Policy Based Messaging Framework

Martin Eggenberger[1,2], Nupur Prakash[2], Koji Matsumoto[2], and Darrell Thurmond[3]

[1] SpinergyGroup, Piedmont, CA, USA
[2] Delta Dental of California, San Francisco, CA, USA
[3] KoolKode Technologies, LLC, Santa Monica, CA, USA
martin@spinergygroup.com, nprakash@delta.org, kmatsumoto@delta.org, koolkode@architect-alchemist.com

Abstract. Due to integration complexities to legacy as well as new systems, a Common Messaging Framework has been developed that is based on policies to control the behavior of the various enterprise services. These policies include both internal and external Quality of Service Policies as well as constraint based business process policies. This paper proposes and identifies a policy based messaging framework for both intranet and extranet services, upon which individual policies can be injected during runtime for individual messages, domains and or processes. Further more these policies can be customized on a per actor basis and dynamically changed during runtime by a console user without having to stop the process.

Keywords: Service Oriented Architecture, QoS, Policy, Dependency Injection, Adaptive Services, Ontologies, Queuing.

1 Introduction

Although there has been considerable attention been devoted in both industry and academia to the design and implementation of new services, little headway has been made to enable legacy systems to truly take advantage of a Service Oriented Architecture. Specifically, non–functional requirements within the Quality of Service (QoS) arena need to be further researched. In essence we found three problems associated with legacy integration using SOA.

First off, most legacy integrations are built using Point to Point integration solutions. Most large scale organizations use batch processes and batch transfers to exchange data between various point solutions and the primary communication channel is file based. Since the individual records in these files do not contain QoS policies and the rewriting of the code is not feasible, no policy enforcement is feasible.

Secondly, the error handling of legacy applications and processes are using different solutions such as log files, databases and simple process return codes. Since these applications were built over the last 20 years, we are faced with various problems in the application logging/monitoring and auditing policies. Specifically, the auditing policies have changed over the years; and therefore, we

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 497–505, 2007.

require an adaptive policy system to adjust to the changing regulatory requirements.

And lastly, the process orchestration used is mostly based on scheduling technology [1,2]; and therefore, only temporal properties are used for process orchestration. The nature of this orchestration limits the introduction of QoS policies, hence a new event driven processing mechanism was explored that enable policies for legacy and new systems.

To address these problems, we have developed a policy based messaging framework that support QoS policies. Our approach is based on a comprehensive messaging model for description, discovery, policy injection and policy enactment that are suited for a Service Oriented Architecture [3]. The messaging model defines a semantic model of the messages' purpose as well as the policy associated within the semantic model. To that end, a message consists of a set of processing instructions related to the domain and process it is used in, as well as a set of policies that are related to the domain, the process or the message itself. Further more we described the relationship between the caller's context (e.g. security context) and the associated policies. For example, a system user may define an Auditing Policy based on a specific computing domain such as Claim Processing. In the above example, the system user requesting such a service would specify what elements within the message have to be auditable.

Given such a description framework, we also required a message discovery framework [4,5] that allows us to apply and inject domain and process information into the individual message. To that end, we developed and implemented a domain and process ontology, that is used as the basis for domain and process discovery purposes. Having obtained the messages' domain and process, the policy set can be injected given the callers credentials.

Since all user and system credentials are stored in an enterprise directory, the individual policies can also be stored in the same directory as part of the user profile. Therefore, if a user authenticates him/herself we can cache the policy set associated with the user and apply case - based reasoning for injecting policies based on the message, process or domain. In general this injection occurs using a set of policy rules (e.g. business rules) that specify the injection behavior of the policies.

Once, all policies have been injected we need to worry about the enactment [5] of the specified policies as well as the monitoring of these policies.

2 Messaging Framework

The messaging framework is a conceptual model that describes messages within the enterprise. It not only allows us to model message payloads, but also message related processing information such as domain, process and policy information. This relationship between the individual messages' domain and process has an advantage over other frameworks [6,7,8] insofar that it allows policy granularity not only on the message, but also on the domain and process level. For example, when dealing with healthcare information during Claims Processing, all data

access has to be auditable; and therefore an Audit Policy on the domain will be sufficient to control the auditing behavior. To that end any message received during processing that is correlated to the Claims domain will have the policy propagated to each message. The relationship between a message, process, domain and policy is shown in Fig.1. A message must belong to a domain and a process at all times. Further more a process must belong to at least one domain and vice versa. All three primary objects may depend on one or more policies that can be message, domain or process centric.

