A generalized RuleML-based Declarative Policy specification language for Web Services

Steve Ross-Talbot[1], Said Tabet[2], Sharma Chakravarthy[3] , and Gary Brown[1]

[1] Enigmatec Corporation, UK
steve@enigmatec.net; gary@enigmatec.net
[2] Said Tabet, 24 Magnolia Road, NATICK, MA 01760, USA
stabet@ruleml.org
[3]University of Texas at Arlington, USA
sharma@cse.uta.edu

Abstract. A policy can be defined as a declarative specification of guidelines, rules of conduct, organization, and behavior of entities in a given environment. The area of policy languages is still maturing from both a research and industrial point of view. From an application standpoint, this field has been dominated by very narrow applications in privacy and security. A number of languages have been proposed, most of them based on limited expressive power. RuleML is a rapidly evolving rule interchange platform for Web Services and Distributed systems, for Semantic Web Rules and Semantic Web Services. In this paper, we introduce a policy specification language extending RuleML to handle various policy descriptions embedding rules and constraints marked-up in the RuleML language for Web Services. Our proposal goes beyond the description of policies and their execution, but also defines a framework for policy interchange.

Keywords. Policy Languages, Policy interchange and execution, Web Services, rules, inference, RuleML, ontologies, taxonomies, Web Services, reactive systems, procedural attachment, Event Models, ECA systems.

1. Introduction

A policy can be defined as a declarative specification of guidelines, rules of conduct, organization, and behavior of entities in a given environment. A policy language will require the expressive power of a declarative language (rule-based) with the capability to support declarative events, and built-ins for Web Services.

Web Services are presented as a foundation for a new generation of e-commerce and business-to-business (B2B), as well as the glue for heterogeneous distributed enterprise systems. For such a model to succeed, handling real-time, reactive component interactions using a declarative approach, web technologies and standards is needed. The development of complex distributed systems requires suitable architectural paradigms, and must adopt a policy approach committed to the expressive power of a declarative rule-based markup language. RuleML provides us with such a platform.

In this paper, we propose a solution that is not biased towards a specific application in network management, access control or privacy. Our proposal includes support for a general purpose, web service oriented policy markup language. The use of RuleML [1], itself based on a modular concept of sublanguages [2], enhances the capabilities of the policy language in a larger setting, allowing it to scale and support direct updates and new features offered by RuleML, in particular in specifying constraints and composing conditional statements for web services. In this paper, we demonstrate the functionality of our policy language with two use cases, fully implemented in RuleML and running with two different implementations. The ability to provide a platform for policy interchange and composition is another strong functionality that is proposed in this paper, and required for the success of web services policy systems.

2. Policy

A policy is by definition “A plan or course of action”, “A guiding principle, “A procedure considered expedient, prudent, or advantageous”.

We take the former as our principle definition, although the latter can also be said to be informative as policy in this sense can provide useful documentation and so describe a guiding principle. A policy is a set of rules and facts that can be used to determine a course of action consider expedient, prudent and advantageous as well as being a description of a guiding principle

Policies can be viewed as constraints as well as facts that can be used to determine what an IT system is required to do to meet some business objective. We might have a policy for premium customers that describes a particular level of security and this may be different to those customers that are not premium customers. We might have a policy that describes some throughput of a system that is a reflection on a service level agreement, perhaps it relates to the number

of trades and prices that can be executed or offered from a customer to an Investment Bank. It might relate to the amount of electricity that a data centre can consume based on the size of their electricity pipe at their building.

Policies exist in the real world in many forms. They document insurance contracts, they record what employees should do in the event of named situations occurring in an employment handbook. Thus policies may constrain, may define pertinent situations and may suggest courses of action. Policies may be static in that they can be applied to some entity before that entity attempts to enact something, they may be dynamic in which case they constrain the runtime behavior of some entity as it tries to enact something.

3. Policy Languages

Policy languages are imbued with nomenclature pertinent to the domain in which the policy is applied. And yet across a wide range of possible uses (and across domains) policy languages have some common features. They must be able to describe constraints and they must be able to record facts and support inferences over such facts (e.g. we need to be able to infer that a customer is a premium customer). We must be able to ensure that a constraint is not broken or if it is another policy is enacted to govern how this is reported or dealt with.

The common features of a policy language can be broken down into facts, derivations and constraints and these form the basis of a taxonomy of language features that are required to express policies. Policy languages are required to import domain specific ontologies [3, 4], which is also a common language feature, to be able to describe policy in the nomenclature of the domain of interest.

The RuleML family of languages provides the expressive power of a declarative rules language with a markup approach enabling modular sublanguages integration for various policy representations. RuleML provides all of the features and meets the requirements of a policy language as described above [5].

Our proposed policy language defines rules and facts to describe constraints. We distinguish between the following facts/rules for policy representation:

- Structural/organizational facts and rules. These rules are used to encode domain specific ontologies
- Service definition facts and rules, provided with links to the structural rules and facts. Such links are defined using webized entities, as supported in the current RuleML specification.
- Task-specific rules and facts, provided by the service clients.

