RJ 10279 (A0301-026) January 21, 2003
Computer Science

IBM Research Report

Coeus: An Approach for Building Self-Learning and Self-Managing Systems

Sandeep Uttamchandani
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Carolyn Talcott
Computer Science Laboratory
SRI International

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Coeus: An Approach for Building Self-learning and Self-managing Systems

Sandeep Uttamchandani
IBM Research Division
Almaden Research Center
sandeepu@us.ibm.com

Carolyn Talcott
Computer Science Laboratory
SRI International
clt@csl.sri.com

Abstract

Systems are becoming exceedingly complex to manage. There is an increasing trend towards developing systems that are self-managing. The eventual target is to build systems in which the administrator specifies overall goals and policies while the system internally decides how these can be achieved. To achieve this target, we need to automate the administrator's task of tuning configuration parameters and invoking of system services such as back-up, replication and so on. The task of automation is non-trivial because of two reasons. First, administrators use implicit heuristics to decide how the system is tuned. Second, by monitoring the impact of their previous decisions, administrators can refine their heuristics i.e. self-learning.

Policy-based infrastructures have been used to provide a limited degree of automation, by mapping actions to specific system-states. But, there is no systematic approach or mechanism within this infrastructure to support automation of the administrator's tasks mentioned above namely decision-making and self-learning. This paper proposes an approach to solve this problem by using Coeus: a model and framework for building self-managing systems. The key contributions of this work are:

- Describes a model to help administrators and system-builders specify policies that can capture the implicit management heuristics.
- Using the Coeus model, it describes automation of the tasks of decision-making and self-learning.
- Describes the Coeus framework in the context of existing policy-based management infrastructures.

1. Problem Statement

Systems are becoming exceedingly complex to manage. The cost of administration is becoming a significant percentage (75-90%) of the Total Cost of Ownership (TCO) [5,14]. The vision to build self-managing systems is not a new one. Jim Gray in his Turing award speech "What next? - A dozen IT research goals" [8] described a self-managing system as one in which the administrator sets goals and create high-level policies, while the system by itself decides how they are executed.

Administrators manage systems by tuning configuration parameters and invoking system services. One of their primary tasks is to ensure that the system achieves the performance goals specified by the Service Level Agreements (SLA). They employ the following action-loop to manage systems: Monitoring → Deciding changes → Tuning the system. The task of decision-making is based on monitoring the system and using implicit heuristics associated with each of the configuration parameters and system services.

Automating the working of the administrator is non-trivial because of two reasons. First, there is no straightforward approach to express the implicit management heuristics and having a decision-making process based on those heuristics. Second, administrators learn from the decisions that have been made and refine the existing heuristics i.e. self-learning.

Currently, policy-based infrastructures have been used to provide a limited degree of automation, by mapping actions to specific system-states [4]. The policy-based approach is fundamentally based on the Event-Condition-Action (ECA) model. Within this model, it is complex (but not impossible) to encode the decision-making and self-learning semantics of the administrator. Decision-making can be encoded by defining a complex entity-relationship between the configuration parameters. Similarly, self-learning can be encoded by defining some variables that keep track of system state and are used during decision-making. In addition to being complex, the implementation will be very specific to those set of parameters. Further, any additions or changes to the system parameters or services is difficult to accommodate.

In summary, administrators and system-builders have no systematic model or framework to express the implicit heuristics of the administrator that can be used to automate the process of decision-making and self-learning. This paper proposes Coeus as an approach to solve this problem. This approach can be used by extending the existing policy-based infrastructures.

The paper is organized as follows. Section 2 enumerates the terminology used. Section 3 analyzes how administrators manage systems and identifies their approach for system management. The Coeus model is presented in Section 4 followed by the framework in Section 5. Section 6 discusses the related work followed by the conclusion.

2. Terminology

Dimensions of behavior:
The term "behavior" is used loosely to describe the properties of the system. If we treat the system to be a black-box, its observed behavior can be specified using abstractions such as QoS goals, transaction-properties[2], etc. In each of these abstractions, behavior is a composition of multiple *dimensions*. Figure 1 illustrates this notion, representing system behavior to be composed of dimensions such as throughput, latency, reliability, security, availability and so on.