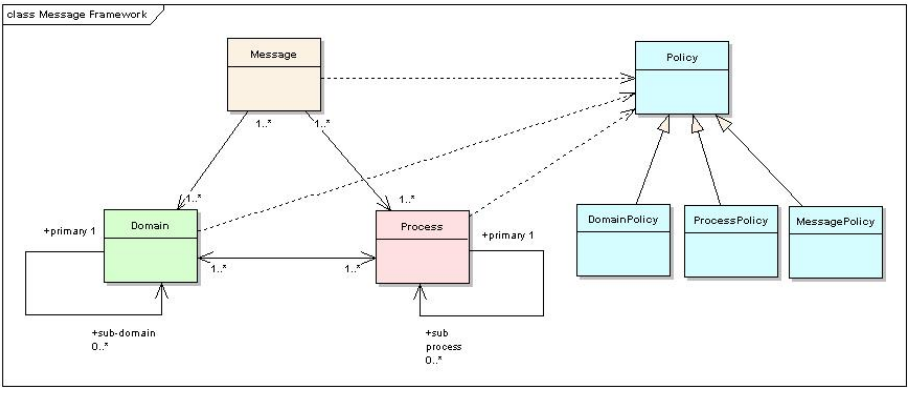

Fig. 1. Simplified Message Framework Model

2.1 Policy Definition Model

The policy definition model can be defined as a set of individual policies that define non-functional processing aspects related to the message itself. Before delving further into the definition model it is necessary to clearly define the difference between a policy and a rule. From our perspective a policy is an atomic enforceable constraint on a system [9] whereas a rule is a conceptualization of a business need. This distinction is necessary to both understand and use this framework. To that end, rules [10,11] maybe used to implement and enforce policies similar to assertions being used in application programming. Fig. 2 shows a simplified Domain and Process Ontology and the relationship between the three different kinds of policies. The domain may subscribe to a domain policy and subsequently all messages related to that domain will use policy propagation from the domain. Similarly, a process may subscribe to a specific process policy, and finally a message itself can subscribe to specific message policy. Below are two examples of defining policies; the first one defines an Auditing policy on the claims domain that specifies to audit every interaction, the second one defines a logging policy on the Adjudicate Claim Process that specifies that a log must be written on every message participating in the process.

```
<SOA.Policy.Audit.Domain Audit.Event="All">
     <SOA.Common.Domain
          Common.Domain.ID="1"
          Common.Domain.Type="Claim"/>
</SOA.Policy.Audit.Domain>
<SOA.Policy.Logging.Process Logging.Level="Debug">
     <SOA.Common.Process
          Common.Process.ID="1"
          Common.Process.Type="AdjudicateClaim"/>
</SOA.Policy.Logging.Process>
```

