspect-Oriented Programming. In: European Conference on Object-Oriented Programming (ECOOP), Springer (1997) 211
[13] Becker, C., Geihs, K.: Generic QoS Support for CORBA. In: $5^{th}$ International Symposium on Computers and Communications (ISCC 2000), Antibes, France, Springer (2000) 211
[14] Hauck, F. J., Becker, U., Geier, M., Meier, E., Rastofer, U., Steckermeier, M.: AspectIX: A quality-aware, object-based Middleware Architecture. In: New Developments in Distributed Applications and Interoperable Systems (DAIS'01), Kluwer (2001) 115–120 211
[15] The AspectJ Team: AspectJ. Xerox Corporation. (2002) http://www.eclipse.org/aspectj/ 211
[16] The AspectJ Team: The AspectJ Programming Guide. Xerox Corporation. (2002) http://download.eclipse.org/technology/ajdt/aspectj-docs-1.0.6.tgz 211
[17] Debusmann, M., Schmid, M., Schmidt, M., Kröger, R.: Measuring Service Level Objectives in a complex Web-based e-Business Environment. In: $10^{th}$ HP OpenView University Association Workshop (HPOVUA). (2003) Geneva, Switzerland 214
[18] tang-IT Consulting GmbH: tang-IT Application Response Measurement (ARM). (2003) http://arm.tang-it.com/ 217

# Discovering Dynamic Dependencies in Enterprise Environments for Problem Determination

Manish Gupta[1], Anindya Neogi[1], Manoj K. Agarwal[1], and Gautam Kar[2]

[1]IBM India Research Lab., New Delhi
{gmanish,anindya_neogi,manojkag}@in.ibm.com
[2]IBM Watson Research Center, New York
gkar@us.ibm.com

**Abstract.** In order to reduce mean time to recovery (MTTR) in heterogeneous enterprise environments it should be possible to easily and quickly determine the root cause of a problem detected at a higher level, e.g. through response time violation of a transaction category, and resolve it. Many problem determination applications use a component dependency graph to pinpoint the root cause. However, such graphs are often manually constructed. This paper introduces a simple non-intrusive technique based on mining of existing runtime monitored data, to construct a dynamic dependency graph between the components of an enterprise environment. The graph is traversed to identify nodes that are the cause of response time related problems.

## 1 Introduction

Typically dependency models of system hardware and software are analyzed for problem determination and impact analysis in complex enterprise environments. Prior work talks about explicit middleware instrumentation [5], or internal instrumentation of the components (via ARM [2]) for obtaining system dependencies. These methods are time consuming and are difficult to apply in legacy environments. The main contribution of this paper is in showing how existing performance monitoring infrastructure available in middleware, such as web application servers and database servers, can be used in discovering dependencies between the various components of a system. Management clients can poll the middleware for performance metrics, such as total number of requests to a component, average response time of a component, etc. This paper proposes a data-mining algorithm that uses this performance data for obtaining "probabilistic" dependencies between components. An online algorithm for discovering and updating these dependencies between components is provided. Because of the probabilistic nature of these dependencies, "false" dependencies may arise and therefore we show how a problem determination application can use the dependencies effectively. Dependencies can be of various types [17], but this paper focuses on finding runtime software/service dependencies among the following components: URLs, servlets, EJBs, and SQLs in a web application.

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 221-233, 2003.

The rest of the paper is organized as follows. Section 2 compares our work with related research in dependency extraction. Section 3 presents the overall prototype system setup for dependency extraction, storage, and usage by management apps. In Section 4, we provide an algorithm for computing dependencies from the management data obtained from the managed resources and show how to use these dependencies for problem determination. Section 5 provides experimental validation of the algorithm on our testbed. Section 6 concludes the paper.

## 2 Related Work

A dependency graph of a system may be obtained using direct or indirect methods [3]. Direct methods rely on a human or a static analysis program to analyze system configuration, installation data, and application code to compute dependencies. However, it is unsuitable to apply such methods in large and heterogeneous systems because they are system specific and do not provide runtime dependency information. Indirect methods operate at runtime, and may be intrusive, semi-intrusive, or non-intrusive with respect to the operational system in the manner they extract dependencies.

An example of an intrusive technique is one that relies on code instrumentation such as ARM [2]. Dependencies are calculated by correlating data gathered during the flow of transactions through various components. eWLM [1] is a workload manager that uses ARM to instrument the underlying components and extract a component dependency graph. PinPoint [5] is a problem determination framework, where coarse-grained client requests are tagged as they travel through and discover the components of an enterprise system. Tagging requires middleware-level instrumentation to pass the request ID between components, similar to ARM. There are certain other approaches [12], [15], and [18] that use instrumentation of application to get dependencies. They instrument the application code such that some probes or hooks are available to management application to get the data out of managed object and to exert control over the managed object.

A key problem with the above approaches is that they may be unusable in situations where multi-vendor components are used and in places where even transaction correlation code cannot be inserted into the system for security, licensing, or other technical constraints. Unless all components adhere to standards, such as ARM, instrumentation based approaches cannot be deployed on a large scale. This motivates the requirement of semi or non-intrusive approaches. An example of a semi-intrusive approach is Active Dependency Discovery [4], where perturbation/fault-injection is used to obtain dependencies.

In addition, Ensel [8] has also suggested using Neural Networks technology to automatically generate dynamic and cross-machine dependency graphs while monitoring is active. The technique however does not provide any evidence of causality and only detects correlation. However, at the time of this writing, there are no details available regarding the training of such networks and experimental or theoretical analysis of the accuracy and precision of the method. Steinder et al. [19] have also used the concept of belief networks for fault localization in network services built on complex communication topologies. The technique is specific to network services and the bipartite graph is developed specifically for problem determination.

Our approach falls in the non-intrusive category because we do not instrument any application and use whatever instrumentation is provided by the vendor of the middleware for generating dependencies between various monitored resources. We rely on the fact that most vendors provide some built-in instrumentation for monitoring statistics primarily for accounting and performance tuning purposes. The statistics, at a minimum, include invocation and average execution time counters for the monitored resources. In Section 4 we will see that, in order to compute a probabilistic dependency between any two monitored resources, knowing at least the above two statistics is sufficient. Also in Section 4 we will see that our technique is independent of the actual "type" of the resource such as servlets, EJBs, SQLs, and database tables and therefore can be applied to other resources.

There are several papers that talk about problem determination and root cause analysis using dependency graphs [2][6][8][10][16][22]. Yemini et.al. [22], Choi et.al. [6], and Gruschke [10] assume the existence of a dependency graph and show how incoming alarms and events may be mapped to nodes of the graph and how the graph may be used to identify dependent nodes, which are the likely root cause of problems. Katker [16] also uses the graph for systematic analysis of a problem to identify the root cause in the network fault management domain. Once we obtain the basic probabilistic dependency graph with potentially large number of edges, we propose a technique to traverse it to quickly identify the root cause of response time related problems in generic systems.

Hellerstein and Ma [13] have applied data-mining algorithms for discovering useful patterns in historical system event data. They discuss how data-mining can be used to identify actionable patterns and in particular they present algorithms for three kinds of frequently occurring patterns in event data. Thoenen et al. [20] have developed an event management and design methodology that has been widely used. The core of this methodology is a graphic representation of the roles and dependencies or relationships between events.

## 3 System Architecture and Design Issues

We have built a prototype system to demonstrate the effectiveness of the algorithm proposed in this paper. The prototype system development is guided by two principles: reuse of existing monitoring information and separation of management interface from the business interface. The first principle is motivated from the fact that most of the existing enterprise code (including legacy code) is not ARM-instrumented and its owners may be averse to introducing instrumentation code in their applications as well. But the middleware that runs this code may provide performance data that can be used to "guess" dependencies between the various components. Furthermore, our technique can come handy to obtain dependencies between components running on middleware from different vendors. The second principle is motivated by the general trend [9] in the industry to separate the management interface from the business one and DMTF's CIM [7] is one such popular effort to achieve this.

The system consists of three tiers as shown in Figure 1. The bottom layer is the managed system, which currently consists of WAS and DB2 running a TPC-W bookstore application. Both the WAS and DB2 have a monitoring API through which run-

time statistics (such as performance-monitoring counters) can be extracted by a local agents A1 and A2, respectively, and sent to a correlation engine (CE) in the central management layer. The data-mining algorithm implemented in CE takes this monitoring data to compute the dependencies between the monitored components. The topmost layer consists of management applications that pull raw dependency data from the repository through CIMOM for various uses. In our test-bed, the application and the database servers run on the same machine and hence both A1 and A2 send data that is time stamped using the same clock. In our next prototype version we will run a distributed system where the issues of clock synchronization will be handled.

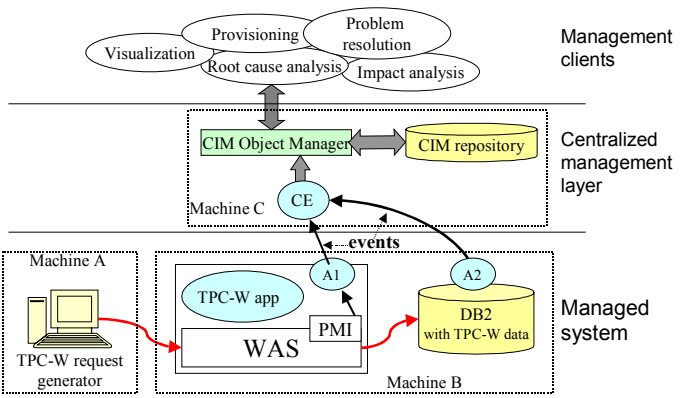



**Fig. 1.** Prototype System Setup: The IBM WAS v4.0 and IBM DB2 UDB v7.1 are installed on machine B (**2GHz, 1GB)**. WAS v4.0 runs the TPC-W (see [21]) bookstore application consisting of 14 servlets and 46 SQLs and a database of 10,000 books. Machine A (**600MHz, 512MB**) is used to run the remote browser emulator (RBE) or the TPC-W request generator and sends URL requests (consisting of 50% buy and 50% browse) to machine B, which also runs the IBM HTTP server. On Machine C (**2GHz, 1GB**) runs the CIMOM and the CE. All Machines are connected over a 100 Mbps Ethernet

The current prototype monitors only a small subset of the objects/resources pertaining to WAS and DB2, namely, the URLs, servlets, EJBs, and SQLs[1]. In other words, the agent A1 provides events for URLs, servlets, and EJBs only, and A2 provides the events for SQLs. Each of the URLs, servlets, and EJBs are modeled using the CIM schema in the J2EE Management Specification [14]. The SQLs are modeled using the *CIM_UnitOfWorkDefinition* class in the CIM Metrics Model [7]. In order to capture the dependency information between the above objects, as for example the dependency between a servlet and an EJB, we introduced additional association classes, which derived from *CIM_dependency*. As and when the instances of the CIM

[1] In an enterprise system the web transactions (or URLs) are typically serviced by servlets which in turn may invoke EJBs or directly execute SQLs. The TPC-W bookstore application [21], on which we have tested our technique, comprises a collection of servlets each of which issues SQLs directly to the database. Also, the WAS' and DB2's monitoring API allows us to obtain the performance metrics that are sufficient for computing the probabilistic dependencies.

classes pertaining to the URLs, servlets, EJBs, SQLs, and their dependencies are discovered, the CIM repository in the middle layer in Fig. 1 is populated with them.

## 4 Dependency Graph Extraction and Its Usability

This section presents the algorithm to extract dependency relationships from existing monitoring data. It also describes how the algorithm may produce false dependencies and the manner in which a problem determination application may cope with them.

**Dependency Definitions.** Consider any two components, say *A* and *B*, where *A*, for example, may be a servlet, and *B* an EJB. In the general case, *A* is said to be dependent on *B*, if *B*'s services are required for *A* to complete its own service. A weight may also be attached to the directed edge from *A* to *B*, which may be interpreted in various ways, such as a quantitative measure for the extent to which *A* depends on *B* or how much *A* may be affected by the non-availability or poor performance of *B*, etc. Any dependency between *A* and *B* thus arises from an invocation of *B* from *A*, which may be synchronous or asynchronous. Note that the algorithm described in this paper handles synchronous invocations only.

Corresponding to each servlet, EJB, or SQL request an event, carrying the essential statistics such as the number of requests and average response time *so far* to the component, is received by the CE (in Fig. 1). Associated with each such event is an activity period, which for the purposes of the discussion here can be taken to be a time interval with its start time and end time being the start time and end time, respectively, of the request to the component represented by the event (see [11] for more detail). We say that an activity period $[b_1, b_2]$ of the component *B* is *contained* in an activity period $[a_1, a_2]$ of the component *A* if $a_1 \leq b_1$ and $b_2 \leq a_2$. Finally, our definition of dependency between two the components *A* and *B* is as follows: we say that *A depends on B with strength p* (or $A \xrightarrow{p} B$ [2]) if the probability that a given activity period of A contains an activity period of B is *p*.

The above definition of containment of an activity period into another captures the following dependency type that we are interested in.

- *A* invokes the services of *B*, *directly* or *indirectly*, and *A* finishes only after the invocation of *B* returns. An example of *A* invoking *B* directly is: *A* calls a method of *B* and waits for the call to return. On the other hand, we say *A* invokes *B* indirectly if *A* directly invokes some component *C*, which in turn can either directly or indirectly (a recursive definition) invoke *B*.

A dependency of the above type is what we call a *true* dependency, and any other type of dependency that is captured by our dependency definition is what we term as a *false* dependency. The motivation for choosing the definition for *p* given above is that we believe that the number *p* comes very close to the probability that a given execution of *A* invokes, directly or indirectly and at least once, the component *B*, and fin-

[2] For expositional simplicity we occasionally omit the strength value on the arrow and simply say A→B.

ishes only *after* the invocation to *B* returns. If *A* calls *B*'s methods on each request to *A* then, as per our definition, *p* is 1.0. If only 20% of the requests to *A* result in calls to *B* then *p* is 0.2. Note that multiple calls or containment of activity periods of *B* per request to *A* are counted as a single call to *B*. In Fig. 2 in [11] we provide another example to describe this concept of containment that not only captures the notion of true dependencies but also of false ones.

In the following subsection, we present an algorithm for computing the dependencies (based on our definition above) for any given component, say *A*. The true dependencies for the component *A* include all the components that are directly and indirectly invoked by *A*. Furthermore, we expect that this algorithm, in the process, also computes false dependencies. A natural *two-level* dependency graph, resulting out of the above definition of a dependency, has a node at level one with all its true and false dependencies situated at the second level. Fig. 3 in [11] shows a two-level dependency graph. Looking at only the two-level graph of a component suffices because all the monitored components that *are* dependencies of the component are captured at the second level of the graph and there is no need to traverse the two-level graphs of any of the dependencies (i.e., the two-level graphs of the antecedent nodes). Furthermore, as the transitive property (see [11]) of dependencies may not hold in general, a two-level graph helps us to consider *only* the true dependencies of a component.

The number '*p*' in $A \xrightarrow{p} B$ (hereafter, referred to as the *p*-value) depends on the business logic in *A* and the workload applied to the enterprise environment containing the component *A*. As the logic and the workload change so will the *p*-value. Assuming that the logic and the workload do not vary with time, we now show how CE (see Fig. 1) estimates *p*-value from the event traces. From the definition of *p*-value given above, the straightforward way of estimating it is to calculate the fraction of the number of activity periods of *A* that contain some activity period of *B* from the event traces received for both *A* and *B*. That is, let #*A* denote the total number of activity periods of *A* seen so far, and #(*B*, *A*) denote the total number of activity periods out of #*A* that contain at least one activity period of *B*. Then the number #(*B*, *A*)/#*A* is an estimate of the *p*-value. It is desirable that as new events are received at the CIM repository (see Fig. 1) our algorithm (to be given in the next subsection) generates and updates the dependency graph online.

