

# A survey of policy-based management approaches for Service Oriented Systems

Tan Phan, Jun Han, Jean-Guy Schneider  
 Faculty of ICT  
 Swinburne University of Technology  
 Hawthorn 3122 , Australia  
 {tphan,jhan,jschneider}@ict.swin.edu.au

Tim Ebringer, Tony Rogers  
 CA Labs  
 Level 3, 380 St Kilda Road  
 Melbourne 3000, Australia  
 {tim.ebringer,tony.rogers}@ca.com

## Abstract

*Policy based management in Service Oriented Architecture (SOA) allows organizations to apply rules and regulations on their business processes. Policy has long been employed in the management of traditional distributed systems and many policy frameworks have been proposed. However, SOA differs in several aspects to traditional systems thus there is a unique set of requirements for an effective SOA policy system. In this paper, we evaluate five popular policy frameworks which are IETF, Ponder, KAoS, Rei and WS-Policy against a number of general and SOA-specific criteria to identify what features of these existing systems can be adopted for SOA and what are not. We then, based on their feature sets, discuss the applicability of the frameworks for SOA management.*

## 1. Introduction

The operation of organizations is governed by business logic, which is the logic of core business processes and activities, and enterprise logic which is corporate and government regulations. Organizations apply enterprise logic on business logic by specifying and enforcing policies about access control, Service Level Agreement (SLA) and dynamic resource provisioning on applications [26]. Policy-based management has the advantages of being able to dynamically change the behavior of a managed system according to the changing context requirements without having to modify the implementation of the managed system nor requiring “the consent or cooperation of the components being governed” [28] in the managed system. Also, the declarative specification of rules and regulations in the form of policy statements are more concise, intuitive and simpler to verify than procedural code.

A system implemented following the principles of Service Oriented Architecture (a SOA system) is often a live network of business and non-business functionalities imple-

mented as services and is characterized by dynamism and heterogeneity of the underlying implementation platforms as well as the loosely coupled nature of its building blocks. These characteristics make SOA systems more complex in nature and inherently hard to manage. Policy is considered a key for SOA management [36]. However, it is not clear whether policy-based management techniques employed in traditional telecommunication can easily be adopted for a SOA environment.

There have been a number of efforts evaluating various aspects of different policy frameworks for different application scenarios. Damianou in [9] focused on comparing the nature of the policy specification languages. Duflos in [11] evaluated the suitability of some policy languages for security management in distributed systems. In [1], Aib et al. evaluated both the specification languages and the policy management models of some frameworks for Internet Quality of Services (QoS) management. Tonti in [28] compared policy specification and reasoning capabilities of three semantic web-based policy frameworks for the management of Multi-Agent System. While these works are useful in their own respects, to the best of our understanding, there have been no attempt to evaluate existing frameworks for the management of SOA systems. In this work, we present our analysis of existing policy frameworks from the angle of SOA management. It serves as our first step toward developing a suitable policy framework for SOA.

We have considered a number of policy languages including IETF [23], Ponder [8], PDL [21], SPL [25], KAoS [32, 31], and Rei [18], and WS-Policy [13]. However, in this paper we focus only on IETF, Ponder, KAoS, Rei because they are more complete policy frameworks (including policy models and services), while the others are policy specification languages. We also include WS-Policy in the review for its being the most widely adopted policy language in SOA even though the language does not come with a set of policy services. The review will also point out the need for a higher level policy language than WS-Policy. For each policy framework above, we present a high level

analysis of both the policy language and the management model to examine how suitable they are for being a SOA policy system and what are the missing pieces.

The rest of the paper is structured as follows. The second section presents an example scenario and to identify the requirements for a SOA policy framework. We then evaluate IETF, Ponder, KAOs, Rei and WS-Policy against these requirements in sections 3 to 7. We discuss the evaluation results and make some suggestions in section 8.

## 2. Evaluation Criteria

In this section, we present the criteria upon which we base the analysis of the five frameworks, starting with an example showing some management challenges facing an organization when adopting SOA. Based on the example, we then identify a set of general and SOA-specific requirements that a SOA policy framework should possess.

### 2.1 A business example

SwinBank is in the process of transforming into SOA. It has implemented SOA in a small scale with some initial J2EE and .Net Web Services. The bank has a workflow engine currently running the loan approval business process and will be expanded for the automation of other processes such as card issuance and billing. The bank's billing process is outsourced to an external party SwinBiller via a service interface provided by SwinBiller.

In this SOA transformation process, new services are being identified and implemented and applications are built on top of the services. SwinBank's IT systems must enable the bank to respond quickly to events happening in the financial market. For example, during the 2007 sub-prime mortgage crisis [19], the bank's credit approval workflow needs to be revised and adjusted. The borrower credit verification process is tightened and some extra credit checking operations are added into the workflow.

Operating in an industry with strict regulations such as those about maintaining transaction records to prevent money laundering, or maintaining privacy of consumer financial information, SwinBank's IT systems must enable the bank to ensure the traceability and security of their data, including any temporary data of their long running business processes. Other regulations such as section 404 of the Sarbanes-Oxley act [7] about "maintaining a high level of adequacy for public company's internal control over financial reporting" requires the bank's IT system to be compliant with internal control frameworks such as COBIT [17].