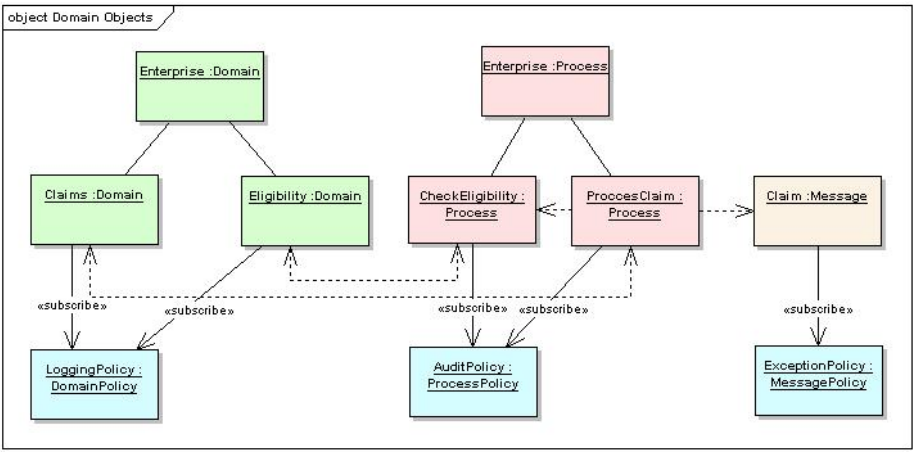

Fig. 2. Domain, Process Ontology with Policy Relationships

2.2 Policy Injection Model

Having defined the overall message model and their relationship with individual policies we now need understand how policies are injected. To that end we developed several policy injection scenarios: Static Injection and Dynamic Injection. Static Injection allows the provider of the message to programmatically specify the policies on the message itself. This approach requires a set of services to access the policy store for domain, process and policies. Dynamic injection on the other hand is based on the domain and process ontology that allows case-based reasoning on the message content and its relationship with the domain or process.

2.3 Policy Processing Model

The policy processing model is based on the translation of the policy definition language into a policy execution language as well as the execution of each policy. The policy execution language essentially invokes a service either synchronously

or asynchronously to validate or enrich the message itself. For example, a logging policy may specify that a message is logged whenever it is being passed between business processes; and therefore, it will be executed asynchronously. A data field encryption policy on the other hand, will enrich the message by encrypting a data filed upon sending and decrypting upon receiving.

3 Message Model Formalization

In order to define and process policies we require a more formalized approach. In this section, we provide a brief introduction to the formalisms used in this research. This framework consists of a mathematical description to specify policies, domains, processes and messages. Further more we describe the mathematical relationship between the individual sets and provide a mathematical induction proof to validate the model.

Definition 1. *(Execution Definition). A message is used within a service S to perform an atomic operation. To that end we define a function f:M'M that takes as input an element of the Message Set M and returns a different element of the Message Set M.*

Definition 2. *(Message Definition). A single message is defined as a four- tuple that contains a payload subset P', a domain subset D', a process subset X' and a constraint subset C'. Therefore a single message is defined as follows.*

$$m_i = \{P', D', X', C'\} \quad i > 0; i \in N' \subset N \tag{1}$$

Given this definition we can define the space of all messages M that are permutations of all individual instances of the above definition. Since the number of permutations does not span a proper vector space we will prove that there exists a subset M' M that represent a valid vector space.

Definition 3. *(Payload Definition).The payload is defined as the data element to be processed within the message. We define the payload as follows:*

$$P' \subseteq P \cup \emptyset \tag{2}$$

Definition 4. *(Domain Definition). The domain is defined as the processing domain the payload is associated. We define the domain as follows:*

$$D' \subseteq D \cup \emptyset \tag{3}$$

Definition 5. *(Process Definition). The process is defined as the process (activity) the payload is associated. We define the process as follows:*

$$X' \subseteq X \cup \emptyset \tag{4}$$

Definition 6. *(Policy Definition). The policy is defined as the policy (constraint) associated with the payload.*

$$C' \subseteq C \cup \emptyset \tag{5}$$

Proposition 1. *(Policy Injection Rule). All policies are derived/defined from a domain, process, or the payload itself; and therefore we can define a function G, that maps a message M to a policy C.*

$$G : M \rightarrow C \tag{6}$$

We need to remember that the domain, process and payload are part of each message; and therefore, for each domain d_i there exists at least one constraint c_i ($\forall d_i \in D \rightarrow \{ \exists\ c_i \subseteq C'\}$). Similarly for each process x_i there exist a constraint (policy) c_i ($\forall x_i \in X \rightarrow \{\exists c_i \subseteq C'\}$). And finally for all messages m_i there exist a constraint (policy) c_i ($\forall m_i \quad \in M \rightarrow \{\exists c_i \subseteq C'\}$).

Proposition 2. *(Policy Execution Rule). Since all policies are based on a message, we can define a function that H that maps the policy C back to a Message M. This is essentially an inverse function of G.*

$$H : C \rightarrow M \tag{7}$$

Theorem 1. *(Completeness of Execution). Let m_i be a message hat defines policies from n=0 .. m, we can proof by induction that the reverse function will exist on the subset M' of all messages.*
If no policies have been defined within a message m_i (n=0), the message will remain unchanged after injecting and executing the policy.

$$m_i = H(G(m_i)) \tag{8}$$

If a single policy n=1 is injected into the message m_i, the outcome of injection and executing the rule results in a message m_j that is part of the message set M' that will have no policies defined (n=0).

$$m_j = H(G(m_i)) \tag{9}$$

Since we defined the policy to be executable and computable on the message, we have proven by induction that the reverse function exists for all messages that have a computable policy set.

