Policies are good for business

Francis G. McCabe[*]

August 28, 2004

Abstract

This note addresses the role and promise of declarative policies as a means for bringing large-scale software systems under the control of their owners – the businesses that they are to serve.

1 Introduction

Traditionally, policies and policy frameworks are seen as a way of encoding security constraints – and perhaps quality of service expressed as Service Level Agreements. Certainly these are important applications of explicit policies; however, a greater potential for policies – one that we hope to highlight in this paper – is in the systematic description – and enforcement – of *application* policies and of the exchange of information about policies via Web services.

These policies can be about issues such as privacy, or they can be about issues more closely related to the business carried out – such as delivery preferences and customer credit ratings. Sometimes there will be community agreement – much like standards – about the form and meaning of these policies, other times more personal policies which reflect the requirements of individuals and companies in their use of Web services will be used.

In effect, policies, specifically declarative policies, are not only useful to business but essential to cope with the world of instant services offered across the public Internet.

2 The Web Services Architecture

The W3C Web Services Architecture[1] identifies a policy model as one of its five main models. This prominence is intended to reinforce the central nature of policies in the

[*]Fujitsu Labs of America

architecture; not only are policies a core element needing explication but they also form an integral binding structure relating many other aspects of the architecture.

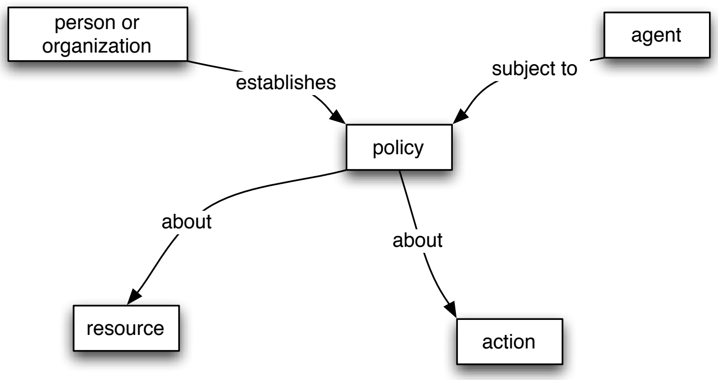
The WSA defines a policy as

> A policy is a constraint on the behavior of agents or people or organizations.

Of particular interest are those policies that can be written down; more precisely that can be processed mechanically.

2.1 Policy framework

The policy model within the WSA is intended to capture the key features of policies that are important to their use in automated systems. The model classifies policies and talks about the relationship between any declarative policy document and the systems governed by the policy.



Policy 101 – from the WSA

The key architectural aspects of policies are that they are constraints, typically about resources and actions, and that policies themselves require enactment – someone has to decide that a policy is to be in force: usually the owner of affected resources decides policy about those resources.

The WSA refers to two main kinds of policies: permission policies and obligation policies.

> A permission is a kind of policy that prescribes the allowed actions and states of an agent and/or resource.

Permission policies are perhaps the most obvious kind of policy as they relate to what a subject is, or is not, allowed to do.

An obligation is a kind of policy that prescribes actions and/or states of an agent and/or resource

Note that policies are not necessarily about actions: they can also concern the permitted *states* of resources. A classic example of this is the constraint that a bank account must be kept in credit: the bank is not concerned with *how* a customer maintains a credit balance – only that the account is always in credit.

Closely associated with policies are *enforcement* mechanisms. In parallel with the two kinds of polices are two types of enforcement: permission guards and audit guards. A permission guard is a mechanism that can be queried to test if a proposed action is permitted; an audit guard is a mechanism that can be used to verify the discharge of obligations. By their nature, obligations cannot be enforced a priori – their discharge can be verified however.

Although very abstract, the WSA policy model captures enough detail to allow the use of policies in Web services to be illustrated. The next necessary step is to develop specifications for policy descriptions that can be widely deployed and enforced.

2.2 Policies within the WSA

The WSA uses policies within the architecture in a number of ways:

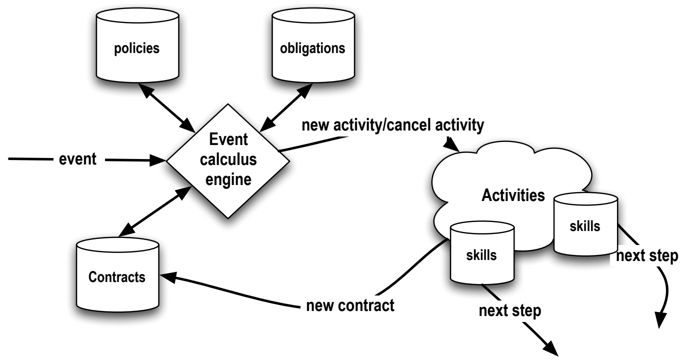
- to express the owning relationship between a Web service and the entities responsible for it,
- to express security constraints,
- to express quality of service issues, such as for message transport, and
- to express the manageability of Web services.

For example, a given service may have a set of policies associated with it. Abstractly these could be about any aspect of the service; however, in practice this is likely to be both about the service being delivered – such as whether an eBusiness service can offer credit terms to a particular customer – and about the means used to deliver the service – such as whether SOAP over SMTP is supported as well as SOAP over HTTP.

2.3 Ingredients of a policy architecture

The WSA does not attempt to explain how policies are implemented – either the details of the kinds of policy descriptions or the mechanisms needed to enforce policies. However, we can surmise that any large scale policy enforcement architecture will share most if not all the following features:

- A base ontology of policy terms – such as permissions, obligations, subject, action, state and so on. This domain independent ontology would have specific elaborations to capture requirements in areas such as security – such as access control lists, file permissions, authentication and so on.
- A language for expressing relationships between policies: such as one policy depending on another, conditional policies, priorities between policies. This language should allow a domain independent *policy interpreter* to be to make *policy decisions* that form the basis of policy enforcement.



A Policy engine

- Policy enforcement mechanisms are likely to fall into several distinct levels: domain independent mechanisms that result in policy decisions of one form or another and domain and application specific mechanisms that ensure that policy decisions are adhered to. An example of the former would be a policy interpreter that can generate policy decisions based on proposed actions or events and a set of policies. An example of the latter mechanism might be the operating system support for access control lists that govern which users can access a given file and ensures that the user does not violate the access permissions to the file.

Implications for existing systems Incorporating policies in applications may well require some re-engineering of those applications. This is especially true in the case of obligation policies where it is important to be able to prove that obligations have been satisfied.

Systems that are able to reason about policies and in particular are able to reason about their obligations has been demonstrated in research systems but is not currently widespread in Industry.

Permission based policies, however, have a long history; originating in various policy-based security environments.

2.4 Business role of policies

Although the WSA focuses on the role of polices within the architecture itself, it is fair to say that the working group imagined that the policy framework would also be applicable to applications: i.e., policies could have a role within Web services themselves.

A key feature of the WSA (inherited from SOAP 1.2) is the structure and processing model for Web service messages. A significant part of this model is the so-called *intermediary*. An intermediary can be thought of as a kind of sub-service – a service functionality with a narrow scope that does not necessarily encompass the entire functionality of a service. Examples of intermediaries include security enforcement mechanisms and application-level mechanisms.

One example of an application level intermediary might be a customer information sub-service – that is capable of examining any SOAP message that has customer information and verifying that the referenced information is known to the service provider. In practice there are likely to be many varieties of customer information and also of customers themselves. Furthermore, the extent to which the service trusts the customer and even the service itself it quite likely to vary from customer to customer.

Such variety is best managed using explicit policies that express the constraints on the service provided depending on the characteristics of the customer. These policies themselves can be managed by the owning controllers of the Web service.

Fundamentally, explicit declarative policies can enable the *owners* of Web services to control the business semantics of the Web service. The greater the inherent complexity and variability of an application – whether a Web service or any other application – the greater the potential for explicit policies to enable businesses to control their Web services.

3 Policies on the Web

We see a significant role for explicit policies in the *deployment* of Web services and in the *use* of Web services. In managing the security aspects, the manageability aspects and other meta-level aspects of services, policies and standard specifications for policies will greatly aid in Web service deployment. Furthermore, the use of policies within Web services has the possibility of greatly enhancing businesses' control over the functionality of Web services – including the possibility for *exchanging* policy documents and *forwarding* policy documents.

In practical terms policies will be used for a variety of system-level and application-level aspects. For example, a Web service provider may have policies about how authentication tokens such as X.509 certificates should be present in SOAP messages. Conversely a client of a Web service may wish to interact with the privacy policies of the provider – perhaps by querying the P3P policies published by the provider linked from a WSDL document. These two case represent two distinct cases in policy application: one imposed by the provider and one negotiated between the provider and requester of the service.

3.1 Security policies

In the classic SOAP 1.2 model, the role of a header is to signal – in a standardized way – some particular aspect of the message. In the case of a X.509 certificate it is a means of authenticating the subject of the message (typically).

Given a policy engine, and given an architecture that is capable of processing SOAP messages using intermediaries (service roles in the parlance of the WSA), a service provider can enforce a requirement that customers are authenticated by requiring that incoming messages bear the appropriate X.509 certificates – within WS-Security specified headers. This might be enforced with a combination of a software agent that is actually capable of processing the certificate, a policy engine that is capable of requesting that that service role be applied and a policy document that determines the requirement.

Essentially, this is a straightforward application of business logic to the processing of incoming messages. The only aspect that is novel is the fact that the policies expressing the constraints are expressed in a standard notation.

3.2 Privacy policies

On the other hand, the processing of a privacy policy is necessarily quite different to authentication processing. The reason is that privacy involves a negotiation and, in the event that the Service provider may have service relationships with other Service providers, may involve a *transitive application* of the Service requester's policies.

For example, in Europe, the various Data Protection Acts require that certain personal information may only be collected in certain registered databases. Furthermore, any organization that collects personal information may only transmit it to other organizations that are similarly registered.

A customer wishing to interact with a service provider may wish to query the Service provider about the provider's policies. This information may be published by the provider in a searchable registry, or it may be available more directly by querying the provider – in effect making it part of the Service interface.

Once a customer agent decides to interact with the service, it may wish to impose its privacy policy on the provider – for example by requiring that the provider does not forward certain personal details to third parties.

The policy can be declared by the Service requester agent in a similar manner to the authentication policy – by using a standardized header encoding a P3P policy (for example). In addition, however, is an implied *obligation* on the part of the Service provider to respect the privacy policy.

As noted above, obligations can not often be enforced a priori. However, they can potentially be *audited*; i.e., evidence may be given that the obligations are properly *discharged*. In this case, part of the policy negotiation may be a commitment to communicate back to the Service requester sufficient information to prove that the Service provider has abided by the privacy policy – perhaps by copying SOAP messages sent to third parties back to the originating Service requester or by identifying an escrow service that the requester agent may query at a later date. For example, the message sent by a book store service to a delivery service should not, and need not, contain the credit card details of the customer. Showing the messages sent is evidence that you are abiding by such a constraint.

It is in the nature of privacy policies that it is hard to prove that you have not violated them – however, it may be in your interest to attempt to give evidence of your good faith to the requester by copying messages or digests of messages you send to help fulfill the request.

4 Very large scale systems

In assuming that policies are explicitly written down we implicitly introduce a new class of document into the world: policy documents. Managing large collections of these documents is similar to managing large collections of any kind of document; however, policy documents introduce some special considerations.

A policy is always *enacted* by *someone* – i.e., the policy document is a record of the constraints that someone has decided should be applied. In managing large sets of policy documents it is important to keep track of the provenance of the enacted policies.

There are two senses in which Web services may give rise to large complex collections of policy documents: large numbers of individuals with their own sets of policy documents and complex enterprise systems with associated policy networks. These two extremes require different approaches.

Where an individual – or small enterprise – has policies, they are likely to be relatively few in number. Moreover, more importantly, the provenance of individual policy documents is clear: they are originated and enacted by the individual or business owner. The fact that there are large numbers of individuals is not important except where individuals exchange policy documents with others. Provenance reduces, in such circumstances, to clearly identifying the enactor of a given policy recorded in a policy document. That, in turn may require that the policy specification language requires this level of documentation.

In the case of a more complex enterprise class system, there may be many thousands of policy documents within one system. Furthermore, two additional factors arise: the provenance of a given policy document is likely to be much more complex and the ratio of system administrators to documents is likely to be much lower.

For example, the policies that apply to particular e-commerce transactions may originate from many stakeholders within a company: the sales and marketing team, the fiduciary team and the executives of the company. Furthermore, a given policy is much more likely to be enacted in a role-based manner. For example, the discount applied to customers in good standing is likely to be enacted by the head of the sales division – acting as that head – rather than by P. J. Seller – acting on his own behalf – even if P. J. Seller is currently the head of the sales division.

This latter complexity of policy management requires approaches to managing documents that is not currently very common in business. It requires that given documents have clear meta-documentation as to their origin. It also requires some discipline in the process of establishing policies; which in turn requires some modeling of the organizations themselves – the powers and authorities that officers of the organization have. In effect, it may become necessary to be able to explicitly model *trust* and *authority* and to reference these from within policy documents.

5 Conclusion

Policies represent a cornerstone of the Web services architecture. The policy framework explicates the role of policies in the deployment of Web services. However, a greater potential lies in controlling the functionality of Web services using a policy-based framework. In addition, permitting the exchange of policies opens the possibility of constraint-based interactions between Web services agents.

Finally, it should not be forgotten that policies introduce a new class of document into the world – with associated complexities in managing those documents.

References

[1] Web services Architecture Working Group. Web services architecture, 2004.