Policies, Rules and Their Engines: What do They Mean for SLAs?

Mark Perry and Michael Bauer

Department of Computer Science, University of Western Ontario
London, Ontario N6A 5K7 Canada
{markp, bauer}@csd.uwo.ca

Abstract. In our model for autonomic management of service level agreements (SLA), the roles played by policy and rules must be clearly differentiated. Although policy is typically an ideal we wish to achieve through the implementation of rules, the use of the terms policy and rule are often treated similarly and that consequently policy-engine and rule-engine are often used synonymously. It is our position that in the management of SLAs these terms have specific meanings. The definitions and models embodied are illustrated.

1 Introduction

The literature is littered with evidence for the need to make use of policies to manage various aspects of systems and networks [1,4,5,6]. Such management software inevitably requires the means to capture and represent policies and often makes use of rule based engines to determine actions to take. More recently, such management systems have begun to incorporate policy engines [1]. In much of this work there is significant overlap between "policy" and "rule", though "rule" is often taken to be the realization or implementation of a policy as embodied in a rule engine. In other work, there is more distinction, such as in [4,7] where a policy is information associated with the behaviour of a system. The range of use of these terms, the variety of contexts (e.g. network, system, application) and variety of approaches have, consequently, led to the terms "rule engine" and "policy engine" to be taken to mean similar modules. Although little distinction has been made between policy and rule driven systems within management frameworks, drawing a line between these two concepts is required to clarify the approaches and models that we can use in systems to manage SLAs. There is also confusion between rule- and policy-engines and the techniques used to manage their implementation. Based on our initial work on the use of policies in the automation of SLA management, it has become necessary to clearly identify distinctions between these terms. This lack of clear distinction represents a stumbling block for autonomic systems that must adjust and react to circumstances in order to adhere to SLAs. These intelligent systems will need to eventually incorporate rule- and policy-engines and make appropriate use of them in management. In this paper we define rules, policies, rule- and policy-engines specifically for the SLA environment and illustrate these via an example.

M.Gh. Negoita et al. (Eds.): KES 2004, LNAI 3213, pp. 1164–1170, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Policy or Rule

A policy is the highest level of abstraction of implementation of an ideal. A policy defines or specifies a desired state – one that should be maintained or one that should be achieved. With the SLA it is desirable to stipulate the typical levels of service, i.e., the desired levels of service during certain time periods (these may be seen as the policies for service levels), with clearly defined minimum service levels. Absolute minimums may be interpreted as rules with a condition and a subsequent (possible) action in specific instances, though at this level of implementation the flexibility of the overall policy is not in question.

A general analogy may help clarify the distinction between the policy and the rule. We may have a government policy to reduce personal borrowing, but cannot have a simple rule "reduce personal borrowing" as this is not sufficiently prescriptive. However, we can have a bundle of rules implemented that will bring about the policy, such as a rule on maximum interest rates for borrowing, a rule on maximum debt level an individual may incur, a rule that limits lending and other rules that can help realize the policy. In this economic situation, people are not stopped from borrowing under the policy, they must simply operate with rules in place that make the option less attractive. The rule is something that must be complied with in a particular circumstance. Failing to abide by the rule will mean that a sanction is applied (such as fine or unenforceability of a non-compliant loan). The policy is at a higher level of abstraction and rules are used to help achieve this objective.

Returning to SLA framework, the question arises as to where we are using rules and policies for autonomic implementation. The policies specify broad sets of requirements or "service levels", whereas the rules specify to the management system how such requirements can be achieved and how "violations" can be avoided are detected and handled. There are also various levels of policy from the highest level to lower level policies that approach our definition of a rule. How they are described depends on where we are sitting…. from the boardroom perspective there may be the decision that the service 'policy' is to give customers very flexible licensing options – such as ability to choose whether access to software is for a set amount of time or set number of executions.