4 Architecture

The overall architecture we have chosen is based on highly scalable enterprise service bus (ESB) that acts as the intermediary for messaging [12,13]. The service bus provides asynchronous processing queues for primary business processes and domain activities that are implemented using BPEL [10]. In addition to these orchestrated services a set of utility services for data retrieval and cross-cutting concerns are registered on the bus. Using an enrichment pattern on the message bus, allows the individual messages to be extended and the policy and domain information to be added, and subsequently transformed into BPEL for the policies to be executed. Fig. 3 depicts the conceptual architecture of the solution.

The core of the system is the Message Bus and the Policy control framework responsible for policy injection, policy definition and policy execution. The Policy Control framework uses a policy store to retrieve policies given the context of the message (domain and process). Additionally, the diagram also shows the primary business process, Claim Processing, and the individual domain activities, Data Receiving, Data Pre – Processing, Data Validation and Data Adjudication.

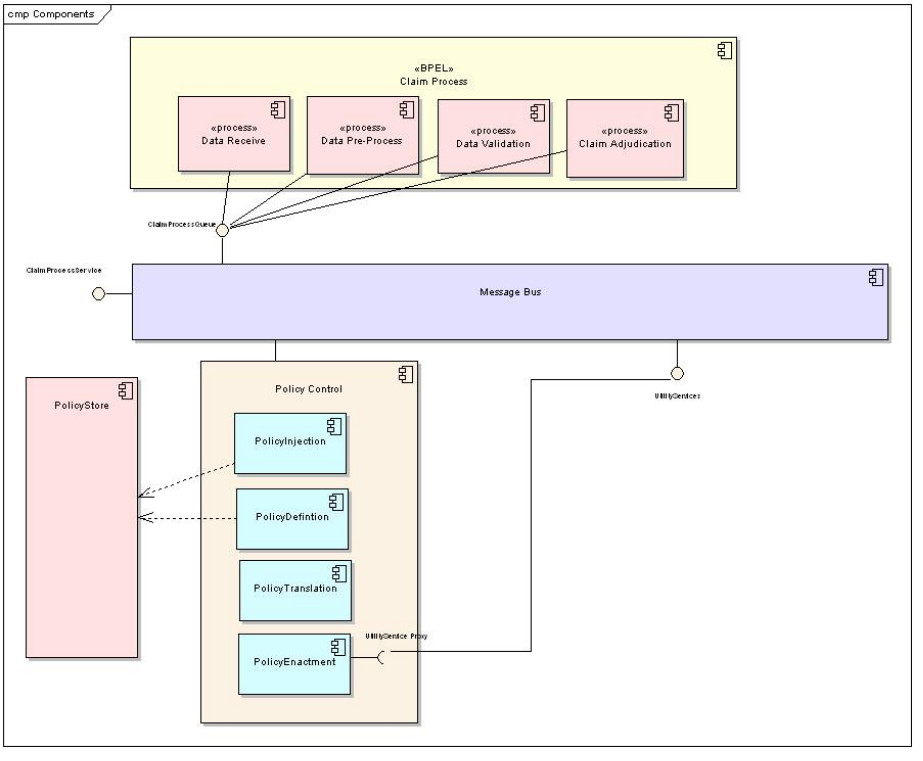

Fig. 3. Conceptual Architecture used by the messaging framework

Since each individual activity is a collaboration of data services that are based on our message model we can use a pipeline execution model to inject, transform and execute the policies using an interrupt pattern on the activity process flow.