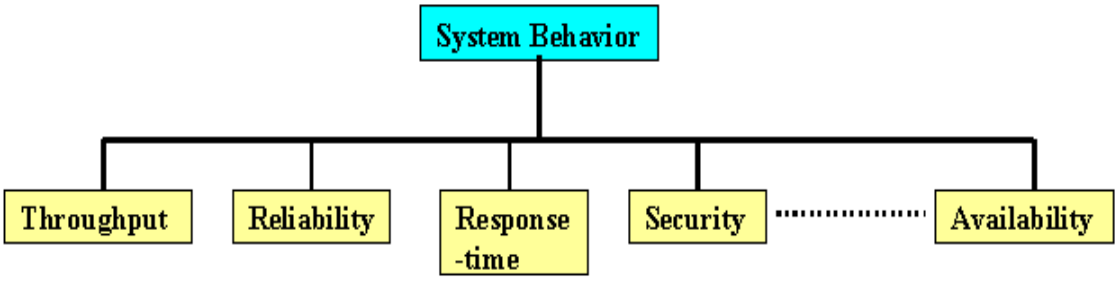

Figure 1: Dimensions of behavior

Management-knob
Broadly classified, administrators have two sets of controls for managing the behavior of the system: First, there are configuration parameters which are either application-specific or system

variables such as buffer-size, cleaning delay, number of outstanding I/Os. Second, there are system services that are invoked in certain scenarios. For example, in distributed file system, there are services such as backup, data-migration, replication and so on. These parameters and services are together referred as "management knobs."

The system-services themselves have tunable parameters. The current Coeus model does not consider these and is a part of future work.

Low-level system-state
It represents details of the system such as workload characteristics, resource utilization and system events. Resource utilization is 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, disk failures, etc.

3. Understanding how administrators manage systems

There are two aspects to the knowledge that administrators have for each management-knob. First, the implicit heuristics for the effectiveness of each knob i.e. the impact of the knob on system behavior. For example they know that invoking the replication service has an impact on throughput, response-time and availability [18]. This notion of effectiveness is derived in part from the know-how of the inner system details (in the manuals) and also from experience. The second aspect of the knowledge is the mapping between the low-level system state and the management-knobs i.e. the low-level system state when invoking a specific management-knob will have an impact on behavior. For example, invoking data replication makes sense if the workload access-pattern predominantly consists of reads and when additional resources are available.

As mentioned earlier, administrators manage systems by using the action-loop: Monitoring $\rightarrow$ Decision-making $\rightarrow$ Tuning management-knobs. The decision-making stage determines the action required to meet the SLA. It is a sequence of two steps. First, analyze the low-level system-state and determine a list of candidate management-knobs that can be possibly invoked. Second, use the implicit heuristics of effectiveness to determine the knob(s) to be tuned from among the short-listed knobs in step 1. In summary, the working of the administrator can be represented as a function that takes as input the SLA goals and their current-values. It outputs the management-knob(s) to be invoked based on the low-level system-state and implicit heuristics i.e. f(goals, state) $\rightarrow$ management-knob. This is illustrated in figure 2.

An important aspect in the working of the administrator is the learning based on the decisions that have been made. Each time the management-knob is invoked, the administrator can monitor the system and refine his implicit notion of effectiveness for the corresponding knob.