Thus we can say, at a minimum, that for a SLA the term 'rule' means a circumstance where there is a condition test with an action specified, typically illustrated by the 'If… then…' construct. The policy, on the other hand, is at the higher level of abstraction. The term 'policy' is a set of conditions on one or more states associated with a particular system, with, perhaps, associated elements defining what constitutes "the system", how those states are determined and how elements of the conditions are measured. We illustrate this with an example, presented in Figure 1, from an SLA (necessarily abbreviated) that defines Internet network access.

This agreement, based on a service provider's published SLA, illustrates policies in the more abstract sense. 1. Several policies are embedded within this single SLA, and within the parts of the SLA – ones dealing with availability, network latency, packet delivery and even reporting. 2. In SLA.1, availability is determined based on the system

Internet Service Level Agreement for AAA Networks

SLA.1

The Availability Guarantee declares that the AAA Network (as defined in the applicable service agreement) will be available 100% of the time. If AAA fails to meet this Guarantee during any given calendar month, the Customer's account will be credited…

SLA.2

The Latency Guarantee averages round-trip transmissions of 50 milliseconds or less between designated inter-regional transit backbone routers ("Primary Routers") in the continental U.S. The transatlantic Latency Guarantee averages round-trip transmissions of 90 milliseconds or less between a Primary Router in the Toronto metropolitan area and a Primary Router in the London metropolitan area. Latency figures are achieved by averaging sample measurements taken during a calendar month between Primary Routers.

SLA.3

There are two types of reporting guarantees, a Network Outage Notification Guarantee and a Scheduled Maintenance Notification Guarantee. The Network Outage Guarantee provides Customer notification within 10 minutes after it is determined that the service is unavailable. The standard procedure is to ping the Customer's router every five minutes. If the router does not respond after two consecutive five-minute ping cycles, the service will be deemed unavailable and the Customer's point of contact will be notified by telephone, e-mail, fax, or pager.

SLA.4

The North American Network Packet Delivery Guarantee is packet delivery of 99.5% or greater between designated Primary Routers in North America, measured by averaging sample measurements taken during a calendar month.

Fig. 1. Example Service Level Agreement

defined in a separate service agreement (not included here). This will clearly differ between clients and so there will be a different, though similar, policy associated with each client. 3. Consequences of not meeting SLA.1 are also specified. 4. Certain measurements are defined, e.g. SLA.2, where the round-trip time for a transmission is specified between specific points. This includes specification of bounds. 5. The level of expected packet delivery is defined in SLA.4. 6. The notification policy embodied in SLA.3 provides for notification of service outages, both scheduled and unexpected. The example SLA outlines the type of basic service that we can expect from this network service provider, as well as measurement metrics and some exception handling.

Interestingly, the policies that can be readily identified within this SLA may also create derivative policies that would be internal to the Internet provider in terms of how it would manage its network to ensure that it meets the SLA. This could include

policies that deal with increased latency over shorter service periods, e.g. a few minutes. This kind of policy might be considered pre-emptive in that being able to detect limited periods of increased latency and addressing any consequent problems, might avoid violation of the SLA. Note that specific rules for each of the policies must be defined.

3 Implications for SLA Management

Our interest is primarily in being able to manage SLAs on-line, supporting an on demand infrastructure. The above example illustrates an SLA as, basically, a monolithic agreement, the exception being the definition of the client system. It is certainly conceivable, and even desirable, to enable clients to specify elements of an SLA to their own situation. Using the above network service provider as an example, one could consider clients choosing a "customized" SLA which has higher latency guarantees, perhaps at reduced service costs, specifying different routers for measurement, etc.

The aim is to allow customers to choose the type of licensing (SLA) scheme that they want and, consequently, the policies that they are concerned about. We see the need for a *policy engine* to facilitate such choices on line. So how do we design a policy engine to execute our flexible SLA, to allow customers wide flexibility in their choice of license arrangement with the service provider? The 'customer' can be an in-house client or subscriber to a service, as illustrated by the above SLA, or could be an organization making use of a computation or storage grid providing additional, on demand, resources [2,3]. The latter, of course, is interesting in that the "on demand" nature means the use of resources may not be anticipated and lengthy negotiations to establish SLAs may not be possible.