**Online Dependency Extraction Algorithm.** We will present the algorithm informally. Consider only a pair of event types; say servlet and SQL events, for expositional simplicity of the algorithm. Let $\Sigma_{sql}$ denote the set of all SQLs and $\Sigma_{servlet}$ denote the set of all servlets in a given web application. Let $A \in \Sigma_{servlet}$. The goal is to discover and update all the dependencies $A \xrightarrow{p} B$ where $B \in \Sigma_{sql}$. A key property that our algorithm uses that satisfied by the system in Fig. 1 is that the events from a given component (say servlet *A*) are received at CE (in Fig. 1) in the increasing order of their time stamps where the time stamp of an event is defined as the end time of the activity period represented by the event (see Section 4 in [11]). This also implies that SQL events received at CE are in the increasing order of their time stamps.

A few definitions are in order before we present the algorithm. We say that a servlet event is *fully processed* if all the SQL events that are contained[3] in the servlet event have been identified; otherwise we say that the servlet event is *partially processed*. The following event lists are maintained: *antecedentList* comprises events received for SQLs in $\Sigma_{sql}$, *dependentList* comprises events received for the servlet A, and *dependencyList* maintains the list of dependencies, of A, obtained so far. The *dependencyList* variable, essentially, keeps the list of all those SQLs in $\Sigma_{sql}$ for which dependencies have been detected so far. We define *LPP* as an event pointer that at any time holds reference to the currently lowest time stamped partially processed event in *dependentList*. We also define *HSQ* as an event pointer that at any time holds reference to the currently highest time stamped event in *antecedentList* subject to its time stamp being less than or equal to the time stamp of the servlet event referenced by *LPP*. A counter *#A* (initialized to zero) keeps the count of the number of fully processed servlet A events so far. For each SQL B in the *dependencyList* we set a counter #(B, A) (initialized to 1) that counts the number of servlet A events so far that contain at least one SQL B event. For any event pointer *e* the notation *e*.id, *e*.timestamp, *e*.starttime, *e*.next, and *e*.prev, respectively, refer to the following attributes of the event referenced by *e* in an event list, viz., the identifier, the time stamp, start time, reference to the event immediately *after* the event referenced by *e*, and reference to the event immediately *before* the event referenced by *e*. Both *LPP* and *HSQ* are initialized to null (see [11] for a complete description).

**Dependency Algorithm** (an outline)**:** The main algorithm preserves the following property at all times: all servlet events in the *dependentList* having time stamps less than *LPP*.timestamp are fully processed and the rest are partially processed, and *HSQ* always points to the highest time stamped SQL event, currently in the *antecedentList*, having timestamp less than or equal to *LPP*.timestamp. In the non-boundary case, i.e., when both *LPP* and *HSQ* are non-*null*, on an arrival of an SQL event with time stamp less than *LPP*.timestamp we set *HSQ* to this event, otherwise *LPP* is set to *LPP*.next, i.e., to the *next* servlet event, whenever it becomes available, and the count #A is incremented by one. In the latter case, we continue to scan the SQL events to the left of *HSQ* until we reach an SQL event whose end time is less than the start time of the new servlet event pointed to by *LPP*; thereafter we scan the SQL events on the right to *HSQ* and compute the new value for *HSQ*. Each time we compare an SQL event with the servlet event pointed to by the *LPP* we also check for containment of the activity period of the SQL event into the servlet event, updating the *dependencyList* and corresponding frequency counts if required. This process is repeated with the new *LPP* and *HSQ*. The algorithm outputs estimates of *p*-values by computing for each SQL B in the *dependencyList* the number #(B, A)/#A.

It is easy to see that the algorithm has the desirable property that the number of event comparisons needed in order to ascertain all the events contained in another event is minimum. In other words, if *n* is the number of activity periods contained in a given activity period then the algorithm finds that out in O(*n*).

---

[3] An event is *contained* in another event if the former's activity period is contained in the latter's.

**Problem Determination (PD) Using Probabilistic Graph.** The presence of false dependency edges in a two-level graph complicates the traversal order of edges in the two-level graph of a component. Note that the edge weights (or strength), in a sense, correspond to the "likelihood" of *A* depending on *B*. False dependencies may be caused by false containment generated due to concurrency of transactions flowing through the server components. Clearly, traversal of false dependencies reduces the performance of the PD algorithm and increases the MTTR.

We introduce another statistic of a dependency $A \rightarrow B$ called, *r*-value, which we use along with the *p*-value to achieve an improved ordering of the dependencies of a node *A*. We define *r*-value as the probability that a given activity period of the component *B* is *contained in* an activity period of the component *A*, where the definition of containment is as described earlier. We estimate *r*-value as follows. Let, $\varphi(A, B)$ denote the number of activity periods of *B*, seen so far, which were found to be contained in at least *one* activity period of *A*. Finally, let #*B* be the total number of activity periods of *B* seen so far. We take the number $\varphi(A, B)/\#B$ as an estimate of *r*-value. To see the difference between *p*-value and *r*-value, consider the earlier example in which *A* invokes *B* only 5% of the time *A* is active. Suppose *A* is the *only* component that invokes *B* in the application, then the *p*-value will be close to 0.05 whereas the *r*-value will be close to 1.0. Another illustration of the difference between *p* and *r*-values is given in Table 1 in [11]. The algorithm for computing *p*-values given earlier an be easily tailored to compute the *r*-values as well.

We now introduce a heuristic that uses the two statistics, the *p*-value and the *r*-value, to order the dependencies of a node: we sort the dependencies of a node *A* in the *non-increasing* order of $\mathbf{max(}p, r\mathbf{)} + pr$**.** The rationale for choosing the heuristic is the following. The first term coarsely sorts the dependencies of a node so that edges with high *p*- or high *r*-values are catapulted up in the order. A dependency is more likely to be true if at least one of these is high. The product term performs finer grain sorting among equals. A dependency is more likely to be true if both *p*- and *r*-values are high compared to the case when only one of the values is high. In this heuristic the true dependencies, which have low *p- and r*-values, will be penalized. For example, if a servlet *X* executes an SQL *Y rarely*, but the SQL *Y* is *frequently* executed by some other servlets in an application, then the dependency *Y* of *X* will appear lower down in the sorted list of dependencies of *X*, because both *p* and *r* values will be small. However, there is an increased likelihood that if the problem in *X* is due to *Y*, then other servlets that more frequently call *Y* also have the same problem and we are able to identify *Y* through one these servlets.

## 5 Experimental Evaluation

This section presents an experimental evaluation of the dependency algorithm described in Section 4 on the testbed, which consists of the three machines as described in Figure 1.

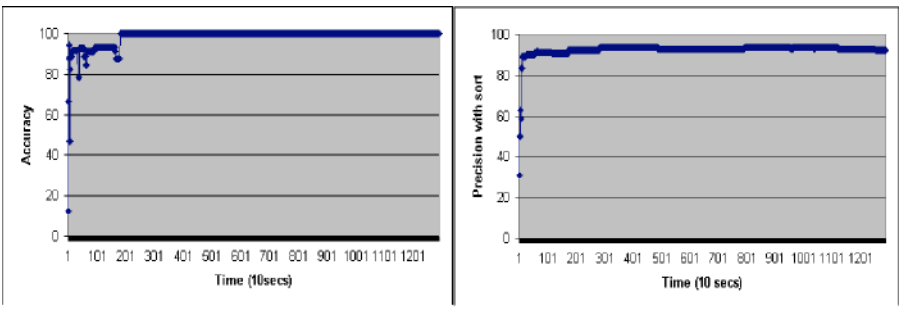



**Fig. 2.** The variation in (a) Accuracy and (b) Mean Precision with time for a particular load of 50 customers. Accuracy and precision values stabilize as the entire graph is discovered. They remain stable with the workload, thus the computed graph can be used for long stable periods. It can be observed that the method is 100% accurate (Fig. 2 (a)) as expected, however precision stabilizes to around 98% (Fig. 2 (b)). The convergence time is also within a few minutes and closely tracks workload stabilization

**Definition of Accuracy and Precision.** The aim of the experiments is to discover the dependencies between servlets and SQLs. In our testbed, there are 54 true servlet-to-SQL dependencies out of the potential set of 644 dependencies. So if we discover all 54 true dependencies then the accuracy is 100%. *Accuracy* is defined as the percentage of the true dependencies discovered. *Precision* is defined based on the two-level graph traversal order in Section 4. In the sorted dependency list of a node having $n$ edges, the edges are numbered (starting from the first edge) from 1 to $n$ with $m$ ($\leq n$) being the last true dependency in the list. We assign a weight of $m$-$i$ to the dependency labeled $i$, where $1 \leq i \leq m$-1. The sum of the total weights from 1 to $m$-1 is therefore $w_{tot} := m(m\text{-}1)/2$. Out of $w_{tot}$ the total contribution of weight coming from false dependencies is defined as $w_f := \sum_{i<m,\, i \text{ is a false Dependency}} (m-i)$ . Finally, we define percentage *node-precision* as $100\,(1-\frac{w_f}{w_{tot}})$. Observe that this definition penalizes a false dependency more if it occurs higher in the list. The precision value reported in the following experiments is the *mean* percentage node-precision over the 14 servlets.

Accuracy and precision are measured on the testbed as follows. We have instrumented the TPC-W application with an in-house developed transaction correlation code to find the actual set of the dependencies that should be discovered at each point in time as user transactions flow through the system. Thus accuracy and precision can be computed at each time point by comparing the dependency information generated by our dependency algorithm with the accurate information generated by instrumented transaction correlation code in the same experiment. For illustration see Fig. 4 for an experiment run with 50 simultaneous customers.

**Table 1.** Precision and Accuracy with Load for traffic mix: 50% buy and 50% browse. The WAS is configured to fork threads on demand with a minimum of 25 pre-forked threads. As customer load is increased, the number of simultaneously active threads grows, and the thread pool size is also automatically increased by the system beyond 25, if needed. The number of customers is increased till a significant percentage of URL requests timed out due to load. The WAS machine can support up to 200 customers. Hence, the experiments have been run only up to 200 customers as a high load case. Each experiment was run for one-hour duration. If we run them for longer duration, we expect precision values to go up

| Load: #Customers | #Avg active threads (Avg thread pool size) | Mean Precision % (std dev σ) | Accuracy % |
|---|---|---|---|
| 5 | 2.2 (25.0) | 100 (0) | 100 |
| 25 | 3.5 (25.0) | 99.94 (0.22) | 100 |
| 50 | 4.6 (25.0) | 98.33 (3.12) | 100 |
| 100 | 10.6 (25.0) | 81.86 (21.00) | 100 |
| 150 | 17.1 (26.4) | 71.52 (32.43) | 100 |
| 200 | 20.0 (31.9) | 62.62 (26.82) | 100 |

**Table 2.** Effect of polling rate on precision and accuracy with a fixed load of 100 customers. Precision is more sensitive to polling rate than accuracy. At lower polling rates, even though accuracy is 100% and the management overhead drops, the precision also goes down. At higher load, high polling rate is required to maintain high precision, however it also increases the overhead, which is undesirable at high load

| Polling interval in millisecs | Mean Precision % (std dev σ) | Accuracy % |
|---|---|---|
| 50 | 88.71 (18.75) | 100 |
| 100 | 81.86 (21.00) | 100 |
| 500 | 77.03 (29.82) | 100 |

**Table 3.** Overhead Measurement in terms of percentage increase in throughput and response time by varying polling interval and customer load. Each reading corresponds to an experiment that was run for 3 hours

| Simultaneous Customers (load) | Polling Interval (milliseconds) | | | |
|---|---|---|---|---|
| | 100 | | 500 | |
| | Throughput (%) | Response Time (%) | Throughput (%) | Response Time (%) |
| 10 | 2 | 10 | 0 | 3 |
| 200 | 9 | 33 | 8 | 25 |

**Accuracy and Precision for TPC-W Bookstore Application.** As more multiple transactions proceed simultaneously, it is harder to separate true dependencies from false ones using containment relationship. The degree of concurrency of transactions on WAS may be increased by increasing the number of simultaneous customers or browser emulators, thus keeping more threads in the thread pool simultaneously active. Table 1 shows the variation of accuracy and precision values with increasing customer load and concurrency. The algorithm shows 100% accuracy under all loads. Precision values decrease with increasing load. The standard deviation (σ) for precision is low at low loads but rises at higher loads. Thus dependency extraction can be

performed at low to medium loads and stopped, if the precision level of the graph is to be maintained. Table 2 shows that a higher polling rate[4] leads to increased management overheads. Table 3 suggests that higher load implies higher overhead of polling.

## 6 Conclusion

In this paper we propose a new method for discovering dependencies between system components using non-intrusive techniques. To demonstrate the usability of the method we performed extensive experimentation on an e-business setup that mimics the operation of a typical storefront system using the TPC-W benchmark. The algorithm for computing the dependencies is experimentally shown to perform well with accuracy being 100% both at low and high loads, and precision decreasing with increasing load. For instance the bookstore application the precision is around 98% for a load of 50 simultaneous customers.

The nature of the algorithm is such that in addition to discovering all the true dependencies, it also discovers false dependencies. To preclude usage of the false dependencies we proposed a sorting heuristic, which increases the probability of placing the true dependencies higher in the sorted list. As expected, our experiments show that the precision increases as load decreases. A decrease in load causes a decrease in concurrency in the system and hence decreases the chance of discovering a false dependency. Thus, false dependencies can be artificially minimized if sufficient time gap exists between two successive transactions. In order to present various views of the dependencies to a system administrator, the dependencies of a component can be sorted in a manner that also uses the execution time or the frequency of invocation, or any other attribute of the antecedent components. We intend to look into such issues in future. Last but not the least, we intend to explore how to relax the containment assumption and capture dependencies in an asynchronous transaction environment.

## Acknowledgements

The authors like to thank Yuan Feng (IBM Watson Research Center), Sugata Ghosal (IBM India Research Lab.) and Parviz Kermani (IBM Watson Research Center) for critical feedback of the paper.

## References

[1] J. Aman, C.K. Eilert, D. Emmes, P. Yocom, D. Dillenberger, "Adaptive Algorithms for managing a distributed data processing workload," *IBM Systems Journal, vol.36, no.2*, 1997.

[2] "Systems Management: Application Response Measurement (ARM)," OpenGroup Technical Standard C807, UK ISBN 1-85912-211-6 July 1998, http://www.opengroup.org/products/publications/catalog/c807.htm

[4] Management data from WAS is obtained through polling at regular intervals. See [11].

[3] S. Bagchi, G. Kar, J.L. Hellerstein, "Dependency Analysis in Distributed Systems using Fault Injection: Application to Problem Determination in an e-commerce Environment," *12th International Workshop on Distributed Systems: Operations & Management* 2001.

[4] Brown, G. Kar, and A. Keller, "An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in Distributed Environment," *International IFIP/IEEE Symposium on Integrated Network Management*, 2001.

[5] M.Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," *International Conference on Dependable Systems and Networks (DSN'02),* June 2002.

