Unleashing the Power of Policies for Service-Oriented Computing

Carlos Kamienski[¥], Ramide Dantas[£], Ernani Azevedo[£],
Cyrus Dias[£], Djamel Sadok[£], Börje Ohlman[§]
[¥]*Universidade Federal do ABC, Brazil, cak@ufabc.edu.br*
[£]*Universidade Federal de Pernambuco, Brazil, {ramide,ernani,cyrus,jamel}@gprt.ufpe.br*
[§]*Ericsson Research, Sweden (Borje.Ohlman@ericsson.com)*

Abstract

Policies have the potential of playing a key role in Service Oriented Computing (SOC) providing many useful features for service creation, execution and management. However, so far the Policy-Based Management area has had a limited adoption and the result is that its advocated benefits still remain unproven in a broader sense. In this paper we show that the integration of policies with services using the Service Refinement Cycle can unleash their power of providing higher levels of adaptation to changing environment conditions and system requirements. We present different uses of policies for SOC and an example scenario.

1 Introduction

The Service Oriented Computing (SOC) paradigm models providers and consumers of computing facilities as services that may be composed together for creating new higher-level value-added services. In order to leverage this new paradigm a new breed of flexible composed services must have the ability to adapt dynamically to fast requirement changes and different environment conditions.

Policy-Based Management (PBM) aims at simplifying the administration of networks and services by establishing policies to deal with typical situations in an automated fashion [11]. Also, policies present the benefit of supporting dynamic adaptability of behavior without recoding or stopping the system [10], thus may play a key role in SOC by providing many useful features for service creation, execution and management. However, so far the PBM area has had a limited adoption and the result is that its advocated benefits still remain unproven. Even though the whole idea of PBM involves the deployment of an external system for isolating policies from the systems they are governing, we think this separation of concerns has so far hindered the adoption of policies in the mainstream of software (and service) development.

In this paper we show that the integration of policies inside services with the Service Refinement Cycle (SRC) [4] can play a key role for unleashing the potential of higher system adaptability. We advocate that the benefit of supporting dynamic adaptability of behavior without recoding or stopping the system of policies can be extended to services with minimal effort with the SRC.

We propose a new classification of policies in generations where our proposal falls within the third generation which seamlessly integrates policies and services under the SRC. Also, we provide examples of different applications of policies within services in order to obtain a synergic effect to leverage the use of SOC. Finally, we highlight the role of policies in providing autonomic features to services with a full and seamless integration between managed service and autonomic manager, which may be developed using the same concepts and technologies.

The rest of the paper is structured as follows. Section 2 presents background and related work. Section 3 our classification of policy systems in three generations and section 4 presents the Service Refinement Cycle. Examples of different applications for policies within the SRC are provided in section 5 and an example of modeling a service with the SRC and the flexibility provided by policies is shown in section 6 and finally section 7 draws conclusions and topics for future work

2 Background and Related Work

Service Oriented Computing is a paradigm that employs "services" as the basic unit for the separation of concerns in the development of computing systems [3]. Services are usually composed together to create other services [3]. On the other hand, policy refinement is usually needed when dealing with policies, where high-level a business-oriented goal is mapped to one or more, low-level enforceable policies [2].

Although policies are intrinsically related to services and the problems of policy refinement and service composition are similar, proposals that deal with both problems together are not common. In this paper a key concept is service refinement, a combination of service composition and policy refinement, where a higher

level service specification may be refined into an integrated composition of lower level services.

The integration of policies with services (or alternatively Policy-Based Management with SOA) is not a new idea by itself though. The authors of [9] analyze five known policy frameworks that may be used together with SOA systems: IETF, Ponder, KaOS, Rei and WS-Policy. They conclude that none of them is adequate for the purpose, because all of them lack some feature. However, [9] considered the use of policies for the management of SOA and the integration of policies and services is not considered. In other words, authors consider policies as being separated from services and not integrated like we propose here. Also, WS-Policy [1] may be considered related to our work but it is very narrow in scope and therefore not adequate for a complete comparison.

3 Policy Generations

We divide policies into three generations. Our proposal represents the third generation which seamlessly integrates policies within the services.

The first generation of policies in IT systems consists of static rules inside systems aimed at providing some level of flexibility through parameterization. In early IT systems and network infrastructures, policies have been largely used for expressing rules and constraints, represented as parameters stored in system tables. Although they provided some basic levels of flexibility, they were static and limited in nature, because the number of configuration parameters a given system accepted was defined at design time. As a matter of fact most systems still today are in the Generation 1.