When it comes to the realization of a client's SLA, after the choices have been made, it is likely that we shall have a number of policies derived from the SLA and likely other policies as derivatives, such as internal policies; let us call these policies *operational policies* since these will likely be policies created to ensure operation of the environment and, most importantly, avoid violation of the SLA. We see the polices derived from the SLA as being the purview of the policy engine. Some operational policies may also be derived from the SLA, but are likely determined by the provider. Given our definitions earlier, we would expect to have rules in place to concretise the implementation. We envisage the policy engine assembling the bundles of rules that we need to implement a particular policy, both for SLA policies and operational policies. Hence, we need both policies and rules in our SLA framework, and if we have on demand and autonomic computing services this implies that we need both policy engines and rule engines.

Rules that fall within the various policy definitions can be constructed to facilitate aims of the policy. A simple example would be the Network Outage Policy in SLA.3 of the AAA Network example. Here, the policy is to inform customers when the network is down. (if the network is out for more than 10 minutes, then the Customer should be notified). A rule associated with this policy might be:

```
Rule: OutageDetect

Customer: IP.IP.IP.IP

    Count: Integer

        1: ping Customer

        2: if no_ping then Count +1; else Count=0

        3: wait 5 minutes

        4: if Count>1 phone customer
```

So for the autonomic SLA scenario, we have the policy engine enabling the kind of policy this particular user is able to engage. It is also providing the means for the service provider to define the kinds of policies that they are willing to operate the service under and then facilitate the implementation of the rules that will execute within the framework of that policy. This clearly is more flexible than a simple rule engine, (or simple decision tree). This relationship is illustrated in Figure 2.

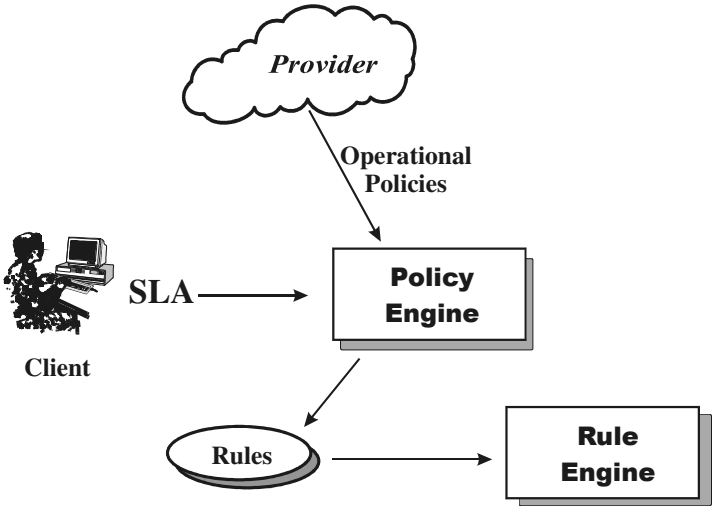

Fig. 2. SLA management

We see a parameterized SLA, perhaps in an on demand scenario for resources or services, being customized by a client to meet their particular needs and timeframe. This customised SLA is processed by a policy engine that, with additional operational policies defined by the provider, generates rules to be used in the management of the underlying system.

This approach raises several interesting questions. Consider the situation in which multiple users are involved with a single service provider in an on demand facility (becoming an increasingly common occurrence), then the policies selected by the individual users, while consistent on an individual basis, may in fact create conflicting or anomalous situations for the provider. For example, suppose there is 10 Mbits of bandwidth available and 2 separate users opt for SLAs that guarantee at least 8Mbs – What does the service provider do? Should the policy engine dynamically adapt the available policies as a result of other user choice? Does this mean that in the dynamic creation of SLAs the policy engine must include some kind of "admission policy" or be able to estimate anticipated resource needs? We are just beginning to explore these questions.

A provider offering software as part of its service is another example that illustrates levels of abstraction of policy within an autonomic SLA model. The greatest level of abstraction would be a management policy: "give users great freedom in their contract (SLA) choice for flexible software licensing". At a lower level we have policies that allow customers to choose the type of licensing for their SLA. This is particularly useful for service providers that supply software components on demand.