[6] J. Choi, M. Choi, and S. Lee, "An Alarm Correlation and Fault Identification Scheme Based on OSI Managed Object Classes," *In 1999 IEEE International Conference on Communication*s, Vancouver, BC, Canada, 1999, pp. 1547–51.

[7] CIM: http://www.dmtf.org/standards/standard_cim.php

[8] Ensel, Christian, "New Approach for Automated Generation of Service Dependency Models," *Second Latin American Network Operation and Management Symposium, LANOMS,* 2001.

[9] J. A. Farrell and H. Kreger, "Web services management approaches," *IBM Systems Journal*, VOL 41, NO 2, 2002.

[10] Gruschke, "Integrated Event Management: Event Correlation using Dependency Graphs," Proceedings of 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 98), October 1998.

[11] M. Gupta, A. Neogi, M. K. Agarwal, and G. Kar, "Discovering Dynamic Dependencies in Enterprise Environments for Problem Determination," *IBM Research Report*, RI03010, 2003.

[12] P. Hasselmeyer, "Managing Dynamic Service Dependencies," Proceedings of 12th International Workshop on Distributed Systems: Operations & Management (DSOM) 2001.

[13] J.L. Hellerstein and S. Ma, "Mining Event Data for Actionable Patterns," *The Computer Measurement Group*, 2000.

[14] Java 2 Platform, Enterprise Edition, http://java.sun.com/j2ee

[15] M. J. Katchabow et al., "Making Distributed Applications Manageable Through Instrumentation," *Journal of Systems and Software*, Vol. 45, 1999.

[16] S. Katker and M. Paterok, "Fault Isolation and Event Correlation for Integrated Fault Management," *Integrated Network Management V, Chapman and Hall*, May 1997.

[17] Keller and G. Kar, "Classification and Computation of Dependencies for Distributed Management," *5th IEEE Symposium on Computers and Communications (ISCC)*, July 2000.

[18] F. Kon and R.H. Campbell, "Dependence Management in Component-Based Distributed Systems," *IEEE Concurrency*, Vol. 8, No. 1, pp. 26-36, Jan-Mar 2000.

[19] M. Steinder and A.S. Sethi, "Multi-layer Fault Localization using Probabilistic Inference in Bipartite Dependency Graphs," Technical Report 2001-02, CIS Dept., Univ. of Delaware, Feb 2001.

[20] Thoenen, J. Riosa, and J. L. Hellerstein, "Event Relationship Networks: A Framework for Action Oriented Analysis for Event Management," *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA, May 2001, IEEE, New York (2001), pp. 593-606.
[21] TPC-W Wisconsin website, http://www.ece.wisc.edu/~pharm/tpcw.shtml
[22] S. Yemini, S. Kliger et al., "High Speed and Robust Event Correlation," *IEEE Communications Magazin*e, vol. 34, no. (5), pp. 82–90, May 1996.

# Bringing AgentX Subagents to the Operating System Kernel Space

Oliver Wellnitz and Frank Strauß

Technical University of Braunschweig
Institute of Operating Systems and Computer Networks
Mühlenpfordtstraße 23, 38106 Braunschweig, Germany
{wellnitz,strauss}@ibr.cs.tu-bs.de

**Abstract.** SNMP agents on conventional operating system platforms are mostly monolithic and implement Managed Objects in a single program. The concept of subagents makes it possible to delegate the implementation of Managed Objects to several subagents located close to the managed subsystems. All subagents are managed by a master agent. While this concept is well accepted for hardware subsystems of modular devices and for host services running in the user space, it is not yet applied for components of conventional operating systems.
This paper examines to what extent the IETF standard subagent protocol AgentX is suitable for the management of kernel components. For this purpose, on the Linux platform two subagents have been implemented within the kernel subsystems they manage. They communicate with a master agent in user space. The implemented software contains a generic intermediate layer which carries out AgentX protocol operations and access to Managed Objects. Based on this layer, the network interface subsystem and the Netfilter subsystem have been enhanced with management extensions.

## 1 Introduction

Network management is essential for the operation and supervision of medium to large computer networks and the Simple Network Management Protocol (SNMP) is the standard network management protocol of the Internet [3]. Current implementations of SNMP agents on conventional operating system platforms are mostly monolithic and rarely extensible. They run in user space and cover a broad spectrum of management information from high-level service management to low-level hardware device management. In many cases they have to gather information from the operating system kernel through different means such as system calls, device driver input/output control functions (ioctls) or special filesystems. These kernel interfaces can be difficult to handle because, for example, on many Unix systems the agent has to parse files from the `/proc` filesystem which have a structure that is likely to change in subsequent kernel revisions. These interfaces can in many cases also be incomplete in respect to the MIB which is to be implemented, e.g., they may lack attributes to uniquely

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 234–245, 2003.

identify instances of Managed Objects. Furthermore, in many cases SNMP notifications cannot be created efficiently because there is no mechanism in the methods mentioned above to notify user space processes upon certain events that occur in kernel space. So whenever an SNMP agent handles notifications, it has to use a polling strategy to gather information and detect changes itself. If the polling interval is set too small, polling data from the kernel wastes CPU time. On the other hand, setting the polling interval too large, the data in question may change two or more times during that time so that changes cannot be detected accurately. Another problem is that aside from the maintainer of a component the author of an SNMP agent must also know specific details about the managed component. If we could minimize the necessary amount of knowledge that a component maintainer needs to have about network management, he could provide a network management interface for his component all by himself, so that it is much more likely to keep the management plane in sync with the component.

A kernel implementation of AgentX subagents may overcome theses problems. Inside the kernel, a subagent has access to every available data structure. Additionally, it also can be synchronously and accurately triggered upon changes because it can call notification functions directly at the point where data is altered. Finally, a simple thus complete management interface for kernel subsystems would enable developers to implement and maintain any network management extension to their subsystem.

This paper is structured as follows: The next Section gives some background information about the subagent protocol AgentX. Section 3 describes the design and implementation of the kernel subagent architecture. It also gives some examples. Section 4 explains the implemented MIB modules and Section 5 gives a short evaluation of the presented architecture. Finally, Section 6 concludes the paper.

## 2 Agent Extensibility Protocol

Developed as a protocol to dynamically extend SNMP agents, Agent eXtensibility (AgentX) was published in January 2000 as an IETF Proposed Standard [4] and advanced to Draft Standard in 2002. The AgentX framework splits the role of an agent into two separate entities: A master agent, which is a traditional SNMP agent but with little or no direct access to management information, and a set of subagents, which have access to a mostly disjunct set of management information and no knowledge about SNMP. Master agent and subagents communicate through the AgentX protocol. The master agent thereby acts as a multiplexer and SNMP/AgentX protocol translator for the subagents. AgentX is transparent to SNMP managers and SNMP independent, which means that AgentX subagents can be combined with SNMPv1, SNMPv2c and SNMPv3 master agents. The upper half of Figure 2 illustrates this concept.

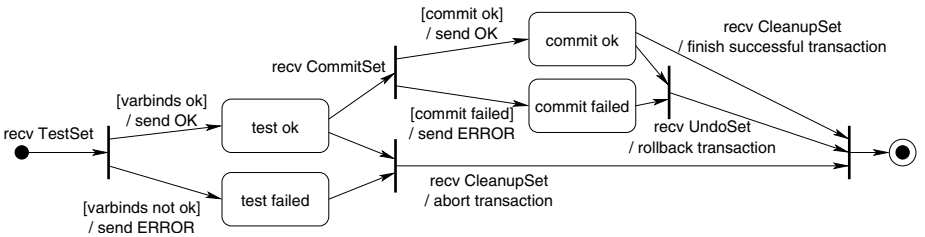



**Fig. 1.** AgentX Set transaction

The design of a single master agent to which one or more subagents connect requires that the master agent is already running when the subagents are initialized.

The AgentX architecture was designed to be simple in respect of authorization, privacy and encoding. It completely leaves the first two points to the master agent, which has to ensure that only allowed mangers can access or change management information. Because subagents usually reside on the same machine as the master agent a native byte-order encoding is used instead of the BER/ASN.1 encoding of SNMP.

While it was tried to keep AgentX simple, it offers full support for data retrieval (Get, GetNext and GetBulk) and data modification operations (Set) as well as notifications (Traps). AgentX uses a multiphase-commit so that SNMP Set operations remain atomic even if the Set request comprises objects located at several subagents. Figure 1 shows the different states in an AgentX set transaction.

AgentX transport mappings are specified for Unix domain sockets and TCP. Any other transport mechanism is likewise conceivable. Every AgentX connection is split up into several sessions, which in turn can convey several transactions.

There are also other subagent protocols: The SNMP multiplexing protocol SMUX [7] and the Distributed Program Interface DPI [2, 11] can be regarded as predecessors of AgentX. Furthermore, there are proprietary agent toolkits, such as EMANATE by SNMP Research Inc., that use their own master/subagent architecture.

## 3 Design and Implementation

Similar to traditional user space AgentX subagent toolkits such as NET-SNMP [1] and JAX [8], we propose a kernel subagent architecture which comprises two major parts: (a) a generic AgentX layer, which is MIB-unaware but omniscient in regard to the AgentX protocol and its variable types and (b) one or more management entities (kernel subagents) which implement the Managed Objects. There are no requirements for the master agent, thus any existing AgentX master agent that runs on the target platform can be used. Figure 2 gives an

overview of the SNMP/AgentX framework with kernel space extensions. In the following, we focus on the kernel elements in the lower right part of the figure, the master agent and user space subagents are not regarded any further.

### 3.1 The AgentX Layer

The AgentX layer is a mediator between the master agent and the kernel subagents. It implements the AgentX protocol to talk to the master agent in user space and it maintains the knowledge of all available kernel subagents which supply management information. The general idea is to put everything into the AgentX layer that can be done in a generic way. The AgentX layer is divded into the following three parts, which are also shown in the grey box of Figure 2.

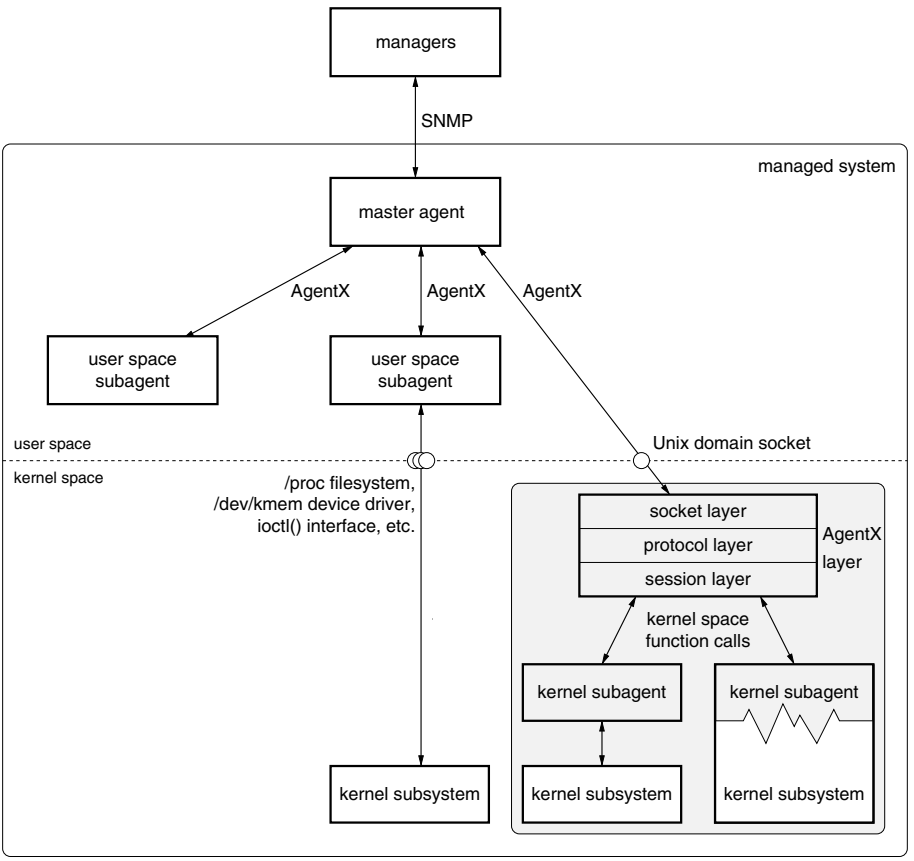



**Fig. 2.** SNMP/AgentX architecture with kernel space subagents

The socket layer forwards all AgentX PDUs between the protocol layer and the master agent. Its interface is very small and can easily be exchanged or modified to support other AgentX transport mappings.

The protocol layer is able to parse AgentX PDUs received from the master agent and to create AgentX PDUs, which are then sent through the socket layer to the master agent.

The session layer decides which kernel subagent is responsible for a request that has been parsed by the protocol layer. Similarly, the session layer assigns responses and notifications from kernel subagents to the right AgentX sessions and passes them up to the protocol layer. For this purpose, every kernel subagent has to register with the session layer prior to any AgentX communication. These registrations result in AgentX registrations.

When the master agent sends an AgentX request to the kernel, it is received by the socket layer and passed to the protocol layer which divides it into smaller pieces, which are simpler to process by the subagents. E.g., a GetNext request may contain more than one SearchRange so that multiple object instances can be retrieved by one request. This is handled as follows: The protocol layer creates an empty Response PDU. Then it takes the first SearchRange and dispatches it via the session layer to the corresponding kernel subagent. The response is appended to the Response PDU. This procedure is repeated until all SearchRanges were processed.

In contrast to userspace subagents where it is easy to ensure that the master agent is started first, the kernel and many of its subsystems are obviously initialized before the master agent program can be started. The implemented solution to this problem is to decouple the registration of kernel subagents to the AgentX layer from the AgentX registration to the master agent. This allows the kernel AgentX layer to delay the AgentX connection. Once the master agent is running a signal has to be sent to the kernel AgentX layer in order to setup the connection. The upper part of the sequence diagram in Figure 3 illustrates this procedure.

### 3.2 Kernel Subagents

Kernel subagents are closely attached to the kernel subsystems for which they implement the Managed Objects. Figure 2 shows the two different approaches of kernel subagents: The subagent can either be closely integrated within the kernel subsystem code, or the kernel subagent can be implemented separated from the kernel subsystem, e.g., as a separate kernel module, if the subsystem supports appropriate kernel level interfaces.

In both variants, a kernel subagent contains notification emitting functions and request callback functions. The notification emitting functions are called by other kernel functions upon certain events, so that they can construct a notification message and pass it to the AgentX layer. If the subagent is integrated with the managed kernel subsystem, notifications can easily be triggered from those functions that actually process kernel data in a way that should raise a

**Table 1.** AgentX PDU type to method translation

| AgentX PDU | *method* parameters |
|---|---|
| Get | GET & EXACT |
| GetNext | GET or GET & INCLUDE |
| GetBulk | GET or GET & INCLUDE |
| TestSet | TESTSET |
| CommitSet | COMMITSET |
| UndoSet | UNDOSET |
| CleanupSet | CLEANUPSET |

notification. On the other hand, if the kernel subagent is implemented as a separate module, it depends on the kernel subsystem to offer hooks so that the subagent can register for events that potentially raise notifications. However, such hooks are more likely to be available within kernel space than for feedback to traditional user space agents.

