Policy Support for Business-oriented Web Service Management

Stephen Gorton and Stephan Reiff-Marganiec

Department of Computer Science
University of Leicester
University Road,
Leicester LE1 7RH
United Kingdom
Email: {smg24, srm13}@le.ac.uk

Abstract

Policies have been adopted for many reasons within web services and Service-oriented Architecture in general. However, while they are a favoured method of management, this only occurs at the service level and in the software domain. Policies already exist in a narrow variety more focussed on service properties such as authorisation. As a significant number of web services become available, more emphasis needs to be placed on management of services in the business domain. In this paper, we propose a policy framework that can be used to express business requirements for web services, at a business level that is more abstract than the current high-level composition and orchestration technologies.

1. Introduction

The deployment of software as reusable services available on a network is the core idea behind the loosely-coupled, open standards-based Service-oriented Architecture (SoA), of which Web Services [1] is an implementation. Whilst still a relatively young technology, Casati *et al* [4] note that as a substantial number of Web Services become available, so the attention shift will be from service infrastructure (i.e. discovery, description and composition) to service management.

Often, the enterprise architectural layer of web services is regarded as a mangement layer, with technologies such as BPEL[1] providing a means to composition and orchestration of composite services. However, approaches such as BPEL are based upon a technical perspective which is of little use in a business domain where the primary user is a business analyst. It also assumes that the underlying business process is pre-defined, despite the fact that business needs include adaptation to technological and political changes. Instead, management of web services should be from a business perspective, focussing on the agility and adaptability of services that can fulfil business requirements. In addition, there is an increasing requirement to align IT objectives with business objectives. This need has been recognised by industry and a recent report ("IBM has high hopes for 'Next Big Thing' in software", Financial Times Online, April 3, 2006) reported that IBM has doubled its business in SoA and stated that "Things really rub on each other - it's [SoA] the intersection of technology and business".

Our approach is to use policies to manage services, in terms of business requirements. Policies are defined in the business domain and express what a service should do. In addition, policies define sequences of events and responses to specific activities. In [6], we presented a context in which policies are used. This paper describes policies in more detail, including their structure and possible implementation with APPEL [12], a policy description language designed for the telecommunications environment. APPEL is made available to end-users through the use of wizards.

In section 2, we give a general overview of SoA, together with policies and their current uses. In section 3, we describe the adapted APPEL framework with respect to web services. In section 4, we present an initial APPEL implementation for SoA, before concluding section 5.

2. Background

SoA, and its implementation as Web Services, provides an opportunity to achieve dynamic applications through automated discovery and composition of services. Services are deployed and made available with well-defined interfaces, so that the implementation details are hidden. This

[1] http://www-128.ibm.com/developerworks/library/specification/ws-bpel

is suitable for the business domain, where the required perspective is a complete picture of the external quality of the interactions, as perceived by the customers [4].

Policies are defined as "...information which can be used to modify the behaviour of a system" [8], without the need for re-compiling or re-deploying. Thus we consider policies as loosely coupled with the systems they interact with. Furthermore, we refine the definition of a policy in the context of our web service management system as *"a high level statement as to how business requirements should be processed in the management system"* (refined from [9]).

The most accepted policy type to be implemented with web services is the access control rule. Ponder [5] provides a framework for specifying authorisation, obligation and delegation properties for allowing specific activities, forcing some event or specifying a delegate, respectively. Other policy approaches include KAoS [3], which express constraints on allowable actions by particular subjects, and Rein [7], which expresses constraints over resources such as services and actions. WSPL [2] is another access control policy language, based on XACML[2].

The Web Services Policy Framework (WS-Policy) expresses capabilities and constraints of a particular web service, in conjunction with various other specifications such as WS-PolicyAttachment and WS-ReliableMessaging. These specifications[3] have service-specific applications in the lower layers of the web service stack.

Thus we can see that policies are a popular approach for many aspects of web services. Though despite the number of uses, policies have yet to be applied to a business management framework. Our work is aimed at developing a business policy framework for the management of web services in the business domain.

The use of policies that we propose is orthogonal to a graphical modelling language, such as the one presented in [6] or similar to UML activity diagrams. Each task within a task map represents a unit of business activity that contributes to the satisfaction of the wider business goal. Tasks are encoded with policies and may be subjected to external policy inputs, or global policies. The notation includes operators that allow tasks to be carried out sequentially and in parallel, whilst addressing synchronisation issues.

3. APPEL Policy Framework

3.1. Overview

APPEL is a generic policy description language created for the telephony call control domain [11]. It was used for specifying policies. It is defined by an XML grammar, enabling the use of many common tools and parsers. We believe there are similarities between telecommunications and SoC, thus by specialising APPEL to the service domain, we can take advantage of its current features.