To meet these requirements, the bank's IT management needs to ensure the quality compliance of all the elements in their IT infrastructure, particularly critical business services, during and even before or after their operation. In dif-

ferent cases, the controls might need to be applied on some individual elements, on selected sets of elements or on every element in the system. In addition, to ensure the agility and flexibility of their business processes, the bank wants the compliance to be enforced dynamically and do not want a change in an existing regulation or the introduction of a new one to stop the operation of any existing business process. Also, the performance overhead of the adaptive actions on the system in operation should be at a minimum.

### 2.2 Criteria

From the example, it is clear that SwinBank needs a SOA governance/management system that allows the organization to specify various types of rules and requirements and have them automatically enforced. This section presents a list of desired general and SOA-specific properties for a SOA policy-based governance and management framework that can be used to manage SwinBank's SOA infrastructure. The properties are identified based on our analysis of SOA principles in general and the above example in particular.

#### 2.2.1 General concerns

**Policy specification, analysis and enforcement.** A complete policy framework should include a policy specification language to allow for the expression of policies, a policy deployment model for the distribution of policies into enforcement points, a policy analysis mechanism for making the correct policy enforcement decisions, and a policy monitoring and enforcement mechanism to allow for the identification and execution of policy actions. A comprehensive policy framework should also have built-in support for policy verification, consistency analysis plus redundancy and conflict detection and resolution mechanism to ensure the correctness and quality of the policies specified.

**Language's formal semantics and extensibility.** Policy languages that are based on rich formalisms such as description logic or event calculus make it easier to perform automatic policy analysis than those that are not. A policy language should also be easily extensible to support new types of policy expressions as needed. In this regard, a language with general constructs that can be combined to express different types of policies is preferred to a language which is a collection of predefined types of policy.

**Domains and other forms of grouping.** A policy framework should have a mechanism to support the grouping of elements in a managed system so that policy can be systematically applied to them. The support for *domain* is most desirable as domains allow for the hierarchical grouping of objects. In addition, policies can themselves be managed objects thus policy grouping should also be enabled so that management actions can be performed on a policy set

rather than on individual policies.

**Distributed policy enforcement.** Distributed policy enforcement increases the flexibility and scalability of the framework. Traffic and the impact of a central point of failure can both be minimized when policy enforcement and some of policy decisions can be made locally.

**Meta policy.** Meta policy (or policy about policies) allows rules and constraints (such as timing constraints, prioritization rule) to be placed on policies themselves thereby facilitating policy analysis and conflict resolution.

## 2.2.2 SOA-specific requirements

**Business oriented policy specification.** SOA systems are business driven in which services are identified and implemented according to primary business activities, and applications (service orchestrations) are formed based on organizations' core business processes. The application logic of SOA systems are thus essentially business process logic. Enterprise logic that is used to control these business processes is independent of the logic of the processes themselves and thus needs to be externalized outside of the implementation of the processes. Also, while there is typically one uniform set of enterprise logic to control the business processes, the underlying implementations of the processes are often heterogeneous (different technology platforms and different vendors). Therefore, a policy system needs to stay at a reasonable level of abstraction so that systematic policy requirements can be specified independent of the implementation details.

This also leads to the need for policy refinement, which is the process of translating high-level, business-oriented policies into low-level, implementation-specific policies that can be automatically enforced.

**“Change” support.** One of the most important characteristics of SOA is the dynamism of the system and there are two types of change that need to be supported in policy based SOA management. The first type is changes of the target SOA system which are driven by changing business logic or the evolution of technologies. The second type is changes of the policies themselves as a results of updates in rules and regulations.

SOA applications are created following business processes and thus need to be dynamically adapted following the changes in the processes to foster the agility of business operations. Also, the application of new technologies, new implementation platforms or the adoption of new industry standards all lead to changes in the target SOA systems. This is different from traditional telecommunication systems, which are relatively more static in nature (people do not add or remove a network router very frequently). Therefore, a policy-based management system should be able to model “change” as a first class entity. That means

“change” should be represented as events in the policy language and the management framework should be able to detect “change” and have appropriate actions. Typical SOA events that need to be modeled are service life cycle events (service creation, registration, usage, removal), the composition of services to form applications, the storing of long-term and transient data, or the flow of data between different systems when a workflow is being executed, etc.

Another requirement is the support for dynamic policy update, which is the update of policies without having to stop any part of a system in operation. This feature requires a dynamic model of the managed system and requires policies to be treated as textual statements that are stored in a repository (independent of the enforcement components). Policies will be queried, interpreted and executed by the enforcement components as the system operates.