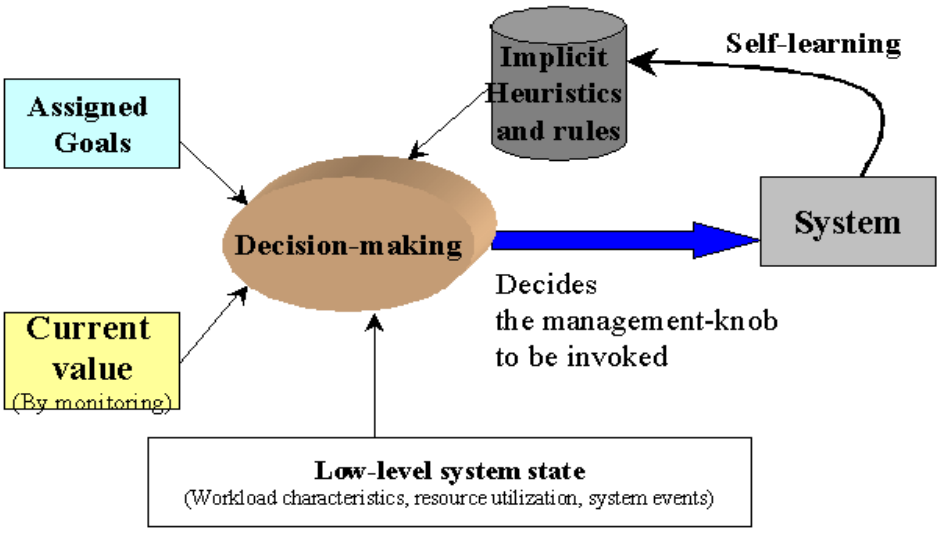

Figure 2: Understanding the working of the administrator

4. Coeus Model

The aim of the Coeus model is to capture the implicit heuristics of the administrator in order to automate the task of decision-making and self-learning. In this model, system behavior is defined as a summation of multiple dimensions. These dimensions have different relationships among themselves. e.g. throughput and reliability generally have a trade-off relationship. Within this definition, each management knob is considered to have an impact on one or more behavior dimensions. This concept is illustrated by the following example.

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 SLA goals specified in terms of throughput, latency, reliability, availability and security

To meet the desired goals, the administrator tunes the file-system using the available management-knobs. In this example, the file-system has tunable parameters namely clean-delay, pre-fetch size, data integrity check and number of server-threads. The file-system services are load balancing, data replication, volume migration, data backup and security services such as authentication and encryption.

In the Coeus model, the administrator or system-designer creates management policies by specifying two sets of details. First, the low-level system state on which each of the management knobs depend i.e. the pre-qualification for invoking the knob. This is similar to the information specified in existing policy-based QoS management. Second, they specify the impact of each management-knob in terms of dimensions of behavior. This is similar to the implicit heuristics that administrators use during system management.

The administrator specifies the impact on behavior dimensions using a abstract description space of terms such as increases, decreases, increases significantly, decreases significantly and unspecified. The accuracy of these specifications doesn't really affect the working, as the system will eventually learns via self-learning. Table 1 summarizes the management policy described above.

Capability	**Low-level system-state (Pre-condition)**	**Behavior Implication**				
		Throughput	**Response Time**	**Availability**	**Security**	**Reliability**
Configuration Parameters						
Clean-delay	Memory available && repeated writes to same data blocks	☑ ↔	☑ ↔			☑ ↔
Pre-fetch size	Sequential access-pattern	☑ ↑+	☑ ↓			
Data Integrity Check (ON/OFF variable)	Application imposed requirement	☑ ↓ -	☑ ↓ -		☑ ↑	☑ ↑
Number of server threads	CPU cycles available and system not I/O bound	☑ ↔	☑ ↔	☑ ↑		
System Services						
Load balancing	Resources not uniformly utilized	☑ ↑	☑ ↑			
Data replication	Access pattern predominantly consists of reads	☑ ↑	☑ ↓	☑ ↑+		
Volume migration	Non-uniform utilization of disks	☑ ↑	☑ ↑	☑ ↑		
Data Backup	Low system-load OR multiple system errors	☑ ↓				☑ ↑
Security services: Authentication and data encryption	Application imposed requirements	☑ ↓	☑ ↓		☑ ↑	

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

Table 1: Illustrates the information specified by the Administrator

Each of the behavior dimensions is quantified by parameters that can be actually monitored. Table 2 lists a few example parameters.

Behavior Dimension	Example of parameters to quantify
Reliability	MTBF, Time-to-repair (TTR), Number of Failures, Type of Failures
Availability	Average number of outstanding I/Os, number of re-transits, queuing delay
Security	Number of integrity failures, authentication supported, level of encryption

Table 2: Parameters to quantify behavior dimensions

Let us now consider the task of decision-making. Table 3 shows the current values of the goals. As shown, the goals for throughput and availability are not being met. Based on the low-level system-state, assume that the following management-knobs from table 1 qualify the pre-condition: Pre-fetch size, Data replication service and Volume migration service.

	Goals achieved	% Change required [% Change Tolerated]
Throughput	×	15[-]
Response-time	☑	0 [2]
Availability	×	8[-]
Security	☑	0 [24]
Reliability	☑	0 [35]

Table 3: Illustrating current system state

The decision-making is based on 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 3, the value of response-time cannot be changed by more than 2%. Thus, based on the above analysis, the volume migration service is invoked.

This is a simple example, but it demonstrates how management-knobs are selected. We can similarly consider other scenarios. In summary, the task of decision-making is mapped to analyzing the behavior implications of management-knobs.