By adopting a policy driven licensing model we can allow for the supplier to offer a licensing model that fits the needs of the customer [9], and the customer could choose to have time-based licensing, concurrent, capacity licensing or any of the common licensing types. These different types of policy that we are going to allow within our overarching policy of 'flexibility' can be regarded as policies rather than rules, as they serve as the guide for the various rules that will be implemented within each choice. For example, an SLA that uses a time-based licensing option allows users to contract for services for a set period of time. The details of the time-based licensing can be best executed by rules, of which there can be many. However, two simple rules illustrate the concept, the first rule being a calendar based license agreement, the second being run time based:

```
Rule: DateBasedTerminate

CurrentDate: Date

ExpiryDate: Date

          1: if CurrentDate>ExpiryDate then Terminate

Rule: PeriodBasedTerminate

TimeGranted: Time

TimeConsume: Time

          1: if TimeConsumed>TimeGranted then Terminate
```

Typically for each type of licensing policy managed within the policy engine there will be bundles of rules that can be executed, depending on the nature of the licensing type chosen within the SLA.

4 Summary

In looking at SLA management for on demand services, we see a policy engine as key in enabling the formation of client-specific policies as a result of choices a client makes in SLA definition. The policy engine is also key in mapping these, as operational policies set by the provider, to collections of rules that can be used to ensure that conditions associated with SLAs are met. As organizations and users come to rely more and more on digital interactions for services, they will come to expect systems to behave within certain limits, that is, adhere to SLAs. In turn, these systems will need to behave intelligently to meet user expectations as captured in policies, their own, those of providers and others. Policy engines and associated sets rules will be part of that intelligent infrastructure.

Within the SLA environment the potential role for policies and their management engines is much deeper. The policy engine in the autonomic, on demand, SLA is even greater than meeting business goals and objectives [10], it becomes the key to achieving the aims of flexible, dynamic and intelligent management.

References

1. E. Bertino et al. "UCS-Router: a Policy Engine for Enforcing Message Routing Rules in a Universal Communication System", **Proceedings of the Third International Conference on Mobile Data Management** 2002.
2. Buco, M.; Rong Chang; Luan, L.; Ward, C.; Wolf, J.; Yu, P, "Managing eBusiness on demand SLA contracts in business terms using the cross-SLA execution manager SAM", **Sixth International Symposium on Autonomous Decentralized Systems,** 2003. ISADS 2003 pp. 157 -164.
3. CIO.com. (http://www.cio.com/archive) "IBM's New Hook.", **CIO Magazine**, July 1, 2003.
4. N. Damianou, N. Dulay, E. Lupu, and M. Sloman: "The Ponder Specification Language", **Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks,** Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 18-39.
5. H. Lutfiyya, G. Molenkamp, M. Katchabaw, M. Bauer. "Issues in Managing Soft QoS requirements in Distributed Systems Using a Policy-Based Framework", **International Workshop on Policies,** 2001.
6. E. C. Lupu and M. Sloman. "Conflicts in Policy-Based Distributed Systems Ma l. 25, No. 6, 1999, pp. 852-869.
7. "The PONDER Policy management", **IEEE Trans. On Software Engineering**, Vo Based Management Toolkit" at http://www-dse.doc.ic.ac.uk/ August 2002.
8. Y. Snir, et al., "Policy QoS Information Model", Policy **Working Group, IETF** info-model-04.txt.
9. Q. Zhao, Y. Zhou and, M. Perry, "Component Software and Policy-Driven Licensing Model", **Proceedings of Policy 2003 -IEEE 4th International Workshop on Policies for Distributed Systems and Networks,** 2003
10. B. Moore, E. Ellesson, J. Strassner, A. Westerinen "Policy Core Information Model -- Version 1 Specification", **IETF RFC 3060** ftp://ftp.rfc-editor.org/in-notes/rfc3460.txt, February 2001.