As described in Section 3.1, callback functions are registered with the AgentX layer at startup. They are called for all Get and Set requests. The signature of a callback function is defined as follows:

```
errorcode Callback(in oid, in method, in context, out result)
```

A callback function gets a single request *oid*, a *method* specifier (which is a set of named flags) and the SNMP *context* as input arguments. A buffer in which the callback functions returns a *result* is passed as the fourth argument. Every callback function returns an *errorcode*. The *oid* is the starting OID of a SearchRange or the exact OID of a variable binding (varbind). It always lies within the range for which the callback function was registered to the AgentX layer. When the AgentX layer receives an AgentX GetNext PDU and dispatches a SearchRange, it compares the starting OID of the SearchRange to the callback functions' registration points. If the starting OID is a lexicographical predecessor compared to the registration point of a callback function, it uses the registration point as the value for the *oid* argument and sets the INCLUDE flag in the *method* argument.

Callback functions are called with one out of five different methods, which is specified by an according flag in the *method* argument. There is one GET method for Get, GetNext and GetBulk requests, and four methods for the phases of Set transactions (TESTSET, COMMITSET, UNDOSET, and CLEANUPSET). Additionally, the *method* argument can hold two flags which describe the interpretation of the *oid* argument: The flag INCLUDE signals that the search range includes *oid* itself, if it is not set *oid* is excluded. The flag EXACT signals that the request is for the exact *oid* and not for any successor. AgentX GetBulk requests with Repeaters are split by the AgentX layer into several invocations of callback functions. This streamlines and simplifies the architecture, because for GetBulk Repeaters the results may come from different callback functions. It is

not a performance issue, since it affects only neglectable local function calls and not PDUs that would have to be transmitted. Table 1 shows how all AgentX request PDUs are translated into *method* parameters to callback functions.

### 3.3 Processing Get/GetNext/GetBulk Requests

The middle part of Figure 3 shows a sequence diagram that illustrates the processing of a GetNext request which is similar to the processing of Get or GetBulk requests. Some details of the socket layer and the protocol layer are omitted here.

The function `ax_dispatch_getnext()` in the protocol layer dispatches every SearchRange contained in the GetNext request. It then forwards the request for every single SearchRange to `ax_dispatch_sr()` in the session layer. This function now iterates over all callback functions for the session and compares their registration OIDs to the starting and ending OID in the given SearchRange. Every matching callback function is invoked until a callback returns a valid result.

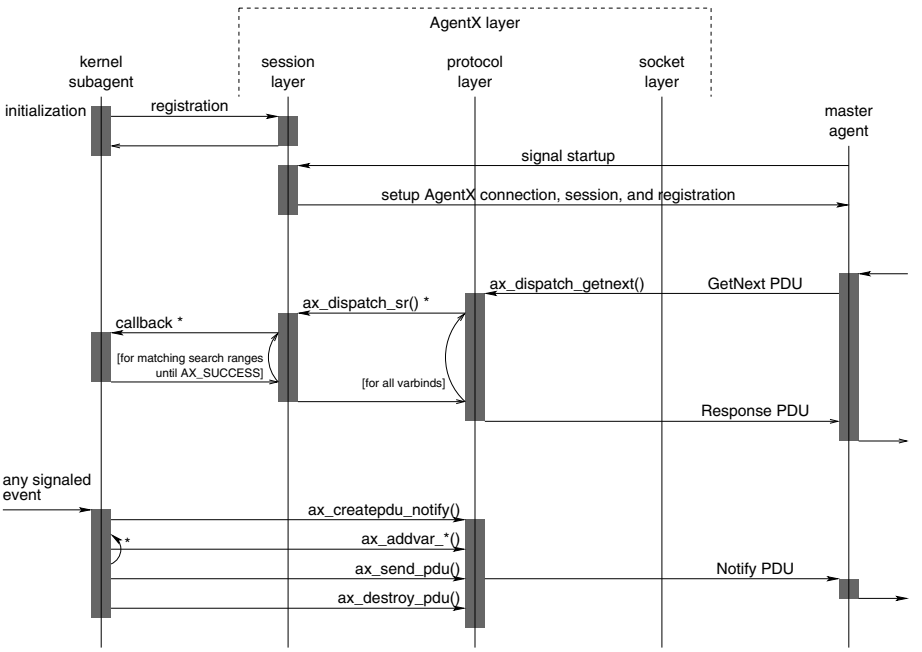



**Fig. 3.** Sequence diagrams: (a) startup procedure, (b) processing a GetNext request, (c) emitting a notification

### 3.4 Processing Set Transactions

The AgentX protocol accomplishes a successful Set transaction in three phases (see Figure 1). This is done to preserve the atomic nature of an SNMP Set request that may span multiple objects even at multiple subagents. However for the AgentX layer, Set transactions are very similar to Get requests, because varbinds are dispatched in the same way as SearchRanges. In case of success, a Set transaction consists of three request PDUs: a TestSet PDU to prepare the write access, a CommitSet PDU to actually write the data and a CleanupSet PDU to complete the transaction. The next example describes how set transactions are safely dispatched to the kernel subagents.

An AgentX TestSet PDU initiates a Set transaction. Every varbind from the TestSet PDU is dispatched to callback functions as explained before for SearchRanges in Get requests. Please notice that varbinds from the TestSet PDU may be dispatched to different callback functions. The callback functions check the varbinds for valid write access, type correctness, and legal values. When the checks have been succeeded, the AgentX layer saves all varbinds from the TestSet PDU because they are needed later. In the second phase, the master agent sends a CommitSet PDU to actually trigger the data change. Because the CommitSet PDU does not contain any varbind data, the AgentX layer refers to the previously saved varbind list to dispatch the CommitSet PDU to all corresponding callback functions. Finally in the third phase, the master agent sends a CleanupSet PDU. Similar to the commit phase, it is dispatched to the callback functions based on the saved varbind list to release any remaining temporary data of the transaction.

### 3.5 Sending Notifications

While Get and Set requests are handled by callback functions, notifications are initiated by kernel subagents. Hence, the AgentX protocol layer provides functions to kernel subagents to create a Notify PDU, to add an arbitrary number of varbinds of specific base types to the PDU, and to send the PDU to the master agent. Finally the kernel subagent has to release the PDU data from the protocol layer. This procedure is illustrated in the lower part of Figure 3.

## 4 Implemented MIB Modules

In order to evaluate the feasibility of the presented kernel subagent architecture it was implemented for the open-source Linux operating system. In addition to the AgentX layer, two MIBs were partially implemented: The `ifTable` and `ifXTable` of the Interfaces Group MIB [6] and a newly defined MIB [9] for the Linux Netfilter subsystem [5, 10].

### 4.1 The Interfaces Group MIB

The Interfaces Group MIB (IF-MIB) defines objects for managing network interfaces. Our implementation accesses existing kernel data structures directly or

through functions already provided by the Linux networking code. It can notice changes of network interfaces through an already existing notification hook, which makes the design of a separated subagent module (see the left part of the grey box in Figure 2) feasible. Hence, the kernel IF-MIB subagent was implemented as a Linux kernel module.

However, there are two objects for which no equivalent variables exist in the kernel: a timestamp for the last status change of an interface (`ifLastChange`) and the value of `ifLinkUpDownTrapEnable`. The latter is to specify whether to send a notification if an interface changes its status. Because a kernel module cannot extend existing data structures, the IF-MIB module introduces an interface shadow list where these values are stored. The elements of this list are created on demand so that the list contains only interfaces which have non-default values on any of these two objects. In addition to readable objects, this module implements the `linkUp` and `linkDown` notifications and write access to the `ifLinkUpDownTrapEnable` and `ifAdminStatus` objects.

Figure 4 gives an example of a callback function. It covers the table row `ifMtu`, the Maximum Transfer Unit. The IF-MIB defines this as a read-only object, hence this function returns an error on Set transactions. The function `get_ifid()` finds the correct interface ID for the requested OID. The function `getmtu()` is called to retrieve the MTU of the interface. Finally the full instance

```
int if_mtu(ax_oid *oid,
        ax_method method,
        char* context,
        ax_variable *res)
{
        u_int32_t ifid;
        const ax_oid IFMTU =
                {10, {1, 3, 6, 1, 2, 1, 2, 2, 1, 4}};

        if (!(m & GET))
                return AX_NOTWRITABLE;

        ifid = get_ifid(oid, method, 4);
        if (!ifid)
                return AX_NOSUCHOBJECT;

        res->oid                = ax_oid_addint(IFMTU, ifid);
        res->type               = AX_VB_INT;
        res->value.integer      = getmtu(ifid);

        return AX_SUCCESS;
}
```

**Fig. 4.** IF-MIB: Callback function for `ifMtu`

OID, the type and the value are stored to the result structure, before the callback function is terminated successfully.

### 4.2 The Netfilter MIB

The second implemented MIB is the experimental TUBS IBR Linux Netfilter MIB [9]. Netfilter is the Linux subsystem for packet filtering, mangling and network address translation (NAT).

The Linux Netfilter subsystem consists of so called tables. As of Linux 2.4.20 there are three tables in the Linux kernel: The packet filtering table, the network address translation table and a mangle table for packet alteration. Each table contains a number of built-in chains and may additionally have user-defined chains. The Netfilter subsystem currently has five hooks at five different points where an IP packet can traverse (INPUT, FORWARD, OUTPUT, PREROUTING and POSTROUTING). So each table has up to five different built-in chains. Every chain consists of a list of rules where each rule consists of one or more conditions (matches) and an action (target). If a packet matches all conditions, the according target is applied. Each built-in chain has a default policy which decides the fate of an IP packet that does not match any rule.

The Linux Netfilter subsystem is divided into several modules, which are responsible for different tasks. One module handles all table and chain management (`ip_tables.c`) so that the according network management functionality has been implemented there. This module defines data structures for Netfilter tables and chains but it, e.g., lacks methods to access a specific chain. This is done in user space by the Netfilter configuration tool *iptables(1)*. This approach is usually preferred because it keeps the kernel code small, but on the other hand, it makes the task of adding network management code to the kernel more difficult. So existing user space functions had to be rewritten and put into the kernel. Furthermore, the Netfilter MIB contains "LastChange" timestamp objects for all Netfilter elements, which the original Netfilter subsystem does not support. For this reason the data structures for Netfilter tables and chains had to be extended. This raised a problem, because as mentioned before, these data structures are used by the user space tool as well, so that adding timestamps to tables and chains breaks compatibility with the Netfilter user space tool. The solution to this problem is either to recompile the user space tool with the new structure definitions or to drop the "LastChange" objects. Finally, both approaches were implemented and the system administrator can decide at kernel compile time.

## 5 Evaluation

A brief evaluation was done in order to see if the presented kernel subagent architecture has measurable impact on the performance in contrast to a monolithic SNMP agent. For this purpose we used a Pentium 200MMX host with 64 MB memory running either a standalone NET-SNMP agent or the presented kernel

| | kernel subagent | | monolithic agent | |
|---|---|---|---|---|
| | mean | std.dev. | mean | std.dev. |
| snmpwalk ifTable | 323 ms | 11 ms | 351 ms | 7 ms |
| 10× snmpget ifNumber | 737 ms | 20 ms | 739 ms | 34 ms |

**Fig. 5.** Performance comparison

AgentX prototype with a NET-SNMP daemon as the AgentX master agent. All SNMP requests were issued by a remote manager over a local area network.

Figure 5 shows the results of the evaluation which presents no distinctive difference in performance. However, the implementation of `ifTable` is not equivalent in both approaches: the NET-SNMP implementation of `ifTable` supports 17 columnar objects while the kernel subagents supports 20 columns. Therefore in the snmpwalk test the kernel implementation returned 81 object instances for four table rows compared to 69 in case of the NET-SNMP implementation.

The implementation costs for kernel subagents turned out to be relatively small. E.g., the `ifTable` and `ifXTable` implementation presented in Section 4.1 comprise of about 900 lines of C code.

## 6 Conclusion

This paper states that the concept of distributed SNMP agent implementation by means of the AgentX subagent architecture is well applicable not only to modular devices and host services in user space, but also to kernel space subsystems. It has been argued that this allows MIB implementors to retrieve and manipulate data that lives in kernel space without the indirection of a potentially changing and limited kernel interface. Furthermore, this way it is easier to handle events in the kernel space in order to emit notifications for which user space agent implementations would often need to poll data frequently from the kernel due to the lack of appropriate trigger mechanisms.

To evaluate the concept of kernel AgentX subagents, two MIBs have been implemented for the Linux 2.4.x kernel: the IETF IF-MIB and a newly designed MIB module for the Linux packet filtering subsystem Netfilter. It has been shown that the effort to instrument typical kernel subsystems with subagent functionality is low and that the intervention to the existing kernel code could be restricted to a small interface. Furthermore it has been proved that the performance is at least comparable to traditional agent implementations.

The downside of implementing management agent functionality inside the kernel is the increased size of static kernel code and the general fact that kernel level code development is a delicate task, because bugs affect system stability more severely than user space programs.

The major benefit of the presented approach is the fact that the development of a kernel subsystem and its management instrumentation can be closely integrated. The development of a MIB implementation is put in the hands of

the maintainer of the subsystem which is to be managed through the MIB. This expertise helps to ensure more accurate MIB implementations and eases to keep track of changes in a subsystem which affect the management portion. At the same time the AgentX layer releases the developer from the necessity to know SNMP in detail.

## References


[1] The NET-SNMP home page. WWW Page. http://www.net-snmp.org 236

[2] G. Carpenter and B. Wijnen. SNMP-DPI: Simple Network Management Protocol Distributed Program Interface. RFC 1228, T. J. Watson Research Center, IBM Corp., May 1991. 236

[3] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction to Version 3 of the Internet-standard Network Management Framework. RFC 2570, SNMP Research, TIS Labs at Network Associates, Ericsson, Cisco Systems, April 1999. 234

[4] M. Daniele, B. Wijnen, M. Ellison, and D. Francisco. Agent Extensibility (AgentX) Protocol Version 1. RFC 2741, Digital Equipment Corporation, IBM T. J. Watson Research, Ellison Software Consulting, Cisco Systems, January 2000. 235

[5] Pat Eyler. *Networking Linux: A Practical Guide to TCP/IP*. New Riders Professional Library, 2001. 241

[6] K. McCloghrie and F. Kastenholz. The Interfaces Group MIB. RFC 2863, Cisco Systems, Argon Networks, June 2000. 241

[7] M. Rose. SNMP MUX Protocol and MIB. RFC 1227, Performance Systems International, May 1991. 236

[8] F. Strauß, J. Schönwälder, and S. Mertens. JAX - A Java AgentX Subagent Toolkit. In *Proc. 1st IEEE Workshop on IP-oriented Operations & Management*, Cracow, September 2000. 236

[9] F. Strauß and O. Wellnitz. The experimental TUBS-IBR Linux Netfilter MIB. `http://www.ibr.cs.tu-bs.de/arbeiten/strauss/kagentxd/TUBS-IBR-LINUX-NETFILTER-MIB`, 2002. 241, 243

[10] K. Wehrle, F. Pählke, H. Ritter, D. Müller, and M. Bechler. *Linux Network Architecture*. Prentice Hall, 2004. 241

[11] B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, and G. Waters. Simple Network Management Protocol Distributed Protocol Interface Version 2.0. RFC 1592, IBM T. J. Watson Research Center, Bell Northern Research Ltd., March 1994. 236

# Management Challenges of Context-Aware Services in Ubiquitous Environments

Heinz-Gerd Hegering, Axel Küpper,
Claudia Linnhoff-Popien, and Helmut Reiser

Munich Network Management Team, University of Munich, Dept. of Informatics
Oettingenstr. 67, D-80538 Munich, Germany
{hegering,kuepper,linnhoff,reiser}@informatik.uni-muenchen.de
http://wwwmnmteam.informatik.uni-muenchen.de

**Abstract.** Ubiquitous environments facilitate the collection of information pieces from sensors, databases, or mobile devices in order to compose the context of entities like users, places, or things. The context obtained in this way can be used to automatically adapt the behavior of services, which results in the new paradigm of context-aware services (CASs). In recent years, a lot of research has covered the functional aspects of CASs. However, CASs in ubiquitous environments impose new management challenges, which has not been considered so far. The goal of this paper is to identify new challenges on CAS management and thus to provide a roadmap for further research in this area.