The benefit of categorizing the policy rules in this way is that an organzation can define a common ontology that can be shared amongst many services. Each service can then refine the rules to meet its individual requirements. However, the real benefit is in the ability of the requesting client to be able to utilize this information to make informed decisions about the usage of those services in achieving business objectives.

Let's consider an example to illustrate these three types of rules and facts. The first type of rules (i.e. organizational) will be described in this section, with the other two types of rules (service definition and task specific) being illustrated as part of the use cases in the following section.

Since the RuleML markup is well known and a number of rules and rulebases have been presented in the past [See www.ruleml.org] and used in various applications, we will focus on a natural language presentation of the rules and facts for this example. The RuleML markup for this example can be found at http://policy.ruleml.org/.

In our example, the organizational rules relate to customers and SLAs. The following two policies are rules on Customer level (gold or platinum), which is based on the customer's support type purchased:

Policy 1:
A Platinum Customer is a Customer who purchased a Platinum Support

Policy 2:
A Gold Customer is a Customer who purchased a Gold Support

The following two policies define the customer's maximum transaction response time based on their support level (gold or platinum). Such SLAs can be linked using Web URIs, as supported in RuleML (rule and fact labels provide such implementation in RuleML).

Policy 3:
IF a Customer is a Platinum Customer THEN the Customer's Max Transaction time is

5 seconds

Policy 4:
IF a Customer is a Gold Customer THEN the Customer's Max Transaction time is 10 seconds

Two additional SLA rules have been defined to define the cost per transaction based on the customer level and their service usage:

Policy 5:
IF a Customer is a Platinum Customer and the Service Usage is less than 30 days THEN the transaction cost is 2 cents

Policy 5b:
IF a Customer is a Platinum Customer and the Service Usage is more than 29 days THEN the transaction cost is 1 cent

Policy 6:
IF a Customer is a Gold Customer and the Service Usage is less than 30 days THEN the transaction cost is 4 cents

Policy 6b:
IF a Customer is a Gold Customer and the Service Usage is more than 29 days THEN the transaction cost is 3 cents

Facts representing the key entities in the model are maintained by the organization in a CRM system for example. Here are some of such facts showing customer Fred as a customer who purchased a platinum support and customer Joe as a customer who purchased gold support:

Fact1:
Customer Fred purchased a Platinum Support.

Fact2 :
Customer Joe purchased a Gold Support.

The facts shown above can be imported from various sources, and used to infer and reason over the rules on a request basis (while performing a service discovery, validation, etc). The RuleML markup is a direct translation from these rules/policies and facts. This specific example is supported by the current version of RuleML. The concept of Customer, Service, and other such entities can be defined either locally, or in a separate ontology that is referenced using a URI, or imported using the RuleML import mechanism (similar to OWL).

At the top level, the set of organizational rules presented above will be encoded as a rulebase (containing all the rules). The rulebase itself will include a URI link to ontologies and facts that are relevant, and can be deployed to rule engines using a URI-based access.

4. Web Service Use Cases

This section describes two use cases for how RuleML policies can be used with a web service.

NOTE: The use cases we have selected include the concepts of constraints based on security and reliable messaging, as specified in the use case within the "call for participation", however we have generalized these concepts within a more comprehensive use case that encompasses business specific information and SLAs. This is because we believe the "constraints and capabilities" standard will need to be relevant in the context of wider business objectives, not just service specific tasks.

a) Deployment (Registration) Validation

When an organization wishes to deploy a new or updated version of a web service, they will want to validate that this service conforms to relevant internal compliance guidelines. For example, to ensure that certain levels of security or reliable messaging are used, or to ensure that only certain approved information types can be used within the WSDL interface.

This type of validation would be performed by either processing the WSDL service definition or inspecting additional properties (capabilities/facts) that would have been associated with the service definition. For example, the following facts could be associated with the service definition:

Fact3:
Messaging is Reliable

Fact4:
Security is Kerberos

[NOTE: This is an example of the second type of web service related policy rules/facts – i.e. associated with the service definition. These facts could actually be encoded as Property definitions within a WSDL2.0 [6] service definition].

Therefore the validation can be performed on the basis of applying a constraint rule over these facts (e.g. does service use reliable messaging?), and/or transformation rules to the service definition to analyse aspects of the description (e.g. does the service only use authorized types). For example,

Constraint1:
Messaging is Reliable AND Security is Kerberos

[NOTE: This is an example of the third type of web service related policy rules, i.e. the task specific rules and facts. This particular use case did not require access to the rules/facts defined in the previous section, associated with the organization.]

b) Advanced Discovery

To enable the associations between web services to be dynamically established based on business policies; it would be possible to define capabilities against a web service definition that can be evaluated when performing a discovery request.