Due to the nature of changing requirements, particularly in the telecommunications domain with switchable features such as call forwarding, it was important to have some control mechanism over call features in such a way that core software did not need changing together with recompiling and redeployment. Web services are similar in that we often need to combine more than one together in order to achieve the results we desire. One specific advantage of SoA is service level reuse, where services are loosely coupled to their clients and can be invoked many times over. In a similar way, web services have the potential to satisfy our software requirements through the use of composition when (atomic) services are not able to individually. One should note at this point that APPEL and other policy languages are *policy description languages* (PDL), and policies are those documents borne out of implementing rules using a PDL.

3.2. APPEL Description

APPEL [11] is a generic policy description language developed for use in telecommunications through language specialisation. More specifically, it is used to specify policies that govern call control with respect to call features such as call forwarding and call barring. APPEL is defined by an XML schema, although a wizard was created for the initial implementation to increase expressive power in the end user domain and allow non-software experts a simpler method of defining their own policies.

A basic APPEL policy defines a number of policy rules, each of which specifies a set of triggers, a set of conditions and a set of actions. The first two sets may be empty, but the last cannot be, thus APPEL has the ability to express ECA rules and user goals.

The policy itself has an owner and may be applied to a user, a set of users or a domain (identified through email-like addresses). Policies can specify modalities through the preferences **must**, **should** and **prefer**, plus their negations. No specification of preference would indicate that the user was neutral about a subject.

Policies are interpreted by a policy server, implying that they have little use other than specification at the end user point. Policy servers are able to link to policy stores that contain information required for policy processing, such as protocol to policy terminology mapping.

3.3. Extension Actions

We extend the initial implementation of APPEL by introducing the following (informally described) actions:

[2] http://www.oasis-open.org/committees/xacml/

[3] located at http://www-128.ibm.com/developerworks/webservices/

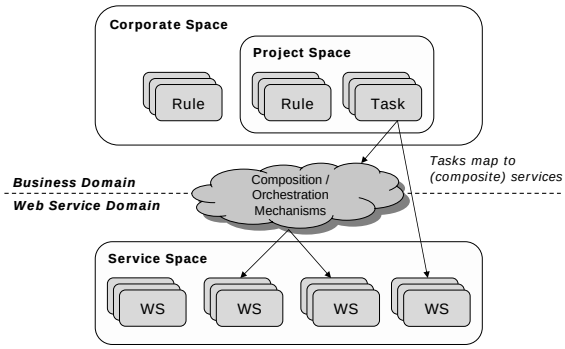

Figure 1. Cross-domain system layout

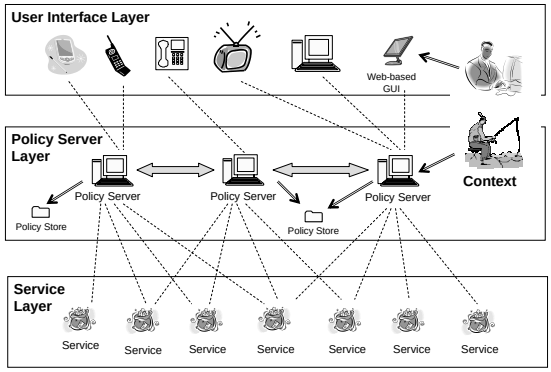

Figure 2. Policy architecture

Prompt: allows the user to enter parameters directly. The action `prompt(String dataName)` asks the user for a value that refers to the parameter `dataName`.
Display: outputs a result or query as specified by the user. The action `display(Data data)` displays service response data, as either a list or a singular object. The action `display_exception(Exception exception)` outputs exception messages to the user, and the action `display_empty()` outputs a default message in the case of an empty response.

3.4. Management System Architecture

The context of our research is shown in Fig. 1. We use policies to drive composition by encoding requirements and rules, more specifically as the details of tasks which themselves are located in the corporate space, constrained by individual task rules and overarching corporate rules.

In Fig. 2 (adapted from [10]) , the bottom service layer represents the original web service stack (including messaging, discovery and description). We propose an intuitive addition of layers to address policies and policy interfaces (e.g. wizards), since we are concerned with increasing the level of abstraction towards and into the business domain.

The policy server layer contains the policy management systems, together with web service search and matching engines. There may be repositories which can store persistent policies as required by policy servers, who can also share repositories. Many policy servers may exist, depending on demand and processing ability. Other aspects, such as mediation and negotiation, can also be addressed in this layer.

The user interface layer allows business users to create new policies and activate their business requirement model. Due to the diversity of services that can exist, we must allow for a number of different interfaces, for example PDAs, PCs, mobile phones and televisions. The interface should include wizards to aid in the creation of policies as we do not expect business users to be familiar with low-level code. Wizards may be customised to their particular platform.

4. Language Specialisation for Web Services

4.1. Events

An event can be defined as either a message being passed in a system (as in event-driven programming) or a change of properties (as in telecommunications). A specified event acts as a trigger to a policy. In the context of web service management, we define simple events to be:

- *Message events*: occur when a message is sent or received. A message includes a record of its source and destination, plus a description of the message and the actual message data.
- *Time events*: are either absolute (prescribed), periodic (regular periods or relative (absolute time calculated according to some criteria, e.g. start and end times). They are modelled by standard XML timestamps where possible.
- *Change events*: occur when system properties change, originating from either an internal or external source.
- *Service events*: are generated before a service is invoked or afterwards (potentially before or after a response is received). These events are based on the afore-mentioned message events.
- *Interaction events*: occur depending on what type of service operation has been invoked. If the operation returns a response, a call event is generated after receiving the reply. If the operation does not return a response, then a signal event is generated on invocation.