**Keywords:** Context Aware Service, Quality of Context, Federative Organization Model, Context Information Model, Context Value Chain

## 1 Introduction

Mark Weiser's vision of *ubiquitous computing* of the year 1991 [18] gains more and more momentum as miniaturization and integration of computing and wireless communication facilities evolve. The Internet has been the enabling infrastructure behind services for individualized information retrieval and new forms of interactivity during the last decade. Now ubiquitous computing is the driving force behind the new service paradigm *context-awareness*.

According to Dey [5] "*context* is any information that can be used to characterize the situation of an *entity*: an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves". A service then becomes a *context-aware service* (CAS) if its behavior or the content it processes is adapted to the context of one or several entities in a transparent way. This adaptation process will be called *contextualization* in the following. However, before a CAS can be contextualized the context of relevant entities needs first to be obtained from various context sources like sensors or databases, which is called *context procurement*. In recent years, several platforms have been developed for realizing context procurement and contextualization, for example Parctab [14], CAPEUS [13], the

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 246–259, 2003.

ContextToolkit [5], and the Technology for Enabling Awareness (TEA) [15], to name only a few of them.

Many research projects have concentrated on functionality aspects of CAS and mostly regard device-centric applications. However, ubiquitous computing implies the interconnection of many local environments and devices and thus makes it possible that users share their context (and that of other entities) among each other. As a consequence, several actors like users, network operators, service and content providers are involved in context procurement, contextualization, and CAS provisioning. Because ubiquitous computing comes along with highly heterogeneous and distributed environments, our idea is that of an integrated approach of CAS management. Real-time requirements, high dynamic and automation are typical for CASs and evolve new management challenges. But management issues have been neglected or even left unconsidered in former platforms for context-awareness. Therefore, this paper identifies new challenges on a CAS management and thus to provide a contribution for further work in this area.

The remainder of this paper is structured as follows: section 2 introduces an application scenario that outlines the complexity of CASs and serves as a reference example for subsequent sections.To identify management challenges and structure the complex processes of context procurement and contextualization we adopt traditional management concepts[8]. Accordingly, we propose information and organization models dedicated to the special aspects of context awareness and, based on these models, derive the challenges within the functional areas of management. Section 3 covers the information model and introduces a process-oriented value chain to describe context procurement and contextualization in a structured way. The necessity of extending information models with the value chain and context description will be shown. Section 4 covers the organizational model and proposes a role model that identifies the different roles the actors of a CAS infrastructure may adopt according to their functional tasks. This role model is used to identify manager relationships across organizational boundaries. Based on the value chain and the role model, section 5 derives management challenges according to the known functional model, i.e., from the point of view of fault, configuration, accounting, performance, and security management. Finally, section 6 concludes the paper and presents further work.

## 2 Application Scenario

To outline the complexity of CASs in ubiquitous environments, but also to provide a reference example for discussing management challenges, we introduce the application scenario *Medical Advice and Emergency System* (MAES). MAES is intended for persons with critical diseases or medical disabilities (*patients*). The system gives medical advice to these patients and, in case of emergency, supports the workflow of the rescue crew. Different users of MAES, like patients, physicians, or the ambulance staff, use either conventional mobile consumer devices or special-purpose devices. They are connected to the system via heterogeneous

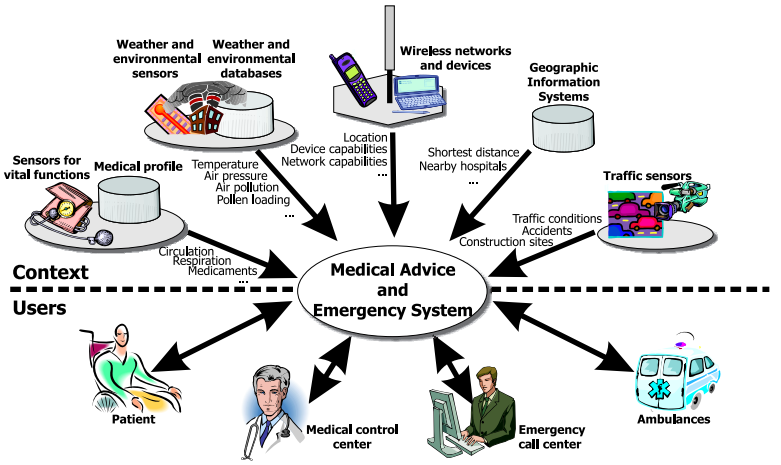



**Fig. 1.** Users and context information in MAES

wireless and wired infrastructures. Thus, from this point of view MAES can be seen as a conventional IT service which is implemented by composing interdependent and layered sub-services of different actors like network operators, service and content providers. However, usage of medical IT services like MAES impose very high requirements such as promptness, simplicity, and reliability on the underlying service infrastructure. It is therefore obvious to make these services context–aware in order to reduce awkward interactions between the users and the system, to automate workflows, and to adapt these workflows as well as the service's behavior and appearance towards the users according to current circumstances. To achieve this goal, MAES incorporates an extensive set of heterogeneous context information which cover medical, spatial, environmental, and technical aspects. Figure 1 gives an overview of the different types of users and the context information processed by MAES. An important input for MAES is the patient's medical situation, which comprises her current vital functions and her medical profile. Vital functions are derived by sensors which are worn by patients [9] and which deliver information about consciousness, respiration, circulation, and metabolism. The medical profile contains diagnostic findings made by physicians as well as prescribed medicaments and therapies. Depending on the current patient's vital functions and under consideration of her medical profile MAES may trigger the medical advice function or the emergency handling function.

The medical advice function notifies the patient or a remote medical control center about the current medical state and proposes remedial actions, for example to take certain medicaments or to consult a physician. These recommendations are combined with a list of nearby pharmacies and physicians, thus it is necessary to derive the current location of the patient. Weather conditions like temperature, precipitation, and atmospheric pressure, as well as environmen-

tal influences like air pollution, pollen loading, and ozone concentration may be considered in addition. Based on this environmental information, MAES can recommend to avoid exhausting activities.

Emergency handling is a workflow starting with the receipt of an emergency call and terminating with hospitalizing the patient. Upon receipt of an emergency call, the operator receives the patient's vital functions and medical history and can select an appropriate hospital with free capacities and in close proximity to the patient. Afterwards, MAES notifies an ambulance which is equipped with the required personal and material configuration (derived from the patient's medical profile) and which can reach the patient as fast as possible. The latter requires to derive the shortest path between ambulance and patient and to consider current traffic conditions along that path. The ambulance is supplied with the patient's medical profile and, if the patient has not been reached yet, with her latest vital functions.

From this application scenario several features of CASs can be derived that impose high requirements on management tasks. The following list itemizes the most important of these features:

- CASs obtain data from a very heterogenous set of *context sources*, e.g., sensors, mobile devices, and databases, which are highly distributed in the ubiquitous environment and which are located in the domains of different actors like operators of sensor networks or content providers.
- Data delivered by different context sources is consequently also very heterogeneous and usually varies in its update frequency (if any), its accuracy, and its format of representation.
- Context information is not only the output of some context sources, but the result of service and user specific context procurement, i.e., the processing and distribution of data delivered by context sources.
- CASs are characterized by a higher level of mobility. Not only users are mobile but even parts of the technical infrastructure, e.g., context sources. A CAS should even work if its mobile users leave the service area of her original provider or the coverage area of a certain context source. There are roaming challenges not only for users and services, but also for context information delivered by different actors.
- A context information may have a spatial and temporal validity. Due to the mobility of users, it might be necessary to locate relevant context sources during service usage and to identify resources needed for context procurement in an ad-hoc manner.

Provisioning of CASs will therefore be a new challenge for involved actors and their management systems. CASs will only be accepted at the market if they are easy to use, easy to configure and if the organizational and management boundaries are absolutely transparent from the customers' point of view. This means that realization and management of CASs require a completely new kind of interworking and cooperation between independent actors.

## 3 Context Information Model

Context-awareness implies the procurement of context and the contextualization of CAS components according to this context. Recent approaches have been dealing with these issues, for example, [5], [15], and [14], slightly differing from each other in the proposed structuring and terminology of these processes.

To automate the context procurement and contextualization we introduce the concept of a *value chain* which covers all steps happening between sensors and CAS components. Classical information models must be extended with a formal description of context and they must represent the value chain to make context procurement and contextualization manageable. We call these extensions context information model, which will be presented in the following.

Figure 2 shows an example of such a value chain which describes the procurement of medical profiles, vital functions, location and weather information as required for the contextualization of the medical advice function of MAES. Clearly, the universal value chain approach applies to other CAS scenarios, too.

The context of an entity may be derived from very different sources, e.g., sensors checking the patient's vital functions, positioning methods like GPS for obtaining her current location, or databases containing her medical profile. Capturing data from these context sources is a process we call *sensing*. However, sensing merely provides the "raw material" of context, which is referred to as *low-level context information* in [5], and which is often not interpretable by the requesting CAS. Usually, one or several steps of *refinement* have to be performed in order to derive *high-level context information* as required by the respective CAS. Among other things, refinement comprises the transformation between different formats of representation (e.g., from GPS coordinates to street names and numbers), the extension of context information with attributes (e.g., to express its accuracy), or the combination of context information to derive another one (e.g., calculating the distance between patient and ambulance by using their locations). After refinement, all the required context information which is related to a particular entity and which is of relevance for a particular CAS needs to be allocated, a process which is called *aggregation*. Figure 2 shows possible context information of a patient required for the medical advice function. Finally, the aggregated context is used to contextualize the CAS, which represents the last step of the value chain.

It must be stressed that a value chain may comprise several sequences of sensing, refinement, aggregation, and contextualization, one for each context information to be considered. These sequences might be executed in a prescribed order or in parallel, either entirely independent from each other or with need for synchronization. The exact coordination of the various sequences depends on a lot of circumstances like the requirements of the CAS, the range of context information to be processed, interdependencies between context information, the availability of context sources, etc. Each step of a sequence may be executed in an iterative manner and may comprise several sub-steps. Furthermore, the steps of a sequence may be triggered on demand or they may be event-triggered, i.e., if the value of a context information exceeds a pre-defined threshold. Thus, the

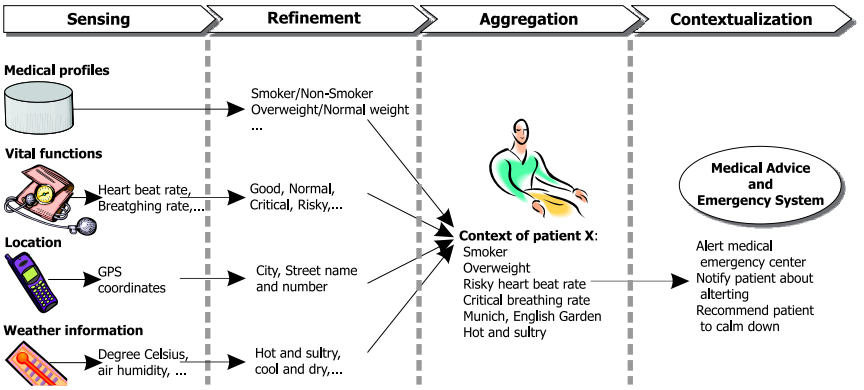



**Fig. 2.** The Context Value Chain

description given here is, due to simplicity, rather a coarse-grained representation of the value-chain concept.

Building a context chain and selecting the most appropriate context sources is a federative process. Automation of contextualization and using context information across organizational boundaries can only be done if all participants have the same notion of how to interpret the data. Therefore the management information model has to be extended with a *Context Description Language (CDL)* to formalize the specification of context. Without such a formal and automatically evaluable description of provided context information selection of context sources and brokerage of such information is infeasible. A starting strategy on the way to a uniform notion of context is to build context categories, i.e., *device-specific*, *environment-specific* and *user-specific context* [2]. However existing models within these categories are focused mostly on the considered application scenario, e.g., [19] concerns human computer interaction and models context within a hierarchical model which is optimized for database storage with quite few context-classes. A quite narrow and static model for device specific context for resource detection in ad-hoc networks is presented in [12]. An all-embracing model for all context categories is still missing, but nevertheless needed. Therefore a comprehensive CDL had to be specified and information models like the Common Information Model (CIM) should be enhanced with it. Even if an unambiguous context description exists, it will be not fully sufficient for selecting context sources.A formal notion of *Quality of Context (QoC)* as an integral part of each context description is needed to build an appropriate value chain in terms of quality and price of their aggregated context. Examples for such QoC parameters are accuracy, availability, timeliness, validity period, system of units, conversion factors, etc. [3].

As there are high timing constraints in CAS the process of context procurement might not contain any procedure which requires human interaction. Automation can only be achieved if all sub-processes of the value chain can be

automated. One of our ideas to investigate in the future will be the modeling of each sub-process of the value chain as a managed object. The management system has to regard the formal context description with actual QoC as input parameters to select the appropriate sub-process(es) with its "best" context source.

## 4 Federative Organization Model

A nontrivial CAS can only be realized in an interorganizational manner. As seen in section 2 the MAES provider can only offer its service by composing sub-services of various other actors (e.g., operators of sensor networks, mobile network operators, information system providers, traffic management systems, ambulances, hospitals, etc.). Thus, several actors must establish a federation in order to establish the value chain presented in the last section by composing their sub-services. To classify the actors according to their roles they play in such a federation, and thus to structure management tasks from an organizational point of view, we propose a role model for CAS.

In this model, an *actor* denotes an individual, organization, department, or enterprise, which offers services to other actors, which consumes services from other actors, or which does both of them. From a system's point of view, each actor autonomously operates and controls its own administrative technical domain, consisting for example, of a network infrastructure, a server farm, or only a single mobile device. An actor may adopt one or several roles. A *role* represents a certain field of activity of an actor and is associated with a certain set of sub-service components for realizing and controlling the value chain. Figure 3 shows our approach of a role model for a CAS infrastructure. Note that our understanding of a role model is in accordance with that of business models that have been proposed for telecommunication systems like TINA [17], UMTS [1], or the MNM service model [7]. However, we prefer to define a dedicated role model in order to be independent of a particular network technology and to highlight the special problems and tasks of context-awareness.

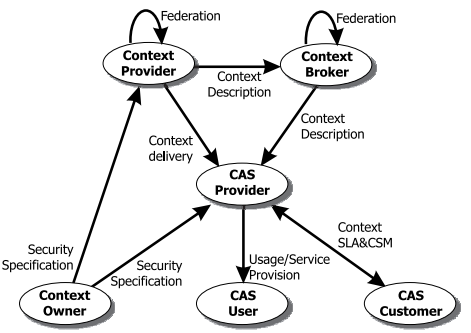



**Fig. 3.** The Role Model of a Context-Aware System

The central role of our model is the *CAS provider*, which creates and deploys CASs like MAES and offers and sells them to a *CAS customer*. The CAS customer interacts with the CAS provider in order to negotiate service level agreements and to control customer service management on behalf of one or several *CAS users*. The CAS provider obtains context information for contextu-

alization of its services from a *context provider*, which is usually the operator of context sources. For example, a context provider may be the operator of a cellular network, which tracks a CAS user and delivers that user's current location to the CAS provider. In some cases more rules can be adopted by the same actor, e.g., the roles of context provider and CAS user are adopted by the same actor. For example, a patient uses the MAES and simultaneously delivers her vital functions to the system.