The first type of request that could be performed is based on static capabilities – where details of the web service could be defined as facts. These details may be based on a domain specific ontology. To define the selection query relevant for these static capabilities would require the discovery request to be accompanied by a constraint rule that would need to evaluate to true for the service to be considered relevant. For example, we may only want to discover services that use reliable messaging and kerberos security (as shown in the previous use case's Constraint 1) – these are fixed (static) capabilities that would not generally vary between different requests for the service.

A more advanced type of discovery request could associate derivation rules with the service that would be evaluated against 'facts' provided as part of the discovery request. This enables the nature of the request to only be known at runtime, when the client provided information is evaluated against those rules. For example, the response time guaranteed by the service may be dependent upon the importance of a customer – and therefore the “response time” capability of a service will be dependent upon who is performing the discovery request (i.e. a dynamic capability only resolved at runtime).

As well as using the dynamically derived information for the purpose of service selection, possibly based on Service Level Agreement characteristics (i.e. response time), on demand usage may also require the 'cost' of a service to be factored into the selection process. The balance between meeting a Service Level Agreement criteria, and the cost of using the service, would potentially be a derivation rule specific to the individual user making the request.

For example, the client request could contain the following task specific rule,

Constraint2:
Transaction cost less than 4 cents

If customer 'Fred' was to perform the request, then as he is a platinum customer, the cost per transaction will always be lower than 4 cents. However, if 'Joe' was to perform the discovery request, and the service usage was less than 30 days, then he would not be given access to the service as it would not meet the constraint (due to Policy 6).

It may also be appropriate to return a list of potential service definitions that meet the discovery request criteria, with the values for any derived information (e.g. the cost of using the service) returned as part of the details associated with each potential service instance. This would enable the client to perform any further selection process locally.

5. Future Work

Reaction rules are a major member of the RuleML family of sublanguages. The common manifestation of a reaction rule is an Event-Condition-Action rule [7,8], where a composite event can be defined, and when detected a condition tested which if evaluated to true will cause an action to be performed. We are exploring ways to introduce them into the specification of complex dynamic web services requiring reactiveness in their behaviour.

We might define an SLA property in terms of a constraint associated with a service, we may wish to associate a reactive rule to be triggered if the constraint is broken at runtime. This rule could be used to either report the violation to the appropriate group within an organization, or initiate some activity to overcome the underlying problem that has resulted in the SLA failure. E-C-A rules lend themselves to this rather well, the constraint is embodied by the composite event and condition and the action embodies the consequence.

RuleML is well suited to both static and dynamic/realtime policy governance. Thus reaction rules are best suited to behavioural manipulation at runtime in order to ensure policy is followed at runtime, and derivation rules are best suited for encoding static policies that govern how components are connected and configured.

We anticipate using both derivation and reaction rules in future work to provide a dynamic feedback to discovery services to test out our ideas. The aim would be to provide dynamic updates to a discover service based on real-time information. This in turn would affect the way in which services are connected based on realtime information and so achieve one of our goals of SLA and policy adherence at runtime.

Further work is also required to explore the range of policies at different levels (Structural, Service and Task) their relationships and their consistency (e.g. It would be good to know that some task level policy that specifies some cost is achievable given the service based policy before the policies become live).

Further work on integration with discovery services is needed to provide a policy driven discovery that reflects customer needs in a service oriented environment moving toward utility on-demand computing.

Finally further work is required to show that our policy language and implementations thereof can work seamlessly with WSDL2.0 and other relevant Web Services standards.

This further work would provide both the SLA monitoring at multiple levels as well as the SLA adherence at runtime along with the business relevance required to deliver a solution that customers both want and need within a Web Services framework.

6. References

[1] RuleML

[2] Harold Boley, Said Tabet, and Gerd Wagner: Design Rationale of RuleML: A Markup Language for Semantic Web Rules, Proc. SWWS'01, Stanford, July/August 2001.

[3] Web Ontology Language (OWL)

[4] Resource Description Framework (RDF)

[5] The Policy RuleML Technical Group

[6] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language

[7] U. Dayal, A. Buchmann, and S. Chakravarthy, ``HiPAC -- An Active, Object-Oriented Database System", in Active Database Systems -- Triggers and Rules for Advanced Database Processing, U. Dayal and J. Widom (Eds.), Morgan Kaufmann Publications, pp. 177-206, 1996.

[8] S. Chakravarthy and D. Mishra, ``Snoop: An Expressive Event Specification Language for Active databases", in Knowledge and Data Engineering Journal, Vol. 14, pp. 1-26, 1994.

[9] Steve Ross-Talbot, Harold Boley, and Said Tabet: Playing by the Rules, Application Development Advisor 6(5), June 2002, 38-43.

7. Acknowledgements

We would like to acknowledge the ideas, requirements and guidance we have received from the SOA Working Group at Credit Suisse First Boston, Harold Boley Adjunct Professor, Faculty of Computer Science University of New Brunswick and Kevin O'Donnell Product Manager at Enigmatec Corporation.