Complex events can be built from simple events using event composition primitives (e.g. conjunction), resulting in event (or trigger) groups.

4.2. Conditions

Conditions are boolean values that must equate to `true` when the policy is triggered in order for the action section

to be invoked. It is either a simple parameter value check or a condition group using condition composition primitives. Parameter values may be subject to operators such as $>, \geq$, $<$ or $\leq$. Parameters are generally local variables that may declared in the policy, or the wider policy system.

4.3. Actions

An action is a step towards fulfilling a business goal. In the context of web services, an action may include the invocation of a service. Actions may be atomic or composite (action group) through the use of action composition primitives. These primitives include `and`, `or`, `andthen` and `else` as defined by the APPEL language.

5. Conclusion

Policies are already in use for a wide variety of applictions, specific to the management of distributed systems. Futhermore, policies are often used in SoA to define access control constraints on services. We have presented an event-condition-action policy framework, derived from the APPEL policy language, to lift policies from the service domain into the business domain.

Our work is novel in that policies are used only in the service domain thus far, rather than in the business domain. The latter domain requires more of a business perspective and the input of business metrics, such as quality. It also requires agility in order to continually align corporate IT objectives with key business objectives.

We have concentrated on the policy description language here, but the framework includes an architecture that allows for policies to be used in the selection of suitable services and also to govern service execution.

The gain achieved by our work is that policies are a powerful and flexible means of management in the face of dynamic environments, such as service composition. While services may change through updates, removals, etc., policies can continue to express the constraints as required by the end user. Therefore, the potentially long process of discovering and composing the "ideal" service for a requirement is significantly reduced. APPEL, although a generic policy framework, can be specialised to the service domain, as it was with telecommunications. It allows us to express a range of policies such as goals and ECAs, whilst using XML as a widely-accepted interchange format.

Our further work involves improving the language specialisation for APPEL by defining more accurately the extended functions as previously described. We also plan to add operational semantics to the language in an effort to formalise it further. Finally, in conjunction with a business modelling notation, we will look to integrate a system that combines graphical modelling with underlying policies.

Acknowledgements

This work is funded by the IST-FET IST-2005-16004 project SENSORIA (Software Engineering for Service-Oriented Overlay Computers).

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraiu. *Web Services: Concepts, Architectures and Applications*. Springer, 2004.

[2] A. H. Anderson. An introduction to the web services policy language (WSPL). In *POLICY*, pages 189–192. IEEE Computer Society, 2004.

[3] J. M. Bradshaw, A. Uszok, R. Jeffers, N. Suri, P. J. Hayes, M. H. Burstein, A. Acquisti, B. Benyo, M. R. Breedy, M. M. Carvalho, D. J. Diller, M. Johnson, S. Kulkarni, J. Lott, M. Sierhuis, and R. van Hoof. Representation and reasoning for DAML-based policy and domain services in KAoS and nomads. In *AAMAS*, pages 835–842. ACM, 2003.

[4] F. Casati, E. Shan, U. Dayal, and M.-C. Shan. Business-oriented management of web services. *Commun. ACM*, 46(10):55–60, 2003.

[5] N. Dulay, N. Damianou, E. Lupu, and M. Sloman. A policy language for the management of distributed agents. In M. Wooldridge, G. Weiß, and P. Ciancarini, editors, *AOSE*, volume 2222 of *Lecture Notes in Computer Science*, pages 84–100. Springer, 2001.

[6] S. Gorton and S. Reiff-Marganiec. Towards a task-oriented, policy-driven business requirements specification for web services. In S. Dustdar, J. Fiadeiro, and A. P. Sheth, editors, *BPM*, Lecture Notes in Computer Science. Springer, 2006. To appear.

[7] L. Kagal and T. Berners-Lee. REIN: Where policies meet rules in the semantic web. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, 2003.

[8] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Trans. Software Eng.*, 25(6):852–869, 1999.

[9] S. Reiff-Marganiec and K. J. Turner. Use of logic to describe enhanced communications services. In D. Peled and M. Y. Vardi, editors, *FORTE*, volume 2529 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2002.

[10] S. Reiff-Marganiec and K. J. Turner. A policy architecture for enhancing and controlling features. In D. Amyot and L. Logrippo, editors, *FIW*, pages 239–246. IOS Press, 2003.

[11] S. Reiff-Marganiec, K. J. Turner, and L. Blair. APPEL: The ACCENT policy environment/language. Technical Report CSM-164, University of Stirling, Jun 2005.

[12] K. J. Turner, S. Reiff-Marganiec, L. Blair, J. Pang, T. Gray, P. Perry, and J. Ireland. Policy support for call control. *Computer Standards and Interfaces*, August 2005.