In the beginning of the 1990s, PBM has been proposed as a separate module of the original systems, thus providing an easy way of dynamically adding, removing and updating policies stored in an external database. A PEP located in the computing or network system sends a request to an external PDP that deals with policy selection and processing, rendering a decision that is sent back to the PEP. Since the PDP is an external component, given that enough "hooks" (PEP requests) are located in strategic places, policy could be easily adapted on the fly.

We argue that services and policies should be integrated to create a new breed of Flexible Composed Services. In the third generation, policies are intended to provide extended adaptability features, as well as to help service composition and even to become an alternative to develop self-managed services. This approach tries to reconcile the use of policies and services together, by integrating them in a single system and providing a flexible service specification format and a service execution environment.

4 Service Refinement Cycle

The Service Refinement Cycle (SRC) [4] is an integrated abstract framework for dealing with the service life cycle, which defines a common "language" for specifying and understanding service composition. It is composed of a sequence of phases, where each phase adds some features for enhancing the capability of services to adapt behavior according to different requirements. The four phases are:

- Service: Contains a standardized service description following the SOA paradigm that assumes that services must be widely advertised.
- Process: Complex services are usually specified as a composition of other services, which always follows a process. The process phase is composed of one or more abstract tasks (not real services yet), which interact with each other in certain ways.
- Policy: Each task of the process phase may (or may not) to be associated to one of more policies, aimed at providing flexibility to composed services to adapt behavior on demand.
- Binding: The binding phase provides isolation from internal phases (service, process and policy) to the outside world, i.e., it maps internal abstract services to external real ones.

Transitions specify links between phases which are designed during service creation and followed by an engine at execution time.

5 Uses of Policies in the SRC

5.1 Multiple Policy Scopes

The most common way of integrating policies with services is by calling the Policy Phase in the implementation of a service function. However, a slightly different approach must be taken to deal with default policies that must be enforced for the whole service, i.e. all functions of the service API, or for a SEE, i.e. all functions of all services. Using the SRC they can be enforced by chaining requests through the process phase and thus creating a nested policy enforcement. In Figure 1 a particular deployment of nested policy enforcement is presented in different scopes: function level, service level and Service Execution Environment (SEE) level. In this example, the first step of the process phase for the service function *processOrder* makes a request to function *enforceServicePolicies* through its binding phase.

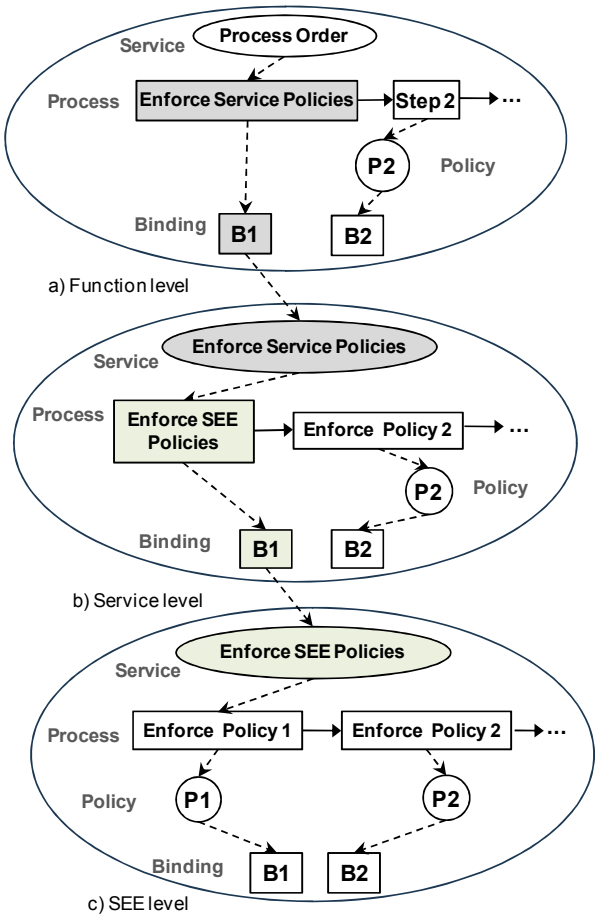

Figure 1 – Nested policy enforcement in different scopes: function level, service level and SEE level

5.2 Policies for Access Control

Granting or denying authorization for users to access system resources is the most common application of the PBM technology. Service-oriented computing also needs access control features and the SRC may be used to develop services that fulfill this requirement. The simplest solution to enforce authorization policies using the SRC is to add one or more steps at the beginning of the Process Phase for requesting those policies. Also, in addition to policy enforcement, service consumers may be provided information about access control before they decide whether to use services in a composition. The OASIS reference model for SOA suggests that service descriptions should add "support for associating policies with a service and providing necessary information for prospective consumers to evaluate if a service will act in a manner consistent with the consumer's constraints" [7].