The task of self-learning involves refining the existing behavior dimensions associated with the management-knobs. Each time a knob is applied, the % change in the system state is recorded along with the corresponding change in the knob value. The actual monitored values are appended to the management policy in Table 1.

A formal analysis of the Coeus model, in terms of the vector space is presented in the Appendix.

5. Coeus Framework

This section describes the Coeus framework in the context of existing policy-based infrastructures.

5.1 Background: Existing Policy-based Infrastructure

Policy based infrastructures have been used to automate the task of management [13]. The underlying policy specification model is based on ECA i.e. Event → if (Condition) → then (Action). There are various approaches (i.e. syntax) for specifying policies. They can be specified as a programming language that is processed and interpreted as a piece of software [7,9] or specified in terms of a formal specification language [15,16] or the simplest approach is to interpret policies as a sequence of rules. The IETF has chosen rule-based policy representation in its specifications [1]. Thus, in this paper, we refer to policies as a collection of rules. In summary, policies define the mapping between the low-level system state and the adaptive action.

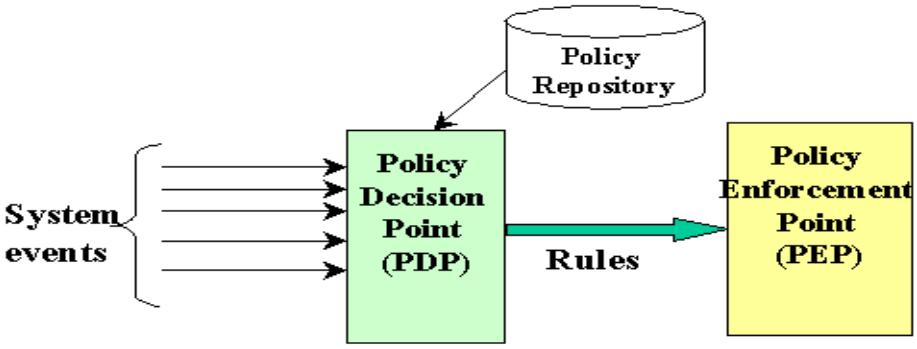

Figure 3: Components of a Policy-based Infrastructure

Existing policy-based infrastructures consist of 3 key entities: A repository, Policy Enforcement Point (PEP) and Policy Decision Point (PDP). As illustrated in figure 3, 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.

5.2 Details of the Coeus Framework

Figure 4 illustrates the Coeus Framework. It working can be defined as a sequence of three stages

Stage 1: Pre-qualification of management- knobs
The rule-filter analyzes the low-level system state and determines the configuration-knobs that can be invoked. For the example in Section 4, the pre-condition for changing the pre-fetch size was based on the workload access-pattern.

Stage 2: Decision-making for selecting knob
As shown in the figure, the Capability Broker compares the SLA goals and their current-values. Then, by analyzing the behavior dimensions of configuration-knobs, it decides the knobs to be invoked. The information in the rule repository will be similar to Table 1.

Stage 3: Self-learning
After the rule is invoked, its impact on system-behavior is monitored and recorded. This information is used to update the behavior dimensions associated with the management-knob.