Due to the heterogeneity and diversity of context information, it is unlikely that a certain context provider is able to deliver all context information required for a CAS. Also, a context provider may only cover a limited geographic area, and due to the mobility of CAS users the relevant context providers are not known in advance, but must be identified during CAS usage. The *context broker* supports this identification. It maintains a directory for registering context descriptions of information a context provider is able to deliver together with its QoC. This directory can then be requested by CAS providers in order to find appropriate context providers.

For many services it would be desirable that a CAS user has access to context information which is related to another actor. For example, in the MAES scenario the emergency control center requires the patient's vital functions and medical profile in order to decide about the right ambulance configuration. From a security point of view, this a very sensitive matter, because an actor must always have control about the processing and accessing of her context by other actors. It is therefore inevitable to establish the role of a *context owner*, which represents an entity context is related to and which is able to specify access restrictions regarding its context.

The value chain presented in the previous section is realized by the interactions between different context providers on the one hand as well as between context providers and a CAS provider on the other. The exact mapping of sensing, refinement, aggregation, and contextualization onto the roles is a point for further discussion. However, it is obvious at least that context providers always performs the sensing, whereas CAS providers are responsible for the contextualization. Whereas the value chain is directly realized by the roles of context and CAS providers, the remaining roles have nevertheless a significant impact on the value chain, which is to be discussed within the scope of the management functional areas in section 5.

As can be derived from this role model, each actor will have its own technical infrastructure and management system(s), and there is no "leader" which can intervene regulatively. Enterprise management concepts with the aim to build an integrative management umbrella on top of various different management systems is infeasible, because in such a federative environment there will be no actor prominent enough to establish and operate such an umbrella system. Most of the participating actors will not allow deep influence within their own sense-making process. Therefore, managing CAS will only work in a federative and not in any kind of hierarchical organization model, i.e., operation and management of a CAS infrastructure will only be achieved in an interorganizational cooperation

between the various actors. The managers of different actors need to interact among each other in order to establish and control the value chain according to the management functional areas. However, in such a federation each actor may deploy its own management system, including actor-specific representations of resources and sub-services (managed objects) as well as management protocols. Thus, management gateways are required at the boundaries of each actor's technical domain in order to mediate between the heterogeneous management systems.

## 5 Management Functional Areas

In contrast to conventional, non-context-aware IT services the realization of management functions including the inter-organizational mediation process is a hard and troublesome matter. The mobility of users in combination with the availability of appropriate context sources forces a high degree of flexibility and automation on management functions during CAS provisioning. In addition, these management functions may be subject to hard real-time requirements. The impact of these requirements is discussed in the subsequent sections for each of the functional management areas separately.

**Configuration** The concept of context value chains raises the question how to establish them if a certain context information is needed. Obviously, its configuration depends on the context information itself, whether or not it is based on other context information, and the requirements of the CAS. Whereas there are some static aspects, configuration is frequently based on dynamic criteria, above all the mobility of users. For example, the availability of context sources depends on the user's location, i.e., if the user moves, the set of available context sources may change accordingly. Furthermore, a context information can usually be derived by different value chains, differing from each other. Establishing the most suitable value chain for each individual user — taking into account her current situation — will be the main configuration management challenge for the CAS provider. For this purpose the CAS provider needs an integrated configuration mechanism across heterogeneous systems and across various context providers. This means that each context provider has to make available an automated configuration interface for its CAS providers. Beneath building the value chain its automatic adaptation to a constantly changing infrastructure, e.g., if sensors vanish, will be another challenge. The CAS provider itself must support its customers with a customer service management (CSM) interface [10]. A CAS must be easy to use and easy to configure. The configuration simplicity will be the most important aspect for CSM.

The service lifecycle as presented in [7] is feasible for establishing a CAS infrastructure in the long term. However it does not cope with the ad-hoc creation of an user individual peculiar instance of a certain CAS regarding the current context of the context owner. Therefore during the usage phase of each CAS a

micro-lifecycle has to be established and traversed for building each individualized service instance. This micro-lifecycle contains all phases of the long term life cycle, i.e., design (resp., adaptation), negotiation, provisioning, usage and de-installation (resp., de-allocation). Constitution and cycling the micro life cycle must be supported by configuration management functions.

**Fault** The challenges in fault management cover two areas: faults in sensing and faults in the procurement of context. Context sensors are often proprietary highly specialized devices with their own raw data format, limited usage interfaces, limited CPU as well as limited memory, and without management or alarming functions. Fault detection and recovery in sensor networks therefore is not trivial. Management mechanisms have to cope with e.g., power managment, radio energy management, sensor location, (collaborative) signal processing, sensor synchronization, etc. [6]. Context sensing within CAS is characterized by highly dynamic and short-term update cycles of context information. Especially, for sensor networks self-healing and self management mechanisms are needed to be able to cope with dynamics and the technical deficiencies of sensor networks — beneath their pure sensing functionality — must be compensated by the management system.

Sensing — capturing data from context sensors — does not always work; this is a technique intrinsic fact within context sensing. A simple example for that is the impossibility of GPS localization without a direct line of sight to some satellites. For localization in CAS this means that a GPS sensor is useless indoor. The effect of such technical restrictions for the process of building a value chain is the same as loosing a certain sensor. The fault management system must be able to detect this (temporary) absence of sensors and must support the configuration management as well as activate the micro–lifecycle to build an alternative value chain with nearly the same QoC.

**Accounting** The accounting management of CAS must be able to deal with the fact that each user gets its own and unique service which will be individually customized for her. In the worst case such a service will only be used once. Developing of fair tariffs for a user individual CAS which will be used only a few times is an unsolved problem. From the customer's point of view the predictability of tariffs and keeping the costs of CAS under control is mandatory.

The accountable units, the mechanisms, the measurement points for gaining them and their exchange within an interorganizational CAS infrastructure have to be defined. Both sides — customer and CAS provider — need a concept for conservation of evidence for the CAS usage and accordant accountable units. The CAS provider needs new cost and charging models for context information and for CASs. These models must then be deployed between different (sub-)providers which contribute information to the context value chain.

Concepts of calculating costs, negotiating them with the customer, and concluding them in an agreement before actually using the CAS will not work in such highly dynamic environments. For CAS ad-hoc agreements and ad-hoc pricing

are necessary. It can not be assumed that each customer will apply and subscribe before actually using a service. Rather, a customer would like to define a price ceiling for a certain CAS and awaits best possible service delivery within her price range. From management point of view this connotes a complete inversion: not the service defines the price but the price causes the type of service provisioning. QoS based pricing concepts can be a starting point (for a review cf., [16]), however concepts for a price-driven service adaptation, building the value chain and price-driven service-provisioning are fundamentally needed.

Besides the classical roaming in cellular phone networks CAS users will claim for "service roaming" which enables a seamless CAS usage even if the user leaves the service area of its CAS provider. For that purpose and to follow the principle "one face to the customer" an inter-provider accounting system is necessary.

**Performance** A CAS "performs well" if its user will always get appropriate QoS. A CAS provider must be able to build and even update value chains with appropriate QoC. In provisioning of CAS, in procurement and update of context information real time requirements can not be eliminated. For building a certain context value chain this implies concepts for optimizing this process in terms of response times. Therefore notions for optimal context caching and context reuse can be helpful. Reuse of sensor data, context or of a preliminary stage within a value chain, can shorten allocation process. A basic QoC–attribute therefore will be the validity period of each context information. Without this caching is not feasible. Because of the frequently changing circumstances during CAS usage and as changes in the actual environment of a context owner might change its context a predefinition of a validity period is not that simple.

**Security** Providing context aware services requires foremost the same security mechanisms — e.g. for identification, authentication, and authorization of participating authorities, non repudiation, etc. — as traditional services. However there are some new aspects respectively tighten requirements especially regarding confidentiality, integrity, availability (CIA), informational self-determination, privacy, and trust level management.

Context information and according profiles might be highly sensitive and private information for each user; e.g., medical profile or current medical status in the MAES scenario. It may be disastrous for the context owner if even parts of this information came to wrong hands, would be manipulated, or if the context sources the service depends on would be unavailable. Existing confidentiality mechanisms do not scale well for CAS, because protect-able data might be generated by a huge amount of context sources under responsibility of lots of different context providers and transmitted via various communication mechanisms. Classical confidentiality mechanisms cope with a manageable group of beforehand known communication partners or they are designed for certain communication channels (e.g. IPSec, ssh, WEP). In CAS none of this prerequisites are entirely fulfilled. There are contractual relations between the context owner and some CAS providers, but normally there is no such contract between her

and the various context providers. Even worse, the context owner often does not even know context providers which generates information about her. The information flow is unknown and uncontrollable for her. Data is transmitted via heterogeneous channels (BAN, WLAN, 3G Networks, TCP/IP Networks, etc.). Therefore integrated and interorganizational security mechanisms are which can cope with and configure different heterogeneous security mechanisms on different and heterogeneous infrastructures in a consistent manner.

Similar problems arise regarding integrity, accuracy, and availability. The CAS user must be absolutely sure that context information about a context owner delivered by herself, by the CAS provider, or even directly by the context providers are reliable, accurate, and on time. For proper CAS operation not only the availability of servers or network components are critical, but also the availability of context sources. However sensors are not designed to have high availability and it is easy to disturb them physically. Mechanisms for automatically selecting and querying alternative context sources with similar QoC are essential for the CAS provider in case of unavailability. All participating roles must be able to verify the integrity-aspects (reliability, accuracy, timeliness) of received data independently of other roles.

For the context owner the most important security requirements are her informational self-determination, her privacy, and controlling the hazards of personalized context information. Obtaining condensed context information might enable serious misuse. Potential dangers are user tracking and sophisticated user profiling. Concepts for the context owner are necessary to specify and perhaps even control information flows and to determine who can see or use her actual and historical context information. This means that the context owner must be enabled to authorize and prohibit access to data which is not under her own control. For privacy aspects within web-services a privacy policy solution has been presented in [4, 11]. Parts of these concepts might be applicable or enhanced for CAS. But specification of such access rules must be done easily and independently of arbitrary access control models used at the provider side. Protectable data is produced by a huge amount of different providers. Even worse for the context owner, there are unknown context providers which are known only by the CAS provider. In such cases, and in order to relief herself, the context owner might delegate the informational self-determination tasks to the CAS provider or to a security provider. Therefore and as a basic security service, a formal trust model and its implementation within a trust level management system could be helpful. Today, trust is mostly defined implicitly and in a "make-or-break" manner. If it is feasible to define, specify, and check the level of trust, then it will be calculable and automatically processable. For services which need the user context without personalized information a concept for anonymous service usage should be available to avoid user profiling.

## 6 Conclusions and Future Work

In this paper new challenges caused by CAS have been identified and presented. No doubt, more questions and problems are raised than can be answered at the moment. Some approaches to cope with this challenges have been presented. Within CAS scenarios with real time requirements, i.e., with short time and user-individual service adaption needs and with different organizational domains there is an urgent necessity for self-management concepts. In coping with sensor networks and their management deficiencies concepts for self–healing will be helpful.

This work is intended as a starting point for lots of research which must be done in the future. Our next steps in this area will be the observation of further CAS application scenarios to identify the functional building blocks for CAS–specific FCAPS. A deeper investigation of interorganisational aspects is planned regarding the interactions which are relevant for management purposes. Furthermore we are building managed object classes for the role model, the processes contributing to the value chain and for context information. We are working on the specification of a CDL and as QoC will be a prominent part we try to identify universal QoC parameters for the different context categories.

## Acknowledgement

The authors wish 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. For more information see
`http://wwwmnmteam.informatik.uni-muenchen.de`.

## References

[1] 3GPP Technical Specification Group Services and System Aspects. Service Aspects; Stage 1 Service Requirements for the Open Service Access (OSA) - Release 4. Technical report, 3rd Generation Partnership Project (3GPP), March 2002. 252

[2] N. Anerousis. Pervasive Computing. Tutorial at the 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003), March 2003. 251

[3] T. Buchholz, A. Küpper, and M. Schiffers. Quality of Context Information: What it is and why we need it. In *Proceedings of the 10th HP–OVUA Workshop*, volume 2003, Geneva, Switzerland, July 2003. 251

[4] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation REC-P3P-20020416, W3C, April 2002. 257

[5] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, November 2000. 246, 247, 250

[6] D. Estrin, A. Sayeed, and M. Srivastava. Wireless Sensor Networks. Tutorial at the Eighth ACM International Conference on Mobile Computing and Networking (MobiCom 2002), September 2002. 255

[7] M. Garschhammer, R. Hauck, H.-G. Hegering, B. Kempter, M. Langer, M. Nerb, I. Radisic, H. Rölle, and H. Schmidt. Towards generic Service Management Concepts — A Service Model Based Approach. In G. Pavlou, N. Anerousis, and A. Liotta, editors, *Proceedings of the 7th International IFIP/IEEE Symposium on Integrated Management (IM 2001)*, pages 719–732, Seattle, Washington, USA, May 2001. IFIP/IEEE, IEEE Publishing. 252, 254

[8] 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, January 1999. 651 p. 247

[9] C. Kasabach, C. Pacione, J. Stivoric, A. Teller, and D. Andre. Why the Upper Arm? Factors Contributing to the Design of an Accurate and Comfortable, Wearable Body Monitor. Technical report, BodyMedia, 2002. 248

[10] M. Langer, S. Loidl, and M. Nerb. Customer Service Management: A More Transparent View To Your Subscribed Services. In A. S. Sethi, editor, *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, DE, USA, October 1998. 254

[11] Marc Langheinrich. A Privacy Awareness System for Ubiquitous Computing Environments. In G. Borriello and L. E. Holmquist, editors, *4th International Conference on Ubiquitous Computing (UbiComp2002)*, number 2498 in Lecture Notes in Computer Science (LNCS), pages 237–245. Springer, 2002. 257

[12] G.-C. Roman, C. Julien, and A. L Murphy. A Declarative Approach to Agent-Centered Context Aware Computing in Ad Hoc Wireless Environments. In *The 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'2002)*, May 2002. 251

[13] M. Samulowitz, C. Michaehelles, and C. Linnhoff-Popien. Adaptive Interaction for Enabling Pervasive Computing Services. In *2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE 01)*, Santa Barbara, Califonia, USA, May 2001. ACM. 246

[14] B. N. Schilit. *System Architecture for Context-Aware Mobile Computing.* PhD thesis, Columbia University, New York, 1995. 246, 250

[15] A. Schmidt and K. van Laerhoven. How to Build Smart Appliances. *IEEE Personal Communication*, pages 66–71, August 2001. 247, 250

[16] B. Stiller, P. Reichl, and S. Leinen. A practical review of pricing and cost recovery for internet services. In *Netnomics – Economic Research and Electronic Networking*, volume 3. Baltzer, The Netherlands, March 2001. 256

[17] TINA Business Model and Reference Points - Version 4.0. Technical report, TINA Consortium, May 1997. 252

[18] Mark Weiser. The Computer of the 21st Century. *Scientific American*, September 1991. 246

[19] H. Wu, M. Siegel, and S. Ablay. Sensor Fusion for Context Understanding. In *Proceedings of IEEE Instrumentation and Measurement Technology Conference*, Anchorage, AK, USA, May 2002. 251

# Aggregation of Composite Location-Aware Services for Mobile Cellular Networks

Alvin Yew, Audun Strand, Antonio Liotta, and George Pavlou

Centre for Communication Systems Research, University of Surrey
Guildford, Surrey, GU2 7XH, United Kingdom
{K.Yew,A.Liotta,G.Pavlou}@eim.surrey.ac.uk
audun@audunstrand.com

**Abstract.** The introduction of context-aware services through service frameworks such as Open Service Access gateways in 3rd Generation networks has coincided with the increasing popularity of Business to Business (B2B) solutions such as Web Services. It is envisioned that B2B characteristics, such as service aggregation will play a part in deploying location-aware services in 3G networks. This paper examines and explores the suitability of service integration models, possible business models, the technical requirements, and suggests a framework for the aggregation and deployment of aggregated composite location-aware services. A prototype of the framework was developed and experiments involving J2EE based and Web Services/SOAP based composite services were conducted and elaborated in the paper. An analysis of the experimental results is presented at the end of the paper.

## 1 Introduction

The deployment of 3rd Generation (3G) mobile communication networks has provided a wealth of service provisioning possibilities for service providers through new service infrastructures, frameworks, and architectures. Advanced service concepts such as the Virtual Home Environment (VHE) can assist a user in circumventing the difficulties relating to accessibility of personalized services during roaming that were characteristic of past mobile networks generations. A new category of services which can adapt to a user's context, i.e. physical and service operation environment, has been steadily gaining prominence in research. Location-aware services, which can provide service functionality and content relevant to the user's location, belong to that (now often dubbed as *context-aware services*) category. The discovery of a user's context in 3G networks by context-aware services will be possible through Open Service Access (OSA) and Parlay gateways owned by network operators[1,2]. These gateways encapsulate context determining technologies in the core network such as the Location Service (LCS) for user location determination and other core network capabilities and functionalities[3]. Advances in Business to Business (B2B) service frameworks and middleware have also been progressing

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 260-272, 2003.

steadily over the years as previous state-of-the-art technologies, such as CORBA, have been superceded by more developer-friendly alternatives such as Java 2 Enterprise based architectures, and Web Services using the Simple Object Access Protocol (SOAP)[4]. It is foreseeable that B2B interactions will have some part in delivering location-aware services to 3G users in the future.

It is common in B2B interactions to combine different, possibly autonomous, services together to form a whole new service. In this paper, we refer to this new service as an *aggregated service*, the individual services that existed before the aggregated service as a *composite service*, and the process of combining the composite services as *service aggregation*.

The study of service composition techniques, requirements, and issues has been identified as a research priority by the World Wireless Research Forum (WWRF)[5]. There is a need to evaluate how service composition may affect the operation and performance of future 3G services such as location-aware services. For example, how do we aggregate composite location-aware services with different requirements (e.g. level of location accuracy)? New possible business models for such scenarios may also be needed and thus require research. This paper aims to tackle and answer the issues concerned with the aggregation of composite location-aware service as well as the characteristics and deployment of aggregated location-aware services.

The format of the paper is as follows: Section 2 introduces the four main models that can be used to integrate composite services. Section 3 analyzes the technical requirements associated with the aggregation of composite location-aware services, and aggregated location-based services operating over mobile cellular networks with a particular emphasis on 3G networks. Section 4 highlights our work on defining a reference framework for aggregating and deploying aggregated location-aware services. It also examines how future business models for 3G networks can have an impact the manner in which our framework is deployed. Section 5 presents an overview of the implemented prototype, details the experiments performed, and evaluates the results of the experiments. The paper concludes by examining the related work done in the area, reiterates the main contributions of our research, and presents potential future work to be done in the area.

## 2 Overview of Composite Service Integration Models

The development of models for integrating composite services is somewhat similar to that of grid computing. The aggregated service represents the combined service output of all composite services from the user's point of view and is usually in charge of initiating the invocation of the various composite services. Therefore, there are two types of message flows in an integration model; a control-flow and a data-flow. A control-flow is used to control the composite services and can include service initiation, service suspension, and service resumption. A data-flow represents the exchange of service-related and service-specific data, such as localized content, between entities in the model. Figure 1 shows the four main models that are available for integrating composite services[6]. We concentrate our explanations on the first two models as they allow the reader to easily comprehend the operation and purposes of the other two. The models in figure 1 are numbered in the order of their description

below. The four available models for the aggregated service to integrate composite services are:

1. Centralized Control-flow Centralized Data-flow – This is the predominant model that is currently documented in literature. The aggregated service is in charge of controlling each composite service, and receives data-flows directly from them as well. This model is used when it is required to have no dependencies between different composite services, and when the aggregated service requires total control in managing the composite services. A disadvantage of using this model is that there will be an obvious bottleneck at the aggregated service in terms of traffic and processing.
2. Centralized Control-flow Distributed Data-flow – This model tries to alleviate some of the bottleneck experienced by the previous model by allowing composite services to exchange data-flow messages between themselves. The model is particularly useful when the output data-flow of a composite service is required as the input data-flow of another composite service, e.g. the aggregated service performs sequential invocation of two or more composite services. This model allows the aggregated service to control a composite service to send its output data-flow directly to the next composite service, which then sends its output data-flow back to the aggregated service, saving both time and network traffic in the process.
3. Distributed Control-flow Centralized Data-flow – Distributed Control-flow models are particularly useful when composite service invocations do not need to be sequential. They are also useful when deploying autonomous mobile code such as mobile agents and in active networks. The usage of a centralized data flow in this particular model allows service data integrity as the service data will not be passed between different composite service provider competitors.
4. Distributed Control-flow Distributed Data-flow – This model inherits a combination of the characteristics inherent in models 2. and 3.

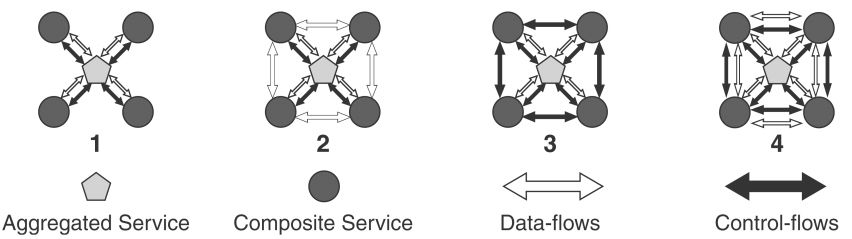



**Fig. 1.** Integration models for service aggregation

## 3 Requirements Analysis

### 3.1 Accommodating Different Types of Aggregated Location-Aware Services

There are three methods of aggregating composite location aware services. Any proposed framework must accommodate these three types of aggregated location-aware services, which are:

1. *Aggregating services of different service categories for a particular location* – CnL1 aggregated service. For example, an aggregated service is charged with finding a list of restaurants in the area with automated on-line reservation capabilities, a list of car parks in the area including the available capacity at the moment, and a traffic update on the roads in the area as well. This example combines three different independent service categories to provide an aggregated location-aware service.
2. *Aggregating services of the same service category over a range of locations* – C1Ln aggregated service. For example, a user may want to take a bus from his current location to another location but there is no single bus route or operator that covers this trip. He/She uses an aggregated service that determines his/her current location, contacts different bus operators' timetables and routes to coordinate an inter-change within a reasonable transit period, and displays to the user the location of the bus stops to board the bus at each transit point via a map. An additional benefit provided to the user by such a service is that a composite service's dynamic information and functionalities, such as bus route detours and cancellations, are incorporated into the aggregated service thus alleviating the user of such concerns. In this situation, the aggregated service may be seen as a single location-aware service whose locality is the union of each composite location-aware service's locality.
3. *Aggregating services of different service categories over a range of locations* – CnLn aggregated service. This type of aggregated service combines the characteristics of the previous two. The locality of a *CnLn* aggregated service is also the union of each composite location-aware service's locality.

### 3.2 Recovering from Limitations of the Overlaid Service Infrastructure

Location-aware services form a service infrastructure overlaid on top of a geographical location. They are, however, different from other service infrastructures because service locality is of primary importance to the functionality of the location-aware service. Figure 2 shows a fictitious service infrastructure available to an aggregated service provider superimposed on a map of London and its surrounding counties. The letters in figure 2 denote the various different service provider localities. Let us consider that this is a service network of bus operator services in the South-East of England, and that a user wants to travel from central London to Eastbourne for the sake of highlighting our requirements. The following observations can be derived from the figure:

- There is only one bus operator in central London.
- In between central London and the edge of the service network near Eastbourne, the number of competing bus operators in a single location can vary from one to greater than one.
- The service network does not extend to Eastbourne.

It is evident that the aggregated service cannot perform optimally and give the user complete instructions and information on the choice of bus routes from central London to Eastbourne. This is due to the fact that the service network does not extend to Eastbourne. More interestingly, there are two possible reasons for the lack of a

service network at Eastbourne. Firstly, there is no actual bus route to Eastbourne and therefore it is impossible to get to Eastbourne by bus. Secondly, a bus route does exist but the bus operator in that area does not offer its services to the aggregated service provider.

The two possible reasons highlighted in the previous paragraphs implicitly impose a requirement on the transactional nature of aggregating composite location-aware services. A non-transactional *best effort* type of aggregated location-aware service with suboptimal performance can be suitable for *CnL1* aggregated services. For the example in subsection 3.1 of a *CnL1* aggregated service, the user may not mind not knowing the available car parks in the area as long as he is provided the list of restaurants. For the bus operator service discussed earlier in this subsection, which is a *C1Ln* aggregated service, a transactional approach may be more suitable if it was impossible to get to Eastbourne by bus. However, a *best effort* type of service may be considered if the service was theoretically possible but undeliverable due to a lack of a service network and infrastructure. These possibilities and alternatives highlight the need for the transactional characteristic requirement to be considered on a case to case basis for each deployed aggregated service.

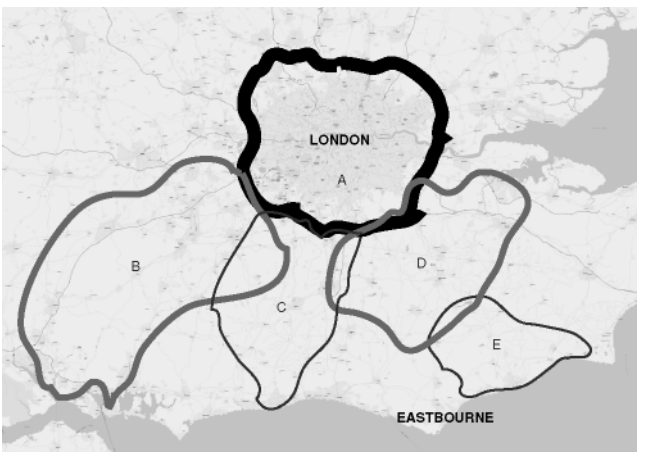



**Fig. 2.** A service network/infrastructure overlaid on geographical locality

### 3.3 Accuracy of the User's Location

There are various methods available to ascertain a user's location in current cellular networks. These methods vary widely in the accuracy of determining the location, and the environmental characteristics that are required for a specific positioning method and technology to be available and accurate. The latter usually affects the results of the former in most location-tracking technologies. For example, the Global Positioning System (GPS) requires the user to be mainly outdoors so as to receive a Line of Sight (LOS) signal from the Low Earth Orbit satellites and its accuracy increases as the number of satellites over the user's location increases. While many positioning methods and technologies do exist, e.g. Round-Trip Time, Angle of Arrival, Reference Node Based Positioning, 3GPP has standardized three methods for the Location Services (LCS) functionality in 3G mobile networks. They are Cell ID

based positioning, Observed Time Difference of Arrival (OTDOA) positioning (which is based on Time Difference of Arrival positioning), and Assisted GPS positioning (which essentially is GPS positioning). OTDOA works well indoors and provide reasonable accuracy when more than two neighboring base stations are used in determining the user's location. GPS is the most accurate but requires LOS between the user and three or more satellites to perform well. Cell ID based positioning is the least accurate and its accuracy is inversely proportionate to the size of the cell that the user is in.

It can be easily deduced that the accuracy of the user's location is a major factor when provisioning location-aware services. Inaccurate location positioning of the user can result in providing the user with services that he/she does not want or need, thus decreasing user satisfaction and service reputability. The importance of this factor increases when various location-aware services are aggregated together. Although all aggregated services would suffer to some extent from positioning inaccuracy, a *CnL1* aggregated service would be more likely to be affected than a *C1Ln* or a *CnLn* aggregated service as its overall service locality is smaller than that of the other two (assuming that the area of each composite service locality is equal and exclusive) – a small inaccuracy would constitute a greater percentage of error over a small area than a large area. However, this may not be the case for an aggregated service with functionalities that has strict positioning accuracy requirements regardless of aggregation type, e.g. a service involving the location of the nearest hospital or 24-hours emergency medical clinic. Therefore, the requirement for positioning accuracy from an aggregated service depends on both the type of aggregated service, and the aggregated service functionality.

# 4 Design of the Proposed Framework

## 4.1 Design Considerations and Features

Figure 3 gives a functional overview of our proposed framework for accommodating aggregated location-aware services using composite services. We stress that the proposed framework should not be seen as a ‘one solution fits all' approach for deploying such services as we focus only on issues and functionalities that are specific to location-aware services, and do not describe other features such as accounting, user authentication and session management in this paper.

A huge factor in the design of the framework is that it must fit within a sensible and realistic business model. During the course of our research leading to this paper, we visited a major UK network operator, as well as an independent investment banker, for consultations on future viable business models for deploying location-aware services in 3G mobile communication service frameworks. The framework design accounts for a key point that they raised during our meetings – information about a user's location is extremely private, and will not be released to any service provider that requests for it through the OSA interfaces. The access to such information through OSA is negotiated through off-line service level agreements, and they predicted that only a few privileged service providers will be considered trustworthy and reliable enough to gain such access. The network provider ultimately

has a responsibility to its customers to protect their privacy as well as its own reputability.

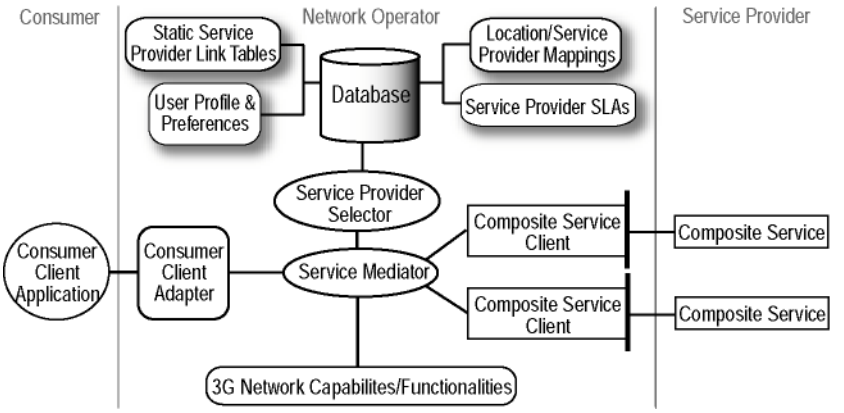



**Fig. 3.** High-level functional view of the framework

We do not include interactions with an OSA/Parlay gateway in figure 3, as our framework places the emphasis on the network operator to initiate and control the service aggregation process. We do, however, acknowledge the possibility that composite services may also be aggregated services themselves – i.e. they use other composite services to perform and operate their services. By placing the service aggregation process within the responsibilities of the network operator, it is also possible to conceal the location of a specific user from composite location-aware services. The latter would not require the user's identity for billing or authentication as the network operator would be charging the user on the composite service provider's behalf. A network operator controlling the service aggregation process also implies ease of control over the choice of positioning methods and technologies used in ascertaining the user's location, thus providing some flexibility in satisfying requirements on positioning accuracy from various aggregated location-dependent services. It is, however, still possible to deploy our framework within the domain of a value-added service provider (to a 3G network operator). This would mean interpreting the interface between the service mediator and the 3G network capabilities/functionalities functional components to be that of OSA's (refer to figure 3).

We also designed the framework to incorporate flexibility in as many aspects of service provisioning and management as possible, and therefore the following features are present in the framework:

a) The framework interoperates with a range of service deployment technologies available such as Enterprise Java Beans, Web Services/SOAP etc., allowing composite service developers flexibility in choosing service deployment middleware technologies.
b) The framework allows more than one service access point to the aggregated service so as provide flexibility in service delivery to the service consumer. E.g.

Java 2ME midlets with Java RMI clients, SOAP clients, HTTP-XHTML/HTML/WML.

c) The framework design allows service aggregation using any of the four service integration models presented in section 2.

### 4.2 High-Level Component Description

This subsection explains the functionalities of each major component in the framework as shown in figure 3.

- *Composite Service Client*: This component serves two purposes. Firstly, it acts as an adapter (design pattern) between the composite service and the service mediator, allowing the possibility of heterogeneous service deployment technologies between the two[7]. E.g. Java EJB communicating to a SOAP agent. It also acts as a facade (design pattern) by providing a consistent control and service data model to the service mediator[7] It does this through adapting the control and service data to match that expected by the composite service.
- *Service Mediator*: The service mediator contains the core logic of the service aggregation process. It acts as a proxy (design pattern) for all the other significant components in the framework[7]. Its main responsibilities are to control and collect service data from the composite services via the composite service client, determine and request the appropriate positioning technique used in the 3G network based on the location accuracy required, construct and aggregate the composite services, and deliver the service to the consumer client application via the consumer client adapter. It is assisted in the aggregation process by the service provider selector component.
- *Consumer Client Adapter*: This component acts as an adapter between the consumer client application and the service mediator. It receives the aggregated service data from the service mediator, transforms it to the correct presentation format if necessary (e.g. HTML or WML), and delivers it to the consumer client application in a protocol that the latter understands (e.g. HTTP/SOAP etc). It is acts as a service access point by allowing the consumer application client to request and initiate the aggregated service.
- *Location/Service Provider Mappings*: This is actually a database table that provides a list of Service Providers within a geographical locality (e.g. Radio Access Network Cell, City, Town etc.) for a specific service category. A table entry also states if a service is not available in a locality because of a lack of service providers or it is theoretically impossible to provide the service.
- *Static Service Provider Link Tables*: This is actually a database table that represents a service network/infrastructure for a specific service category in tabular form. Table entries include the number of ‘hops' required for a service provider to get to each other service provider within the service network. This table will be used mainly for selecting a list of available service providers for C1Ln and CnLn aggregated services. It is also used to determine if the service network/infrastructure can support the required service, and thus influence the operation of both transactional and *best effort* aggregated services.

- *Service Provider SLAs*: Service Level Agreements (SLA) are a key part of service management and deployment. They contain information about Acceptable Service Response Time, Guaranteed Network QoS between the aggregated service and the composite service, cost of using the service, Unit of Work definitions etc.[8,9,10]. They are used in the framework to ensure that the performance of any single composite service does not hinder the overall performance of the aggregated service.
- *User Profile and Preferences*: The user profile and preferences houses a list of service specific user preferences, such as cost of using the aggregated service (this can be partly determined from the Service Provider SLAs), location accuracy and response time, for a particular aggregated service.
- *Service Provider Selector*: This component contains the logic of selecting multiple service providers that will form the aggregated service according the user preferences, and other aggregated service requirements discussed earlier.

## 5 Implementation, Experiment Results and Analysis

We implemented our framework on a Sun ONE Application Server 7 in an Intel Pentium III 1.0 GHz server and used a Sun PointBase database server. Most of the components in the framework were written in Java, and the service mediator and service provider selector components were implemented as session EJBs. The LCS in the 3G core network was simulated by a Java application triggering location information using the data model specified in [11].We decided on, and developed, a road traffic router aggregated service to perform our experiments on the framework. This aggregated service provides a user with a choice of routes from the current location to his/her stipulated destination according to the user's preference (e.g. distance, journey duration etc.). Every composite service involved in constructing the aggregated service, e.g. a private toll-paying highway company, provides the routing functionality and traffic monitoring functionality for its own locale in the journey. The road traffic conditions within different locales can influence the routing between the user's embarkation and destination points, and all of the composite services collectively provide a C1Ln aggregated service via a Centralized Control-flow Centralized Data-flow integration model. We chose road traffic router service because the UK Highway Agency and the Transport Research Laboratory were researching on providing a similar service in the UK[12]. A traffic monitoring system also constitutes as a valid value-added service as the monitored roads may be owned by a private company, e.g toll-based roads. Furthermore, our framework can be easily adapted into their business model, thus conceptualizing such a service is realistic and non-idealistic. The routing application in the composite traffic router service used an algorithm based on Dijkstra's algorithm, with the improvements and modifications detailed by Gutman[13], and incorporated distance, average speed, and journey duration as road cost attributes.

### 5.1 Analysis of Prototype Experiments

We decided to examine two deployment scenarios in aggregating the road traffic composite services by implementing all the composite services as EJBs, and as Web Services (SOAP). To build the Web Services version, we used Sun Microsystems' Java Web Services Developer's Pack v1.1, and its JAX-RPC APIs. For the EJB version of the service, we used the Sun ONE Application Server 7 and the associated J2EE 1.3 APIs. Method calls between EJBs were performed through Java's Remote Method Invocation over Internet Inter-ORB Protocol (RMI/IIOP). We performed all our experiments on a single computer as there were no intentions in our experiments to analyze delays caused by network traffic and latency due to network QoS issues. Although the size of the network traffic from control and data SOAP messages for Web Services depends on the design of the Web Service interface, the service data model, and the schema used, we designed them to be similar to those in the EJB scenario in our experiments for comparison purposes. In our experiments, we were more interested on how the choice of a service middleware platform for composite services, e.g Web Services and J2EE, can affect the service response time of the aggregated service, which can be due to the traffic size of the control and data messages, without the effects of delays in the network. We scrutinize the level of response times because it is a metric that will explicitly affect the user's enjoyment of the aggregated service. When performing the experiment, we assume that a service level agreement between the aggregated service provider and the composite service provider has been pre-established, and the relevant data models have been agreed on. We also developed the respective composite service client components (an EJB and a JAX-RPC/SOAP version) for each scenario. To complete the entire scenario, we developed both a Java RMI/IIOP based, and a Java 2ME midlet with SOAP agent based consumer client application each for the EJB and Web Services scenarios respectively. We measured the response times using appropriate time stamps in the programming code and for each scenario, we invoked the aggregated service for five different routes of similar complexity involving three separate composite service providers with different service locality area size. We measured the response time for each route 10 times, excluding the discarded first run – mandatory when testing Java code reliably (a practice attributed to the idiosyncrasies of the Java Virtual Machine). Table 1 shows the response times measured in milliseconds.

**Table 1.** Response times of the experiments (millisecs). WS = Web Service Scenario, EJB = Enterprise Java Beans Scenario

| | **Client** | | **Aggregated Service** | | **Composite Services Total** | | **Client Overheads** | | **Aggregated Service Overheads** | |
|---|---|---|---|---|---|---|---|---|---|---|
| Service Architecture | WS | EJB | WS | EJB | WS | EJB | WS | EJB | WS | EJB |
| Route 1 | 700 | 802 | 423 | 431 | 174 | 303 | 277 | 372 | 249 | 128 |
| Route 2 | 715 | 679 | 407 | 346 | 186 | 247 | 308 | 333 | 221 | 99 |
| Route 3 | 510 | 560 | 242 | 296 | 130 | 181 | 267 | 264 | 112 | 115 |
| Route 4 | 478 | 701 | 242 | 415 | 108 | 281 | 237 | 286 | 134 | 134 |
| Route 5 | 482 | 463 | 270 | 205 | 117 | 99 | 212 | 258 | 153 | 106 |
| Average | 557 | 641 | 317 | 338 | 143 | 222 | 260 | 303 | 174 | 116 |

The ‘Composite Services Total' column in table 1 represents the total time taken for all the composite services to compute an optimum traffic route in its locality without including the time needed to serialize, un-serialize, parse, and un-parse data and control messages. The ‘Aggregated Service' column shows the time elapsed between the consumer client adapter in the framework initiating the aggregation process through the service mediator, and the consumer client adapter having the aggregated service data ready for transmission back to the consumer client application. This too does not measure the time required to serialize, un-serialize, parse, and un-parse data and control messages. The ‘Client' column shows the total time taken in milliseconds from the sending of the aggregated service request to the receipt of the aggregated service response (inclusive of processing all control and data messages) by the consumer client application. The ‘Client Overheads' column represents the time taken to serialize, un-serialize, parse, and un-parse control and data messages between the consumer client application and the consumer client adapter. The ‘Aggregated Service Overheads' column shows the total time taken to serialize, un-serialize, parse, un-parse control and data messages between the composite service clients and the associated composite services. The interesting result in table 1 is the comparison between the web service and the EJB scenarios for ‘Aggregated Service Overheads'. This is mainly the overheads incurred when collaborating with the different composite services. We can see that EJB's RMI/IIOP incurs lower overheads than the Web Service's JAX-RPC/SOAP on the average. This is because when objects and value objects are serialized in RMI/IIOP, they are also compressed before being sent through the network, whereas JAX-RPC/SOAP messages are not compressed. Therefore, the size of RMI/IIOP messages sent and received is less than that of JAX-RPC/SOAP. Furthermore, the parsing and un-parsing of SOAP's XML based messages require more processing power and time than Java's RMI/IIOP.

## 6 Related Work and Conclusions

Though there are many documented service middleware and frameworks developed for location-aware services, our work is most closely related to the IST MOBIVAS project which was a recent EU funded project[14]. Their middleware resides on top of an OSA/Parlay gateway and caters primarily to providing a full service support platform for context-aware value-added service providers. However, their platform differs from our framework by not considering issues concerning service aggregation, and the significance of location accuracy in service performance. [15,16] both cover platforms for service aggregation that emphasizes on dynamicity. These platforms are extremely suitable for on-the-fly B2B transactions and on-demand service SLAs. Our research differs in approach as our business model assumes that B2B relationships and SLAs are negotiated off-line.

The standardization and impending implementation of the LCS technology in 3G mobile communications is indicative that location-aware services will soon play a significant role in the design of future service architectures and frameworks. As the influx of B2B oriented service architectures such *.Net* and Web Services grows, the integration and aggregation of composite location-aware services will seem evermore

attractive to service providers. In this paper, we attempt to identify the main hurdles that aggregated location-aware services will face and suggest a reference framework based on our informed and calculated prediction on the future business model for such services. Nevertheless, our framework can also operate on top of an OSA/Parlay gateway if required.

We identified the three main types of aggregated location-based services, and examined their requirements and possible dependencies on location accuracy and transactional behavior. We have also presented on the various service integration models available for aggregating location-aware services, their benefits pertaining to location-aware services, and ensured that our framework allow flexibility in deploying any of the models available. The experiments conducted with our framework highlighted the subtle differences when aggregating composite services using two well received service middleware architectures: the current predominant favorite J2EE Enterprise Java Beans, and the up and coming Web Services/SOAP based architecture.

There are two certain avenues in which we would further pursue the research detailed in this paper. It would be interesting to perform a more detailed analysis on the size of the control and data messages, so as to find the correlation between the time lost on serialization/parsing overheads and the control and data message size. Another intriguing experiment that we have planned for the near future is to examine the performance of our framework in a Centralized Control-flow Distributed Data-flow integration model, as we feel that this may yield some gains in response times.

We would like to acknowledge that the work presented in this paper has been developed in the context of the POLYMICS project, funded by the UK Engineering and Physical Sciences Research Council (EPSRC) - Grant GR/S09371/01. Alvin Yew would also like to thank Mr. Harry Chang for his insightful comments and advice about business models for the 3G mobile communications market.

## References

[1] 3rd Generation Partnership Project: Open Service Access (OSA); Application Programming Interface (API); Part 1-12. 3GPP TS 29.198.

[2] Parlay Group: Parlay API Specifications 4.0. Available at http://www.parlay.org/specs/index.asp, (current May 2003).

[3] 3rd Generation Partnership Project: Location Services (LCS); Service Description Stage 1. 3GPP TS 22.071.

[4] M. Champion, C. Ferris, E. Newcomer, D. Orchard: Web Services Architecture. W3C working draft dated 14 Nov. 2002, available at http://www.w3.org/TR/2002/WD-ws-arch-20021114/ (current May 2003).

[5] R. Popescu-Zeletin et al: Service architectures for the wireless world. Computer Comms., vol. 26, no. 1, Jan. 2003, pp. 19-25.

[6] D. Liu, K.H. Law, G. Wiederhold: Analysis of Integration Model for Service Composition. Proc. Wkshp on Software and Performance (WOSP'02), ACM Press, Rome, Italy, 2002, pp. 158-65.

[7] E. Gamma et al: Design Patterns: elements of reusable object-oriented software. Addison-Wesley, Reading, Mass, 1994, pp. 139-50, 185-94, 207-22.

[8] L. Lewis, P. Ray: On the Migration from Enterprise Management to Integrated Service Level Management. IEEE Network, Jan-Feb 2002, pp. 8-14.
[9] Keller, A. Koppel, K. Schopmeyer: Measuring Application Response Times with the CIM Metric Model. Proc. 13th IFIP/IEEE Int'l Wkshp Distributed Sys (DSOM 2002), Montreal, Canada, 2002, pp. 66-81.
[10] M. Debusmann, A. Keller: SLA-Driven Management of Distributed Systems using the Common Information Model. Proc. 8th IFIP/IEEE Int'l Symp. on Integrated Network Mgmt (IM 2003), IEEE Press, Colorado Springs, CO, 2003, pp. 563-76.
[11] European Telecommunications Standards Institute: Open Service Access (OSA) API; Part 6: Mobility SCF. ETSI ES 202 915-6v1.1.1, Jan. 2003.
[12] M. Yearworth et al: A CORBA Service for Road Traffic Information on the Internet. Proc. Int'l Symp Dist. Objects and Applications (DOA'00), IEEE CS Press, Antwerp, Belgium, 2000, pp. 231-41.
[13] R. Gutman: Priority Queues for Motorists. Dr. Dobb's Jrnl, Sept. 2002, pp. 89-92.
[14] MOBIVAS: Downloadable Mobile Value-Added Services through Software Radio and Switching Integrated Platforms. Home page at http://mobivas.cnl.di.uoa.gr, (current May 2003).
[15] G. Piccinelli, G. Di Vitantonio, L. Mokrushin: Dynamic service aggregation in electronic marketplaces. Computer Networks, vol. 37, no. 2, Oct. 2001, pp. 95-109.
[16] D-R Liu, M. Shen, C-T Liao: Designing a composite e-service platform with recommendation function. Computer Standards & Interfaces, vol. 25, issue 2, May 2003, pp. 103-117.