5.3 Providing Information to Policies

An action as simple as a PEP sending a request to a PDP to select a policy and rendering a decision may be complicated at times. It involves a variety of different conditions which may need to query different data sources. In the OASIS architecture for XACML there is a special component called Policy Information Point (PIP) [8], aimed at obtaining additional information so that to the original request this information may be added and the policy can be processed by the PDP.

Figure 2 presents an alternative model, where the phases of the SRC are used as architectural components for obtaining additional attributes that policies may need. Instead of using a different entity such as the PIP, the SRC-based model request additional information directly to services. The main difference between these models is that XACML needs new modules to be added to its architectural model, whereas using the SRC the same concepts are applied again and again, for dealing with the same situation.

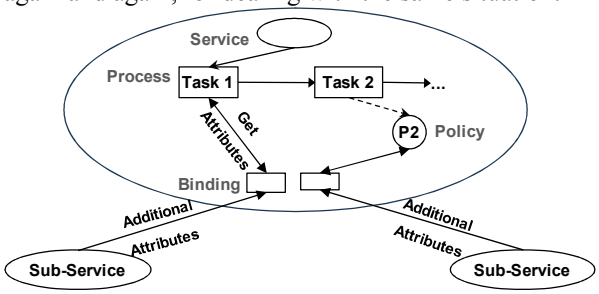

Figure 2 – Obtaining additional attributes using the SRC model

5.4 Flexible Execution of Composed Services

Policies in the SRC have "extended action power", in the sense that they can invoke services for performing whatever actions they are intended to, in addition to those expressed in the limited *Effect* keyword of XACML. Services may be invoked through the *Binding* keyword that may be used inside the *Condition* and *Processing* keywords as depicted by Figure 3.

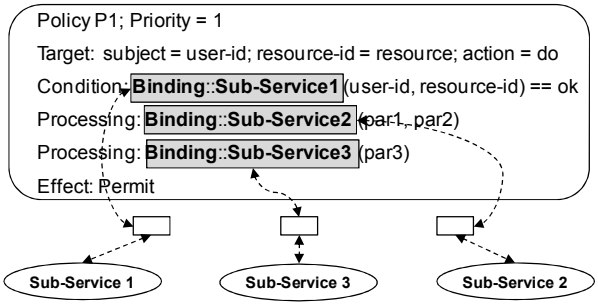

Figure 3 – Using policies for providing flexibility in the execution of composed services

Binding is used for calling the Binding Phase of the SRC. *Condition* is an auxiliary way of selecting policies (the primary one is *Target*) and contains an expression that evaluates to "True", "False", or "Indeterminate". *Processing* clauses provide enhanced action capabilities for policies and are executed whenever a policy is selected for execution [5].

5.5 Policies for Autonomic Services

Autonomic Computing is all about adapting behavior due to changing requirements and environment conditions. Also, an autonomic system needs an Autonomic Manager that continuously monitors Managed Elements for taking decisions about changing behavior when needed. The SRC may be used to build autonomic services where managed service and autonomic manager are fully integrated as suggested by Kephart and Chess [6]. Figure 4 depicts this approach, where both Managed Service and the Autonomic Manager Service are seamlessly integrated inside a single service using the SRC. In a SRC-based Architecture, autonomic services may be based on high-level policies and goals. A high-level Autonomic Manager may be fed in with high-level policies which may be refined into lower level services with technology oriented policies that will monitor and control adaptation in the managed service.

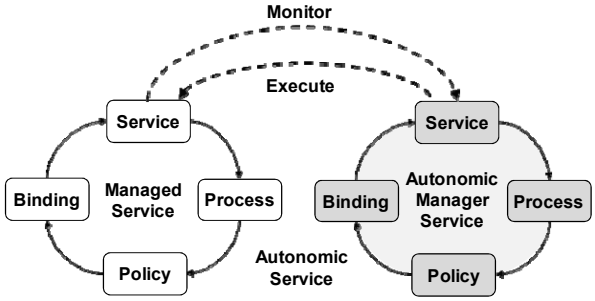

Figure 4 –Architecture for autonomic services with full integration between Managed Service and Autonomic Manager

6 Virtual Flower Scenario

We model a Virtual Flower (VF) scenario focused in the policy phase, due to space constraints, which allows customers send virtual flower bouquets to another person via a MMS message. Our simplified modeling for the VF scenario is composed of three services: Billing, Info and Messaging. Table 1 presents the description for the sendVFlower function, which allows sending virtual flowers, taking some parameters and at the end calling the Process Phase.