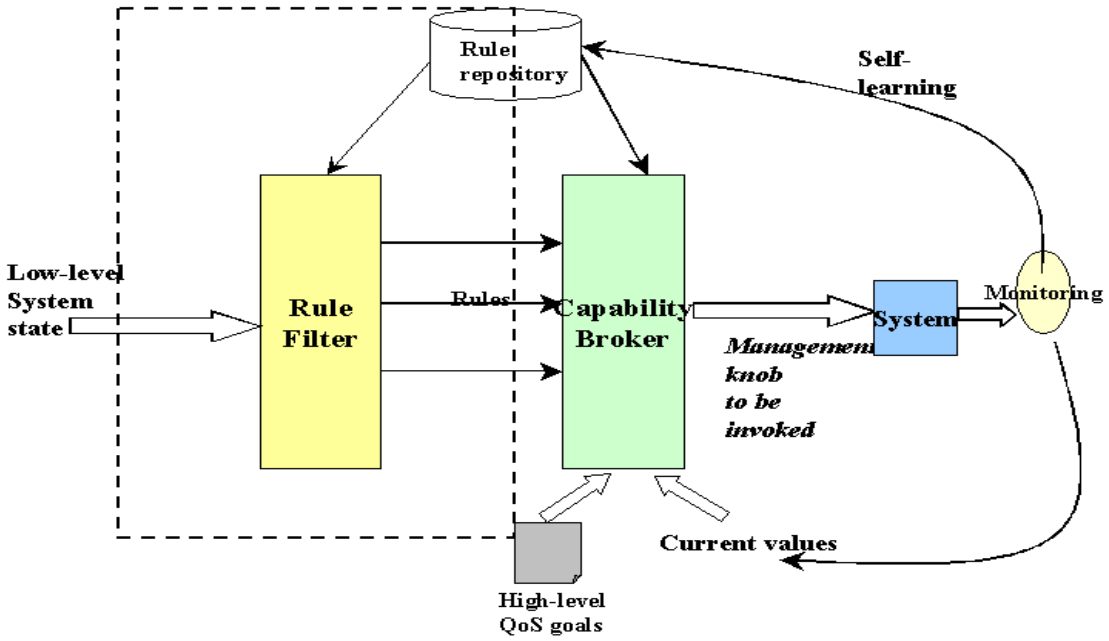

Figure 4: Architecture for self-managing systems

5.3 Using the Coeus approach in existing policy-based infrastructures

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 4 illustrates this. Adding one more reasoning layer i.e. the Capability Broker to the existing infrastructure, allows for higher-order operations on rules. Finally, the specification of policies using the Coeus model allows refinement of the policy via feedback from the monitoring infrastructure

To support the Coeus model, the existing rule-based specification is extended by defining behavior implication as a wrapper around the existing rule (Figure 5). The behavior implication is defined as a collection of tuples of the form: <Behavior Dimension><Impact>.

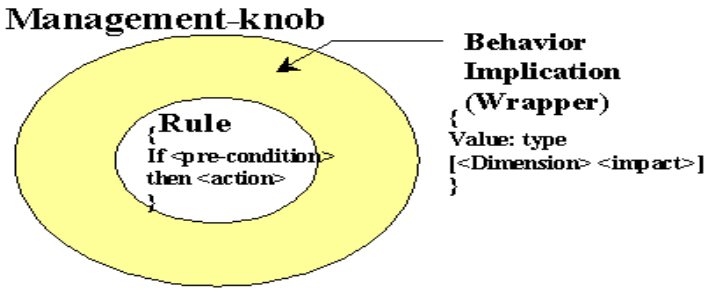

Figure5: Extending rule-based specification to define capabilities

To illustrate this concept, we define behavior implications using the following simple notation in BNF.

Capabili**t**y:= <Parameter> <Implication> <Rule>
Parameter:= <Parameter-name>: <Parameter-type>
Parameter-name:= <Configuration-parameter-name> | <System-service-name>
Parameter-type:= Integer | Boolean| Floating-point
Implication := <Dimension> <Impact> | <Implication><Implication>|NULL
Dimension:= <Performance> |<Reliability> | <Availability>| <Security>| <Error-recovery>
Impact:= <Increase>| <Decrease>
Rule:= Specified using existing rule-based languages

As an example, consider the specification of the data-replication service.

Capability
{
Data-replication: Boolean
Implication
{
Performance Increase
Latency Decrease
Availability Increase+
}
(Treated as black-box)

Rule
{
If (num_reads/num_writes > 0.9)
Then replication = ON
}
}

6. Related work

Our work is complementary to most of the existing self-managing research systems in the domains of extensible operating systems [12], databases [17], file systems [10], and storage devices [11]. For example, the VINO[12] project has investigated the design of self-managing techniques for various OS tasks such as paging, interrupt latency and disk waits. Our work can be seen as providing a framework to coordinate the adaptations among various sub-systems in order to achieve the overall system goals.

Zinky et al. [6] 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.

[3] 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.

7. Future Work

In order to be more effective in real-world scenarios, the existing Coeus model can be enhanced along the following dimensions:

- The implicit heuristics of the administrator can be extended to capture combinations of management-knobs that are invoked together, values of the management-knob for specific workload characteristics and so on.
- The system-services themselves have tunable parameters. E.g. Data replication service has parameters such as number of replicas, % data replicated. In this paper, we assume that once invoked, the services themselves decide the value for these parameters. The Coeus model can be extended to decide the values for these parameters.
- In self-learning, including the specification of workload characteristics along with the degree change.

8. Conclusion

This paper is aimed as a starting-point in describing a systematic approach to build self-managing systems using existing policy-based infrastructures. It proposes to extend the policy-based management model in the following manner. First, it provides explicit semantics to capture the mapping between the rule and its implication on system-behavior. Second, it provides a mechanism for reasoning among rules especially when there is more than one rule that is applicable i.e. it treats rules as first-class entities and allows higher-order operations on them. Currently the only flavor of higher-order operations among rules is in terms of priorities and rule hierarchy. Third, it allows for self-learning i.e. adding information to the rules based on the feedback from previous decisions.

9. References

[1] The IETF Policy Framework Working Group. http://www.ietf.org/html.charters/policy-charter.html.
[2] B. Sabata, S. Chatterjee, M. Davis, J. Sydir, T. Lawrence. Taxonomy for QoS Specifications. Workshop on Object-oriented Real-time Dependable Systems (WORDS), 1997.
[3] D. Feitelson, Michael Naaman. Self-Tuning Systems IEEE Software 16(2), pp. 52-60, 1999.
[4] D. Verma. Simplifying Network Administration using Policy based Management. IEEE Network Magazine, March 2002.
[5] E. Lamb. Hardware Spending Matters. *Red Herring*, pages 32–22, June 2001.
[6] 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.
[7] J. Fritz Barnes and Raju Pandey. ``CacheL: Language Support for Customizable Caching Policies. In Proc of Web Caching Workshop (WCW), March 1999.
[8] J. Gray "What Next? A Dozen Information-Technology Research Goals," ACM Turing Award Lecture, June 1999, MS-TR-99-50
[9] J. Hoagland, "Specifying and Implementing Security Policies Using LaSCO, the Language for Security Constraints on objects". Ph.D. Dissertation, UC Davis, March 2000.
[10] 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.
[11] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. A*CM TOCS*, pages 108–136, Feb. 1996.
[12] M. Seltzer and C. Small. Self-monitoring and self-adapting operating systems. In *Proc. of HOTOS Conf.*, pages 124–129, May 1997.
[13] M. Sloman, E. Lupu. Security and management policy specification. IEEE Network, pp. 10-19, March-April 2002
[14] N. Allen. Don't Waste Your Storage Dollars. Research Report, Gartner Group, March 2001.
[15] 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.
[16] 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.
[17] S. Chaudhuri and V. Narasayya. AutoAdmin "what-if" index analysis utility. In *Proc. of ACM SIGMOD Conf.*, pages 367–378, June 1998.
[18] S. Mullender, Distributed Systems. Addison-Wesley 1993.
[19] Salton, G. and McGill, M. J. *Introduction to Modern Information Retrieval*. McGraw Hill, New York, 1983.

Appendix: Formal description of the Coeus model using Vector spaces

The concepts of the Coeus model are represented as elements and operations within a vector space. Vector spaces have also been used in other areas of research such as information retrieval [19].

The dimensions of behavior have different relationships among themselves. 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 1. 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. It is summation of the components along the behavior dimensions and is of the form:

$$\text{Current-state} = \sum_{i=1,n} a_i \, t_i$$

where a_i is the component along the dimension t_i.

SLA goals define a subset within the behavior space. This space represents a range of values that are permissible. The aim of system management is to ensure that the current-state of the system is within this space (Figure 2).

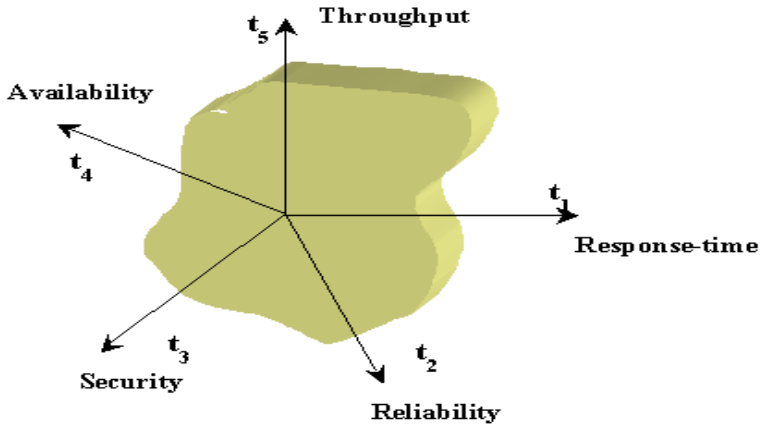

Figure 3: Representing system behavior in terms of vectors

By definition, each management-knob has an impact on one or more dimensions of system behavior. The amount of impact on different dimensions may vary. The impact of the management knob on the system state is a function of the current system state (α_j) and the degree change in the value of the knob (β_j)

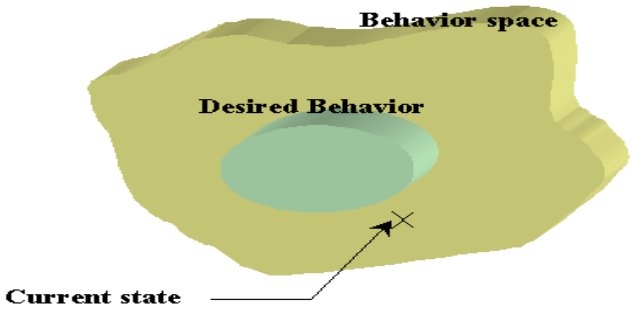

Figure 2: Desired and current behavior space

A management knob is represented as a difference vector between the current-state and the new-state created by invoking the knob. It is of the form:

Management-knob $= \sum_{j=1,n} x_j \; t_j$

where $x_j = g(\alpha_j, \beta_j)$ is the component along the dimension t_j. Depending on the value of α_j, the function $g(\alpha_j, \beta_j)$ can be linear, polynomial, exponential, etc with the change in knob value β_j.

To simplify the representation of the management-knob, we make the following assumptions: First, we record the value of system-state (α_j) separately i.e. the knob is a set of vectors, each representing the knob for a non-overlapping system-state. Second, we assume that the dimensions of behavior are linear with the degree change in the parameter value. The new representation is as follows:

Management-knob $= \sum_{\text{system-states}} \{\sum_{j=1,n} (x_j \; t_j) \; . \; \Psi \}$

Where x_j is the component along t_j. Its value is derived by monitoring the % change in the system. Ψ is a scalar and signifies % change in the value of the knob.

During the task of *decision-making,* the first step is to generate the difference vector between the goals and the current-state. It is referred as the target vector. The target vector encodes the dimensions of behavior that need to change and the degree of change.

To illustrate the concept of decision-making, we enumerate two simple strategies:

- Cosine approach:
  In the vector representation, the cosine of the angle between two vectors is the measure of their similarity. In this approach we find the cosine of the angle between the target-vector and each of the management-knobs. By arranging the knobs in a descending order of the cosine value, we determine the closest knob that matches the requirements. The dot product is then used to derive the value for the degree change in the knob.
- Vector addition approach:
  In this approach, we derive combination of management-knobs that are invoked to match the requirements as shown in Figure 3. Deriving these combinations can be NP-hard. Since the decision-making need not be precise, we are currently exploring approximate techniques to derive these combinations.

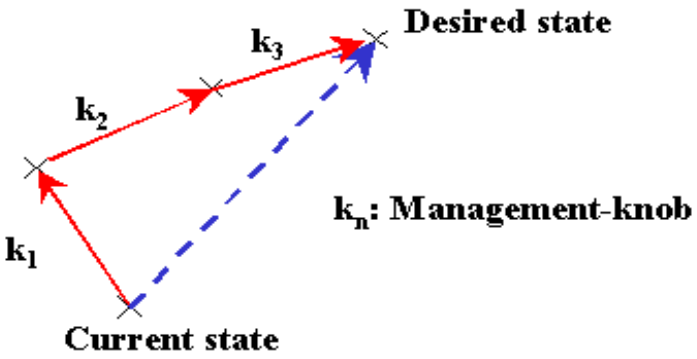

Figure 3: Decision-making by vector addition

Self-learning involves adding information to the existing knowledge of the impact each knob has on system behavior. Every time the knob is invoked, the changes to system behavior are monitored. This feedback knowledge is stored in the form of a triplet of the following:

- Current state value
- % change in state value by invoking the knob
- % change in the knob value

In the vector-space model, it may or may not know the relationship between the behavior dimensions (t_j) i.e. there is an assumption that t_i and t_j are orthogonal i.e. $t_i \cdot t_j = 0$ if $i \neq j$. This may or may not be always be true. A more precise representation of the vector space will be in terms of a vector basis where n linearly independent vectors generate a n-dimensional subspace. For simplicity, this has not been included in this discussion.