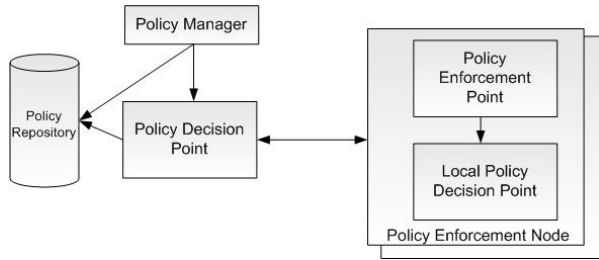
**Policy derivation for service composition.** Service composition is a key aspect of SOA as it allows for the recursive formulation of various applications on top of a set of primary services. Because of this, an essential element of a SOA policy-based management framework is the ability to automatically derive individual services policies from the policy of a service orchestration and vice versa. This is critical as it saves policy editors from having to manually codify policies, which is time consuming and error-prone. In order to support this automatic derivation, a formal representation of the service composition (using formalisms like temporal logic or finite state machine) is needed and essentially the policy representation needs to have a formal basis that supports reasoning.

## 3 IETF

### 3.1 General characteristics

The IETF working group has proposed a set of standards that aim at defining a framework for the representation, management, sharing and reusing of policies and policy information in a “vendor-independent, inter-operable, and scalable manner” [35]. The IETF policy framework includes a policy framework definition language, a policy model, a set of policy terminologies, and a policy architecture meta model. It defines a policy deployment model (as seen in Figure 1) comprising of a policy manager for managing the life cycle of policy objects, a policy repository for storing and classifying policy objects, a centralized policy decision point together with a number of distributed local policy decision points for analysing policy and deriving policy actions based on the state of the environment, and a number of distributed policy enforcement points deployed at the managed element sides to enforce the policy. This model has been employed in many policy-based manage-

ment scenarios such as in XACML access control [24].



**Figure 1. IETF's policy deployment model**

IETF does not define a specific policy language, instead Object Oriented (OO) policy information models are proposed. They include an abstract Common Information Model (CIM) [34] for general representation of a managed system and the Policy Core Information Model (PCIM) [23], which is an extension of CIM to represent policy-related information. In these models, logical and physical elements and their relationship in a managed environment (such as system, services, and users) are represented as OO classes. Both CIM and PICM are vendor and network independent and are useful for defining and modeling at high level a policy system.

IETF models are “sufficiently generic enough to allow them to represent policies related to anything” [23]. However, IETF's initial focus is on network policies to control Quality of Service (QoS) and IPSecurity. Specifically, attempts such as QoS Policy Information Model (QPIM) [35] have been made to extend and translate CIM and PICM for QoS management. In this QPIM model, QoS control is performed by adjusting network device configuration according to the predefined policies. There have also been efforts to define the mapping of IETF models into implementation-specific schema such as Directory Enabled Network (DEN) schema [27] or Web Based Enterprise Management (WBEM) schema [16].

In IETF models, a policy is in the form of *if conditions then actions* rules in which *conditions* are a set of (potentially logically nested in conjunctive or disjunctive normal form) expressions and *actions* are a set of actions that must be executed eventually (regardless of order). For example, according to [35], for the business rule

```

WEB traffic should receive
  at least 50 percent of the
  available bandwidth
  
```

the corresponding IETF policy is

```

If protocol == HTTP
  then minimum BW = 50 percent
  
```

The *actions* and *conditions* can be stored separately in a repository and combined to form different rules. Policy rules in IETF can be prioritized by attaching a priority attribute to the rule, which would help in conflict resolution. However, as the priority value is manually assigned by the administrator, this approach does not scale very well.

In IETF, there is limited support for *domain* (managed objects are not deployed to a domain and policy can not be applied on a domain) and the main mechanism for associating policy with the managed elements is via the use of roles. Managed elements are assigned a role or a number of roles and the policies are then applied to the roles. There is no direct support for meta policy in IETF, however IETF policy are themselves managed elements and can be grouped together so that policy can be applied on the policies.

While being very useful as a general abstract model, the lack of a specific policy language makes IETF's framework not directly usable. The lack of a policy language also means policy verification and refinement are not built-in features of IETF. Rather, the support for these depends on the vendor-specific implementation of the framework. There have been efforts to define policy languages for the framework such as Policy Description Language (PDL) [25]. However, PDL and similar efforts focus on specific aspects of policy management such as access control and can not be used for other aspects.

### 3.2 SOA-specific evaluation

**Business oriented policy specification.** Thanks to CIM and PCIM being abstract, implementation-independent representations of a policy system, high-level policy specification is naturally supported in IETF. That is, users can specify policy using a high-level notation of choice which can then be mapped to IETF models and finally translated to low-level, enforceable vendor-specific instructions or configurations. However, that translation process needs domain-specific information that is external to IETF. For instance, for QoS policy management we first need to start with high-level, informal business policies such as *GOLD class clients have utmost priority access to available bandwidth*. Then the topology of the network, together with the methodology for QoS management (such as Differential Service), will be taken into consideration to refine the high-level policy into QPIM/PCIM models, which can then be mapped to a number of different traffic flow management network device configurations. The mapping of PCIM to specific device configurations is vendor-dependent. Another advantage of the IETF approach is that policy can be easily mapped onto administrative organizations, as “the hierarchical organization of (IETF) policy mirrors most administrative organizations” [35].

**“Change” support.** As mentioned before, IETF models

are very generic making it possible to extend the framework to support any policy-based management. However, the main disadvantage of the IETF approach is its assumption of a static managed system. A complete model of the managed system must have been derived first before any mapping from CIM and PCIM models to device-specific configurations can take places. Any change to the model of the managed systems means the entire mapping process needs to be performed again making the IETF approach not suitable for supporting dynamic changes.

**Policy derivation for service composition.** It is not clear that support for policy derivation from application/service composition to individual service policy and vice versa can be easily accommodated in IETF because IETF's models are based on UML, which has no direct support for reasoning. Elsewhere, there has been efforts to map UML to formalisms which supports reasoning such as description logic [5] or Maude (which is based on rewriting logic) [12], thereby allowing reasoning over systems described using UML. However, it has been revealed that the mapping does have limitations and is not widely accepted at the moment [5, 12].

## 4 Ponder

### 4.1 General characteristics

Ponder is a policy management framework developed at Imperial College. The framework includes the Ponder policy specification language [8], a general architecture and policy deployment model and various extensions of the core model for access control and QoS management. The framework also offers a policy editor customized for Ponder and a domain browser tool.

Ponder allows for the specification of security policies (role-based access control) and management policies (management obligations). A notable feature from Ponder is the support for the parameterization of policy by allowing the instantiation of "typed policy specification", during which any policy element can be passed as a formal parameter to the typed policy. Ponder also supports OO features like inheritance of policy types.

There are four primitive "basic" types of policy supported in Ponder which are (1) authorizations: what activities a member of the subject domain can perform on the set of objects in the target domain; (2) obligations: what activities a manager or agent must perform on target objects; (3) refrains: what actions a subject must not execute on target objects; (4) delegation: granting privileges to grantees. Ponder also supports the creation of composite policies such as *group*, *roles*, *relationship* and *management structures* to facilitate policy-based management in large organizations.

Composite policies allows for the grouping of policies to reflect the organization's management structure and foster the reusability of common definitions [8].

Let us consider a SOA example of Ponder obligation policy that can be used to divert service invocation message to another backup service when the primary existing service is down. This policy specifies that when a service failure happens, the WebServer needs to first log the failure, which is then followed by an adjustment by which the failed service is replaced by a service (with the same id) from the backup service subdomain.

```
inst oblig substituteService {
  on isFailed(serviceID) ;
  subject s = /Server/WebServer/;
  target
    t =/Service/Backup^{serviceId} ;
  do s.log('`failure`', serviceID)
    -> t.enable();
}
```

Ponder allows users to define events, constraints, constants and other reusable elements that can be part of many policies. Domain, which is a grouping of managed objects that share the same "geographical boundaries, object type, responsibility and authority", is an important concept in Ponder. Domain membership can change dynamically with managed objects being added or removed and a managed object can belong to multiple domains. In Ponder, meta policy, with Object Constraint Language (OCL)-based syntax, can also be defined to specify the constraints over a set or group of policies. This enables a more systematic approach for policy analysis such as conflict resolution than the approach adopted by IETF (which assigns properties like priority to policies).

The main problem with the Ponder language is its lack of generality. The language itself is more like a collection of different groups of features rather than a well-designed set of constructs that can be used to describe any behaviors in general. That is, instead of having a common set of language constructs that can be used to describe different types of policy, Ponder has five basic policy types and four composite policy types each with different syntax.

Ponder has different management models, each of them is an extension of the IETF's general policy management model, for different types of management. For example, it uses IETF's CIM for policy-based management of Differentiated Services (managing QoS of Telecommunication services based on predefined Service Level Agreements). Ponder is a collection of different services: role service, policy service, domain service, and event service.

In Ponder, each policy rule contains the target and subjects to which the policy is to be applied. Therefore, low-level, translated policies can be automatically deployed to

the enforcement components. Specifically, obligation and refrain policies are deployed to their subjects while authorization policies are deployed to their targets. In Ponder, each PEP (Policy Enforcement Point) needs to implement a *policy enforcement interface* to enable the loading, unloading, enabling, and disabling of policies once those are deployed to the PEPs [8].

## 4.2 SOA specific evaluation

**Business oriented policy specification.** Ponder only partially supports business-oriented policy specification. It is a middle-level language which is implementation-independent while still requires the editor to have technical understanding about access control and configuration management. However, there have been efforts to allow for automatic refinement of high-level policies, which are business-oriented policies specified as a set of business goals, into Ponder using goal elaboration and abductive reasoning techniques [4] (note that the same techniques are also used for policy conflict analysis and resolution in Ponder). The design of Ponder language takes into consideration organizational structure by enabling concepts like domain, role and relationship.

**“Change” support.** In Ponder’s policy deployment model, policies are eventually compiled by the Ponder compiler into policy Java classes and represented at runtime as policy Java objects deployed in virtual machines at enforcement points. Policy enforcement actions are performed by calling the methods presented in those policy classes. This is a low-level approach that makes policy deployment an one-off step and makes dynamic policy update difficult because whenever there is a change in policies or in the target managed system (such as a new service being implemented and deployed), the related enforcement points need to be reconfigured to accommodate for the new set of generated policy Java classes.

**Policy derivation for service composition.** As such concepts like composition of managed elements does not currently exist in Ponder, there is no direct support for this feature in Ponder.

## 5 KAoS

### 5.1 General characteristics

KAoS is comprised of a set of platform-independent services that “let people define policies ensuring adequate predictability and controllability of both agents and traditional distributed systems” [31, 30, 32]. KAoS is an ontology language as KAoS concepts are defined using an ontology system and KAoS policies themselves are represented using the Web Ontology Language (OWL). KAoS has been used

in many distributed system management situations including Grid Computing and Multi-Agent System and recently, KAoS for Web Services management [30].

In KAoS, ontologies are only currently defined to support four types of policies which are PositiveAuthorization, NegativeAuthorization, PositiveObligation, and NegativeObligation. A policy is associated with an *action* class, which represents a collection of events with similar nature performed by a given actor (a managed element in the managed system). Similar to IETF’s approach, each KAoS policy is associated with management properties such as *priority* to facilitate policy grouping and analysis.

KAoS’s formal semantics is description logic and the KAoS’s OWL-based policy language can be extended to accommodate new types of policies by introducing new classes of ontology. Language extension in KAoS is thus more natural than in Ponder.

The following is an example of an obligation policy in KAoS, which can be used to specify that a service must log any invocation message that it receives. Specifically, the policy mandates that an entity X must perform a CommunicationAction (this is a predefined ontology class in KAoS of which predefined properties are *destination* and *message*). This is the action of sending a LogMessage to the LoggerService when X receives a service invocation message (a CommunicationAction of which the message type is ServiceInvocationMessage that targets at X). Here, we assume that LogMessage, LoggerService, ServiceInvocationMessage are ontology concepts that have been defined in the ontology system. Because of the space limitation, the example is represented in textual description instead of the verbose OWL-based format of KAoS.

Policy p:

```
X is obligated to perform
  CommunicationAction with properties
    hasDestination is subset
      of LoggerService
    carriesMessage is subset
      of LogMessage
when X performs
  CommunicationAction with properties
    hasDestination is subset
      of X
    carriesMessage is subset
      of ServiceInvocationMessage
```

There are two notable set of services provided by KAoS: policy services and domain services. KAoS policy services focus more on the specification, management, conflict resolution, and enforcement of policies. KAoS domain services, on the other hand, enable the hierarchical grouping of people and computational entities to facilitate policy administration. Policy specification, revision, browsing and

domain membership management is made easier in KAoS thanks to the KPAT graphical policy editor tool and KAoS's extensive support for policy templates. Also, the framework supports automatic modality (such as when a subject is both authorized and forbidden to perform the same actions on the same target objects) and application-specific policy conflict resolution using theorem proving. KAoS's current implementation uses the Stanford Java Theorem Prover [14] as the policy reasoning engine.

## 5.2 SOA-specific evaluation

**Business oriented policy specification.** The use of an ontology-based system allows KAoS to have strong support for high-level policy specification. Business and organizational concepts are defined as ontologies and implementation specific configuration are plugged in as sub classes of those high-level concepts. A high-level KAoS policy can be defined as an abstract class which contains a set of abstract *Condition* classes and a set of abstract *Action* classes. Environment-specific concrete conditions and actor concepts are represented as their subclasses.

As mentioned above, KAoS representation is in OWL, which is backed by description logic, making automatic policy refinement becomes description logic inference in nature. As many of the inference problems in description logic are NEXPTIME, NEXPTIME-Hard, or even NP-complete [2], there would be performance issues when the inference needs to be performed on a large set of concepts and relationships.

**“Change” support.** Being an ontology language, KAoS can easily be extended to support the modeling of “change” by adding new ontology sub-classes of KAoS's existing *Action* to represent “change”. The most important feature that makes support for “change” natural in KAoS is the framework's support for dynamic policy update at runtime.

**Policy derivation for service composition.** An advantage of description logic is that it can be utilized in automatic derivation of policy for a service composition from those of individual services and vice versa using KAoS's policy reasoning engine. Also, there have been initial efforts in KAoS to support automatic policy analysis and enforcement for workflow [29].

## 6 Rei

### 6.1 General characteristics

Rei is a policy framework developed by HP which aims at providing domain-independent policy specification using deontic-based [18] constructs. It comes with a policy language and a policy reasoning engine based on the F-OWL

reasoner [38]. Similar to the policy types in KAoS, Rei allows the specification of rights, prohibition, obligations and dispensations policies. The semantics of Rei language is based on a structure of abstract, application-independent ontology hierarchy. Similar to KAoS, domain-dependent and application-dependent ontology can also be plugged into Rei's existing hierarchy as sub-classes of these predefined classes. Rei's policy specification can be in forms of Prolog predicates or RDF-S statements. Rei extends OWL with the expression of relations like role-value maps, which makes the language itself more expressive than original OWL.

The following is an example of a positive authorization policy in Rei in Prolog format, which specifies that an entity A can use the printing service when it is a member of the IT department.

```
has(A, right(usePrintingService,  
            [member(A, ITDePARTMENT)]  
            )
```

There are two main types of meta policies supported in Rei: meta policies for “defaults”, which describe the default behavior of the policy, and meta-meta policies, which allow the setting of precedences within the meta policies themselves (for conflict resolution purpose). Similar to IETF and KAoS, this mechanism of conflict resolution in Rei is in the forms of priority and modality precedence among the policies, which is not very scalable as discussed previously in section 3.1 about IETF.

The most notable missing feature in Rei is the lack of an enforcement model. It is assumed that external enforcement mechanisms can utilize Rei for policy analysis and decisions.

### 6.2 SOA-specific evaluation

**Business oriented policy specification.** Similar to KAoS, the use of ontology allows Rei to have support for high-level policy specification. However, unlike KAoS, Rei does not provide a graphical policy editor making policy specification less user-friendly. Also, Rei's current policy engine does not support policy refinement.

**“Change” support.** Rei's ontology system can be extended to support the modeling of “change” by adding new ontology classes into the model.

**Policy derivation for service composition.** Being an ontology-based language with deontic-logic semantics, Rei can be extended to support policy derivation in a similar way as KAoS.

## 7 WS-Policy

### 7.1 General characteristics

WS-Policy is a W3C recommendation framework for policy specification for Web Services. The WS-Policy framework comprises a set of specifications that together offer mechanisms to represent the capabilities and requirements of Web Services as policies [13]. The framework includes the WS-Policy language [13], which provides a simple and extensible notation to combine the various kinds of policy assertions and form policy descriptions. A related specification - WS-PolicyAttachment [3], which specifies how to associate a policy with Web Services entities (services, endpoints, operations and messages) has also become a W3C recommendation. A collection of domain-specific standards have also been defined to allow for the expression of policy assertions in different contexts like WS-SecurityPolicy [20], WSReliableMessagingPolicy [6], and MTOM [15]. Policy-aware tools such as WSE [22] can generate code to perform policy enforcements automatically.

The WS-Policy language is represented in XML and, in normal form, represents a set of policy alternatives that an entity can choose to comply with. Each policy alternative represents all the requirements that the entity must satisfy to be compliant with that alternative. The language, unfortunately, is not based on a rich formalism. It is loosely based on Boolean logic (a policy is a logical XOR of choices and each choice is a logical AND of requirements). This not only leads to ambiguity in interpretation but also makes policy analysis such as conflict resolution or verification more challenging. However, the design of WS-Policy maintains a nice separation between an outer policy containment structure (the WS-Policy language itself), and domain-specific policy statements (WS-SecurityPolicy, WSReliableMessagingPolicy and MTOM) making the framework very extensible to accommodate policies from different domains.

An example of a WS-Policy policy which specifies that messages coming in and out of a service should be encrypted is as follows. In the example, there is only one alternative to follow, which contains an assertion from the WS-SecurityPolicy domain (*sp* namespace) that mandates the encryption of the envelope of SOAP messages.

```
<wp:Policy...> <wp:ExactlyOne> <wp:All>
  <sp:EncryptedParts>
    <sp:envelope />
  </sp:EncryptedParts>
</wp:All> </wp:ExactlyOne> </wp:Policy>
```

The WS-Policy language is still being revised and at present, it is fairly simple. Features such as meta policy is completely unsupported. Neither the policy themselves

nor the assertions can be prioritized or associated with any other meta-data. The policy is also restricted to Web Services capabilities and requirements, and not to other elements (workflows, client applications, etc) in the managed system.

### 7.2 SOA-specific evaluation

**Business oriented policy specification.** WS-Policy is a low-level policy language which is specific to Web Services implementation. While different types of domain-specific assertion can be plugged in to WS-Policy, these are also low-level. Therefore, WS-Policy and its domain-specific assertion languages do not support business-oriented policy specification.

**“Change” support.** “Change” is not currently modeled in any of the WS-Policy’s assertion languages. Also, as WS-Policy is used to describe the properties related to a specific Web Services, it can not be used to represent changes that happen to the system outside the scope of individual services.

**Policy derivation for service composition.** As mentioned above, WS-Policy is not based on a rich formal semantics making it hard to reason about the policies and to derive policies from other policies.

## 8 Discussion

A summary of the main findings from the analysis of the five popular policy frameworks against the requirements is presented in Figure 2. In this figure, characteristics/features are rated as *support* (a tick), *partial support*, and *not support* (a cross). A framework is considered to support a specific feature when the design of its policy model(s) and/or language(s) and its existing policy services can be adopted, with minimum adjustment, to support the feature. In other words, this means there is “natural” support for the feature built-in the framework. Partial support is when the design and the principle of the framework tends to support a feature (thus the framework can be extended with considerable effort) to support it but there has been no built-in service to support that feature at the time of the review. A feature is considered not supported when the design and principle of the framework can not be extended or adapted to support it without significant compromise of existing features. We apply a similar rating to see whether a framework possesses a specific characteristic or not.

From the table, it can be seen that none of the frameworks are readily applicable for SOA policy management because they are all short of some important features. IETF’s policy deployment model, which comprises a policy manager, a policy repository, a policy decision point

Policy Frameworks		IETF	Ponder	KAoS	Rei	WS-Po licy
Features						
General Charac- teristics	Policy specification, analysis and enforcement	<i>partial</i>	✓	✓	<i>partial</i>	<i>partial</i>
	Language's formal semantics and extensibility.	✗	<i>partial</i>	✓	✓	✗
	Domains and other forms of grouping	✓	✓	<i>partial</i>	<i>partial</i>	✗
	Distributed policy enforcement	✓	✓	✓	✓	✓
	Meta policy	✗	✓	<i>partial</i>	✓	✗
SOA- specific criteria	High Level Specification	✓	✓	✓	<i>partial</i>	✗
	"Change" Support	✗	✗	✓	✓	✗
	Policy Derivation	✗	✗	<i>partial</i>	<i>partial</i>	✗

**Figure 2. Feature comparison**

and distributed policy enforcement points, scale well and can be adopted for SOA management but the lack of a specific language and specific policy services makes the framework less readily applicable. Ponder's lack of support for dynamic policy update limits the framework to systems which are more static in nature, also making the framework less applicable for SOA. KAoS and Rei appear to be more suitable for SOA systems due to their support for dynamic policy update. Also, both KAoS and Rei are ontology languages, of which semantics are based on description logic and de-ontic logic respectively, which help facilitate policy analysis and reasoning such as policy verification and refinement as well as policy conflict resolution. Among the two, KAoS appears to be a more complete framework. Rei lacks a policy enforcement model and does not come with a graphical tool for policy specification and domain management. The main drawback of both is the fact that they are less scalable when the managed systems are large because complex ontology hierarchies need to be formed and reasoning about them is computationally expensive. WS-Policy is a low-level policy language that is specific to Web Services implementation and is not suitable for managing an overall SOA system.

**The missing feature - design time governance.** Design time governance represents a new application scenario for policy-based management - the management of development projects: including system development processes and quality control of the systems being developed. Traditional policy systems focus more on runtime management such as access control, service level delivery, exception handling, or disruption prevention. SOA systems, on the other hand, need to be governed from the first step of their life cycle so

that quality is assured and regulations are respected.

Design time governance focuses on quality control and compliance verification and needs a more declarative approach for policy specification. To support design time governance a policy language should be able to be extended to represent policies that control the quality of services, their compositions, and related artifacts such as service contracts and SLA. For example, a policy can be defined to regulate that any service developed must support public key encryption. Our evaluation shows that none of the discussed policy framework is suitable for design time SOA governance.

Specifically, the general design of IETF is toward being a runtime management framework with policies specified as a combination of conditions and actions. This makes it hard to extend IETF to support design time governance, which is less reactive in nature. Ponder is a runtime policy management framework which supports event-based runtime controls like authorization, authentication, and obligations, etc. Similarly, the current KAoS ontology system only supports runtime management and in order to have support for design time governance, a new set of ontologies to model service design artifacts such as service interface descriptions, service contracts and policies and other service metadata will need to be introduced first. Rei is also designed to be a runtime policy framework and there is no simple way to extend the framework to support design time governance. Finally, WS-Policy is still more a runtime-management framework because the policies specified are mostly those that will result in actions to be manifested on the wire during runtime interaction. However, as WS-Policy is a declarative language with focus more on describing the capabilities and requirements of individual services, the language is more suitable than other procedural policy languages for describing service properties.

In summary, KAoS or Rei's ontology-based languages are suitable for high-level SOA policy representation. WS-Policy, being widely accepted and supported by major SOA vendors, is a good choice for low-level policy representation for Web Services-based implementation of SOA. It would be useful if a policy refinement mechanism can be developed for the translation of high-level business oriented policies specified in KAoS or Rei's into low-level WS-Policy statements for runtime execution.

## 9 Conclusion and future work

In this paper, we have presented a set of desirable properties for a SOA policy framework and have evaluated five popular policy frameworks based on these desired properties to see to what extent the frameworks can be adopted for the management of SOA systems. We conclude that given their feature sets none of the existing frameworks are readily applicable for SOA management. However, parts of

each framework such as the policy specification languages of KAoS and Rei or the deployment model of IETF can potentially be adopted for SOA. This work serves as the starting point for us to develop a comprehensive policy framework for SOA governance and management.

## References

- [1] I. Aib, N. Agoulmine, M. Fonseca, and G. Pujolle. Analysis of policy management models and specification languages. In *Network control and engineering for Qos, security and mobility II*, pages 26–50, Norwell, MA, USA, 2003. Kluwer Academic Publishers.
- [2] F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. A Description Logic Based Approach to Reasoning about Web Services. In *Proc. WWW 2005 Workshop on Web Service Semantics (WSS2005)*, Chiba City, Japan, 2005.
- [3] S. Bajaj and et al. Web Services Policy Attachment 1.2. Technical report, W3C, Apr. 2006.
- [4] A. K. Bandara, E. C. Lupu, J. Moffett, and A. Russo. A Goal-based Approach to Policy Refinement. *Policies for Distributed Systems and Networks. POLICY 2004*.
- [5] D. Berardi, D. Calvanese, and G. Giacomo. Reasoning on UML Class Diagrams using Description Logic Based Systems. In *Proc. of the KI*, 2001.
- [6] R. Bilorusets and et al. Web Services Reliable Messaging Protocol 1.0. Technical report, W3C, feb 2005.
- [7] Congress of America. Sarbanes-Oxley Act of 2002. 2002.
- [8] N. Damianou. *A Policy Framework for Management of Distributed Systems*. PhD thesis, Imperial College, 2002.
- [9] N. Damianou, A. Bandara, M. Sloman, and E. Lupu. A Survey of Policy Specification Approaches. Technical report, Department of Computing, Imperial College of Science Technology and Medicine, London, 2002.
- [10] N. Damianou, D. Naranker, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *POLICY '01: Proc. of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, London, UK, 2001. Springer-Verlag.
- [11] S. Duflos, G. Diaz, V. C. J. Gay, and E. Horlait. A Comparative Study of Policy Specification Languages for Secure Distributed Applications. In *Management Technologies for E-Commerce and E-Business Applications: 13th IFIP/IEEE International Workshop on Distributed Systems, Operations and Management, DSOM 2002, Montreal*, volume 2506 of *Lecture Notes in Computer Science*, pages 157–168, Berlin / Heidelberg, 2002. Springer.
- [12] F. Durn, J. Herrador, and A. Vallecillo. Using UML and Maude for Writing and Reasoning about ODP Policies. In *Proc. of POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003*, 2003.
- [13] S. B. et al. Web Services Policy Framework 1.5. Technical report, W3C, apr 2007.
- [14] G. Fank. JTP: An Object-Oriented Modular Reasoning System. Technical report, Stanford University, 2004.
- [15] M. Gudgin and et al. SOAP Message Optimization Transmission Mechanism 1.0. Technical report, W3C, 2005.
- [16] IETF. Specification for the representation of CIM in XML, version 2.2. Technical report, IETF, 2007.
- [17] IT Governance Institute. Control Objectives for Information and related Technologies COBIT 4.1. 2007.
- [18] L. Kagal. Rei : A Policy Language for the Me-Centric Project. Technical report, HP Labs, September 2002.
- [19] A. N. Krinsman. Subprime Mortgage Meltdown: How did it happen and how will it end? *The journal of structured finance*, 8, No. 2, 2007.
- [20] K. Lawrence and et al. Web Services Security Policy. Technical report, W3C, jun 2005.
- [21] J. Lobo, R. Bhatia, and S. A. Naqvi. A Policy Description Language. In *AAAI/IAAI*, pages 291–298, 1999.
- [22] Microsoft. Web Services Enhancement 3.0. Technical report, Microsoft, 2005.
- [23] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model – Version 1 Specification. 2001.
- [24] T. Moses. eXtensible Access Control Markup Language.
- [25] C. Ribeiro and P. Guedes. SPL: An access control language for security policies with complex constraints. 1999.
- [26] D. Scott and T. Bittman. The Evolution Toward Policy-Based Computing Services. Technical report, 2001.
- [27] J. Strassner. Mapping the Policy Core Information Model to a Directory. Technical report, OASIS, 2001.
- [28] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. Technical report, IHMC.
- [29] A. Uszok, J. Bradshaw, R. Jeffers, and et al. Applying KAoS Services to Ensure Policy Compliance for Semantic Web Services Workflow Composition and Enactment. In *International Semantic Web Conference*, pages 425–440, 2004.
- [30] A. Uszok, J. Bradshaw, R. Jeffers, M. Johnson, and et al. KAoS Policies for Web Services. Technical report, W3C.
- [31] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. *Policy*, 00:93, 2003.
- [32] A. Uszok, J. Bradshaw, M. Johnson, R. Jeffers, and et al. KAoS Policy Management for Semantic Web Services. *IEEE Intelligent Systems*, 19(4):32–41, 2004.
- [33] A. Uszok, J. M. Bradshaw, R. Jeffers, M. Johnson, and et al. Policy and Contract Management for Semantic Web Services. Technical report, IHMC, 2004.
- [34] A. Westerinen and J. Strassner. CIM Core Model White Paper: Common Information Model (CIM) Core Model, version 2.7. Technical report, IETF, 2003.
- [35] Y. Snir and Y. Ramberg and J. Strassner and R. Cohen and B. Moore. Policy Quality of Service (QoS) Information Model. Technical report, IETF, 2003.
- [36] Zapthink. The SOA Management Landscape. *Zapthink Bulletin*, 2006.
- [37] Y. Zhang, X. Liu, and W. Wang. Policy Lifecycle Model for Systems Management. *IT Professional*, 7(2):50–54, 2005.
- [38] Y. Zou, T. Finin, and H. Chen. F-OWL: an Inference Engine for the Semantic Web. In *Formal Approaches to Agent-Based Systems*, volume 3228 of *Lecture Notes in Computer Science*. Springer-verlag, November 2004.