A simplified process for the sendVFlower is depicted Table 2. From the three tasks, only the first, Charge-User calls the Policy Phase.

Table 1 – Virtual Flower Service Description

Function	Parameters	Next
sendVFlower	flower-id, dest-number, source-number, user-id, payment-method	Process

Table 2 – Process Tasks for sendVFlower

#	Name	Parameters	Next	Service
1	Charge-User	uid, pay-meth	Policy	Billing
2	Get-Picture	flr-id	Binding	Info
3	Send-MMS	flr-img, dst-nr, src-nr	Binding	Messaging

Figure 5 shows the policy that is selected when task Charge-User from the Process Phase calls the Policy Phase, represented in XACML-like syntax [8]. It calls the Binding phase, which invokes an external service.

Policy 1: **Priority** = 1
Target: subject = any-user; **resource** = any-flower **action** = charge
Condition: Binding::chargeUser(uid, pay-meth) == ok
Effect: Permit

Figure 5 – Policy for sendVFlower

Now we introduce a new requirement for showing adaptability provided by policies. The VF provider wants to create a prepaid system and a fidelity program. In order to deal with the brand new class of prepaid registered users, a new policy must be created, which is policy P2 in Figure 6. Now the subject is the group prepaid-user, whereas in Figure 5 it is any-user. Policy P2 was given priority 2, in order to force it to be selected over policy P1 (priority 2 is higher than 1). The result is that when the customer is a registered prepaid user, policy P2 will be selected and a different billing method will be used. This policy also credits the bonus to the user account in the flower shop fidelity program. As a result, two new services have been created, Prepaid Service and Fidelity-Program Service.

Policy P2; **Priority** = 2
Target: subject = prepaid; **resource** = any-flower **action** = charge
Condition: Binding::debitUser(uid, price) == ok
Processing: Binding::creditBonus(uid, price) == ok
Effect: Permit

Figure 6 – Policy for the new sendVFlower

7 Conclusions

We believe that the advocated benefits of PBM are yet to be unfolded and tested at large by the community. We have shown that the integration of policies with services can be done by the use of the SRC, which makes possible different uses of policies in the future.

8 Acknowledgements

Work supported by the Research and Development Centre, Ericsson Telecomunicações S.A., Brazil.

References

[1] Bajaj, S. et al., “Web Services Policy 1.2 - Framework (WS-Policy)”, W3C Member Submission, April 2006.

[2] Bandara, A. K., Lupu, E. C., Moffett, J., Russo, A. "A Goal Based Approach to Policy Refinement", IEEE Policy 2004, June 2004.

[3] Erl, T., “SOA Principles of Service Design”, Prentice Hall, 2007.

[4] Kamienski, C., Dantas, R., Fidalgo, J., Sadok, D., Ohlman, B., “Service creation and execution with the Service Refinement Cycle”, NOMS 2010, April 2010.

[5] Kamienski, C., Fidalgo, J., Dantas, R., Sadok, D., Ohlman, B., “XACML-Based Composition Policies for Ambient Networks”, IEEE Policy 2007, June 2007.

[6] Kephart, J. & Chess, D., “The Vision of Autonomic Computing”, IEEE Computer Magazine, January 2003.

[7] MacKenzie, C. M. et al: “Reference Model for Service Oriented Architecture – Version 1.0”, OASIS, 2006.

[8] Parducci, B., Lockhart, H., Levinson, R., McRae, M.: “eXtensible Access Control Markup Language – Version 2.0”, OASIS Standard, 2005.

[9] Phan, T. et al., “A Survey of Policy-Based Management Approaches for Service Oriented Systems”, 19th Australian Conf. Software Eng.(ASWEC 2008), 2008.

[10] Sloman, M. & Lupu, E., “Security and Management Policy Specification”, IEEE Network, 2002.

[11] Verma, D. C., “Simplifying Network Administration using Policy-Based Management”, IEEE Network, March/April 2002.