4.1 Message Processing

Given that we use an enterprise service bus, the policy control framework will inspect the message while executing the business process orchestration. To that end the policy control framework will subscribe to the policy service queue that is invoked by the BPEL process. At that point the message is inspected, the domain, process and policy information injected. Once the message is complete

the policies will be transformed into executable code and subsequently called based on the context. Once the policy enactment stage is complete, control is returned to the calling context. In other words, the pipeline execution model is guided by the policy control model. The typical flow of a message, once it is put onto a process or domain queue involves the following steps:

- The Message is published onto the primary/main flow queue.
- The Message is inspected synchronously by the policy control framework.
- The Policy Control Framework executes the policy set on the message.
- The Main Process Flow is resumed upon execution/scheduling of all policies.

As can be seen by the scenario above, the policy control will interrupt the main process flow until all policies have been evaluated or processed; and therefore, special care has to be taken on the execution times of the aspects that are being injected. To that end, there are two distinct ways to execute these policies: asynchronously and synchronously. Logging, Auditing and other high volume aspects, are all asynchronous requests to perform a certain action on the message, where the return result is not necessary for the main process to continue. Synchronous policies on the other hand, such as Check Policies and Encryption policies will have to execute synchronously and publish the result message back onto the main process queue.

5 Related Work

A lot of work has been devoted in both industry and academia to policy enforcement, little industrial progress has been made to allow the business stakeholders to define such constraints. The SCA initiative [14] defined a policy framework [8] which allows developers to use doclets and annotations to define policies during development which does not allow a quick adoption to changing policies. Other approaches such as [9], use a constraint based methodology for web services, but leave little room for change.

6 Conclusion and Future Work

Policy definition and policy enactment is an important issue in any successful implementation of a Service Oriented Architecture. In this paper we described an approach that allows various stakeholders in the ecosystem to define policies that will be executed during the execution of a business process or activity. Further more, we showed that policies can be defined coarse grained for optimal usability. Because our approach is unique insofar as the definition and execution of policies is concerned we provide adaptability to changing requirements and let the business and operational stakeholders constrain the business processes. In doing so we reduce the total cost of ownership as no further development effort is necessary, unless new processes have to be built. Our model could easily be extended to include the governance of any processes as it represents a way to

constrain processes with policies, although our focus was based on an adaptable messaging model.

This work is at an early stage, and much more has to be done. The policy definition language, as well as the policy translation and execution language must be refined and evaluated. The performance of the policy control framework has to be considered and tuned as there are many times the injection and enactment algorithm has to be executed.

References

1. Brucker, P.: Scheduling algorithms. Springer, Berlin (2001)
2. Zhao, J.L., Stohr, E.A.: Temporal workflow management in a claim handling system. In: ACM SIGSOFT Software Engineering Notes, Proceedings of the international joint conference on Work activities coordination and collaboration WACC '99 (March 1999)
3. Fremantle, P., Weerawarana, S., Khalaf, R.: Enterprise Services, Examining the emerging field of Web Services and how it is integrated into existing enterprise infrastructures. Communication of the ACM 45(2) (October 2002)
4. Hoschek, W.: The Web Service Discovery Architecture. In: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, Baltimore, Maryland, November 16, 2002, pp. 1–15 (2002)
5. Kozlenkov, A., Fasoulas, V., Sanchez, F., Spanoudakis, G., Zisman, A.: Service discovery and binding: A framework for architecture-driven service discovery. In: SOSE '06. Proceedings of the 2006 international workshop on Service-oriented software engineering
6. Anderson, A.: An Introduction to the Web Services Policy Language. In: POLICY'04. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, IEEE Computer Society Press, Los Alamitos (2004)
7. Web Services Policy Framework (ws-policy). Technical Report, IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, VeriSign (March 2006)
8. Beisiegel, M., Kavantzas, N., Malhorta, A., Pavlik, G., Sharp, C.: SCA Policy Association Framework. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 613–623. Springer, Heidelberg (2006)
9. Aggarwl, R., Verma, K., Miller, J., Milnorm, W.: Constraint driven web service composition in METEOR-S. In: SCC'04. IEEE Conference on Service Computing, Shangahi China, pp. 23–30. IEEE Computer Society Press, Los Alamitos (2004)
10. Andrews, T., Cubera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services version 1.1. Technical report, OASIS, `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf`
11. ILog JRules, `http://www.ilog.com/products/jrules`
12. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley Professional, Reading (2002)
13. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, Reading (2003)
14. Service Component Architecture (SCA) Specifications, `http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications`