Eos: An Approach of Using Behavior Implications for Policy-based Self-management

Sandeep Uttamchandani, Carolyn Talcott*, and David Pease

IBM Almaden Research Center, San Jose CA
{sandeepu, dpease}@us.ibm.com
*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 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.

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.

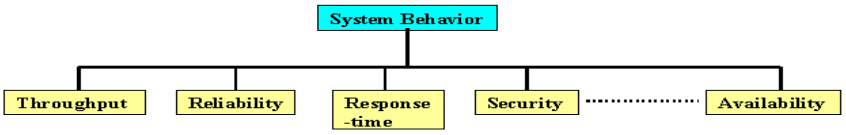

Fig. 1. Dimensions of behavior

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.

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.

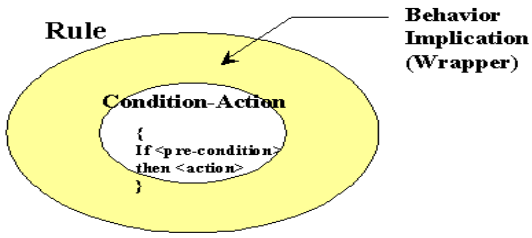

Fig. 2. Extending existing specification model with Behavior implications.

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, ….. 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.

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:

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

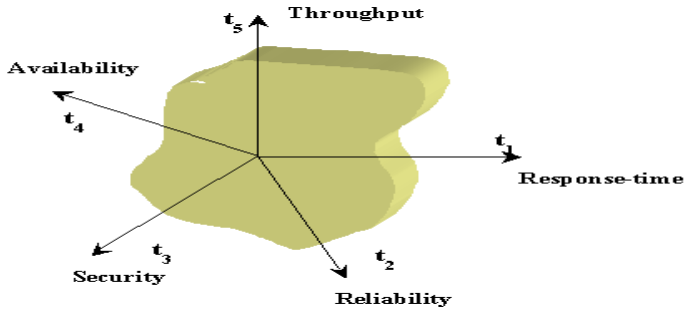

Fig. 3. Vector space to represent system behavior.

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(data-replication) = [(0.3)Throughput – (0.1)Latency + (0.2) 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 (β)
- Workload characteristics when the knob was invoked (γ)
- Current value of the knob (η)

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.

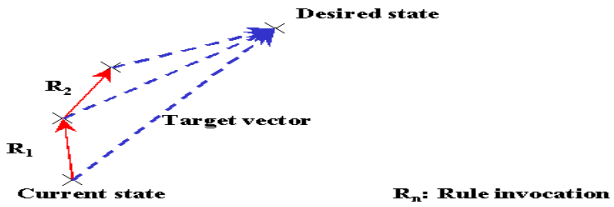

Fig. 4. Strategy for combining rules.

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.

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

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	☑ ↓	☑ ↓			☑ ↑

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.

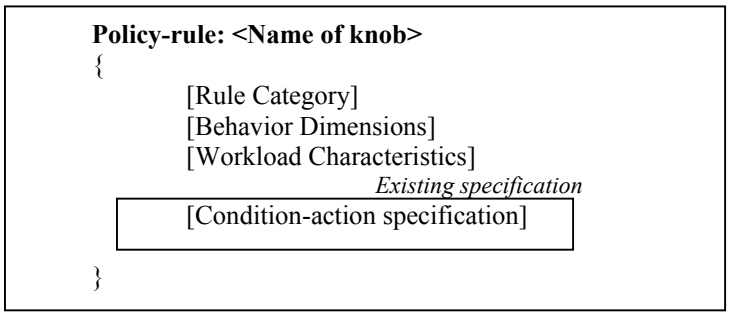

Fig. 5. Specification template.

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
}
}

6.2 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).

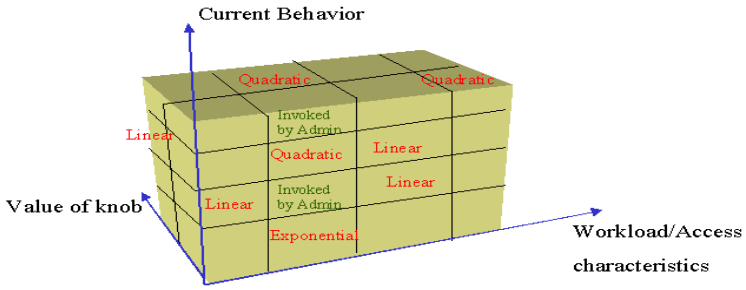

Fig. 6. Self-learning by dividing space into "zones."

6.3 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.

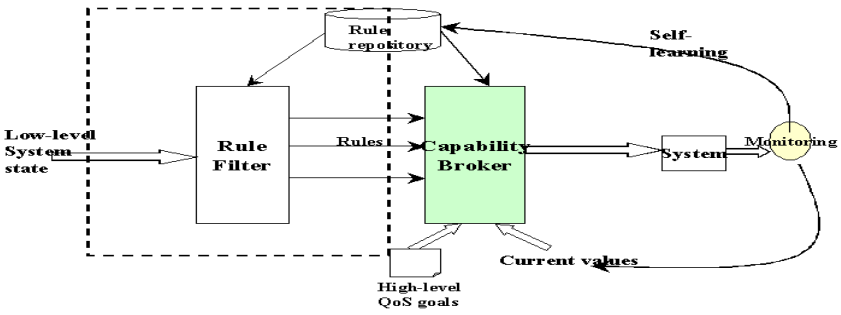

Fig. 7. Components of the Eos Architecture.

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