A Service Oriented Policy Architecture for Managing Services Provided by Web-Application Frameworks

Kevin Feeney, Dave Lewis, Declan O Sullivan

Abstract

Policy Based Management technologies represent a potentially important tool in the management of user services provided by web application frameworks. Current policy management systems are, however, a poor fit for the domain due to their lack of support for decentralised management in open environments. The Community Based Policy Management System is a policy framework designed to facilitate management of services in domains where relationships are highly dynamic and flexible, where policy specification is distributed and may use multiple languages, and where management decision-making is shared between communities of users and service providers. This article describes the CBPMS schema and architecture and uses an example to show how it provides flexible, dynamic and extensible policy based management capabilities to the providers of user-services on the web.

Introduction

In the era of web 2.0, the Internet can increasingly be considered to consist of a network of web-application frameworks, providing a wide range of services[1] to users, who consume services from a wide range of providers. The increasing availability of open web-service interfaces provided by these frameworks has created potential for the composition of these services into dynamic compositions - “mashups” that draw information from multiple independently-provided services to better serve user-requirements. However, from a management point of view, web-application frameworks largely remain isolated islands. They lack facilities to apply management rules, such as differentiated pricing models and access control rules, to the individual services used in compositions, in combination with rules that apply to the overall composed services, which may be collaboratively managed. This limits the ability of service providers to take advantage of the opportunities for collaboration that the increasingly rich world of open web-services provides.

[1] By "service", we mean any service that is provided to a user - email, social networking, photo-sharing, etc, not to be confused with "web-services" which are generally understood to have a much narrower definition.

Policy Based Management (PBM) is an increasingly popular method for combining flexibility and efficiency in systems and network administration. In PBM systems, decisions about the behaviour of the system are specified as rules, often expressed in a high level language, which are then mapped into concrete behaviours by the policy system. The Community-Based Policy Management System (CBPMS) is a policy framework that is uniquely designed to manage policies in situations where services are owned by a potentially dynamic network of different organisations, communities and individuals. It is designed as a suite of open Internet services, with all elements addressed by URIs. Its architecture enables a wide range of independently managed application frameworks to be seamlessly integrated into its management model.

We first introduce a scenario to illustrate the management problems that service providers face in deploying collaborative services. We use this scenario to show how the decentralised model of the CBPMS facilitates the collaboration of a wide range of actors in specifying policy rules to solve management problems. We then describe the CBPMS service oriented architecture and show how its design supports PBM approaches in an open Internet environment. The practicalities of policy integration are illustrated by examples drawn from our implementation of this service based on the Drupal web-application framework[2].

Application Scenario

Imagine a company providing a service called *AboutMe,* which aggregates information about groups and individuals from a wide range of Internet services[3]. It traces their social networks, press releases, blog postings, favourite books, latest photographs, share prices and so on. Each user can choose what personal information the *AboutMe* service has access to, by using the *addSource* and *removeSource* capabilities that it provides.

Our company would like to make the service available to third-party service providers who might offer additional indexing, collation and analysis of the data. However, this would inevitably give rise to privacy concerns among users. As an example of the pitfalls of aggregating and sharing personal information, the Facebook beacon system, which shared details of users' online purchases with their network of contacts, was turned off by default in November 2007 in response to user-protests [1]. The problem is that groups and individuals

[2] www.drupal.org
[3] www.spokeo.com website provides a somewhat similar service

often have very different requirements or preferences in how their information should be used in different web applications, but are rarely provided with a suitable means for flexibly controlling and monitoring this.

There is no problem in finding ways to express precise and expressive rules about the sharing of data. A very wide range of policy specification languages have been proposed [2]. Some of these languages are capable of expressing sophisticated rules in a wide range of application domains, such as the eXtensible Access Control Markup Language (XACML) [3] and Ponder [4]. Others are tightly focused on a specific application domain. For example, the Web-Services Policy Framework [5] and Web-Services Security Policy Language [6] define an extensible policy model and a concrete language designed to express SOAP message security assertions respectively. The problem is, however, that most PBM systems are designed primarily for application within large organisations and depend on a carefully analysed, centrally-defined set of roles. They lack abstractions for modelling the fluid relationships and distributed management that is characteristic of open Internet services – where each actor has their own view of the world and defines their own policy for the use of their services. PBM systems also tend to be tightly-coupled to policy languages, with architectures that depend upon the language's semantics. Given the heterogeneity of service compositions and their policy requirements and the wide variety of policy languages available, it is not clear that it is desirable for a management system to impose a single, system-wide policy specification language. Also, in practice, service management systems tend to be components of proprietary management systems. For example, IBM's SOA governance products [7] are designed for deployment within IBM's enterprise management suites. Such proprietary systems tend to be difficult to deploy in open environments across multiple independent organisations.

The CBPMS, by contrast, is a policy framework designed to facilitate the application of PBM to domains where relationships are highly dynamic and flexible, where policy specification is distributed and may use multiple languages, with management authority shared between independent groups and individuals. Instead of using centrally defined roles, the CBPMS maintains a distributed map of the relationships between individuals, groups and the services that each manages, with each collaborating individual and group defining their own part of the map - their own *community*. This distributed *community map* is used as the basis for the delegation of management rights and access to services across the network. The web

application framework is modelled as a set of managed services, each of which is sub-divided into sets of capabilities, to create a hierarchical *resource model.* The model can be sub-divided along any axis that partitions the services' capabilities into well-defined subsets - and the system uses this model to enable communities to delegate the right to write rules about well-defined subsets of the capabilities offered by their services to third parties. Service owners still retain ultimate control of delegated services due to the fact that the policy rules which they specify have precedence over those specified through delegation.

So how does the CBPMS help solve the *AboutMe* provider's problem?

First we must create a hierarchical resource model of the *AboutMe* service, with each node representing a well defined subset of its capabilities. In this case the service's capabilities can be neatly sub-divided into a tree of actions and a tree of targets as shown in Figure 1. The CBPMS uses this resource model to manage the controlled distribution of service capabilities across the community map. Each community in the map can grant access to a well-defined subset of the capabilities that it owns by delegating a *resource authority* to another community. Resource authorities are secure tokens which contain a reference to nodes on the resource model. For example, in figure 1 *[View, Contact Details]* is a resource authority representing the capability to view any of the contact details that *AboutMe* can provide.

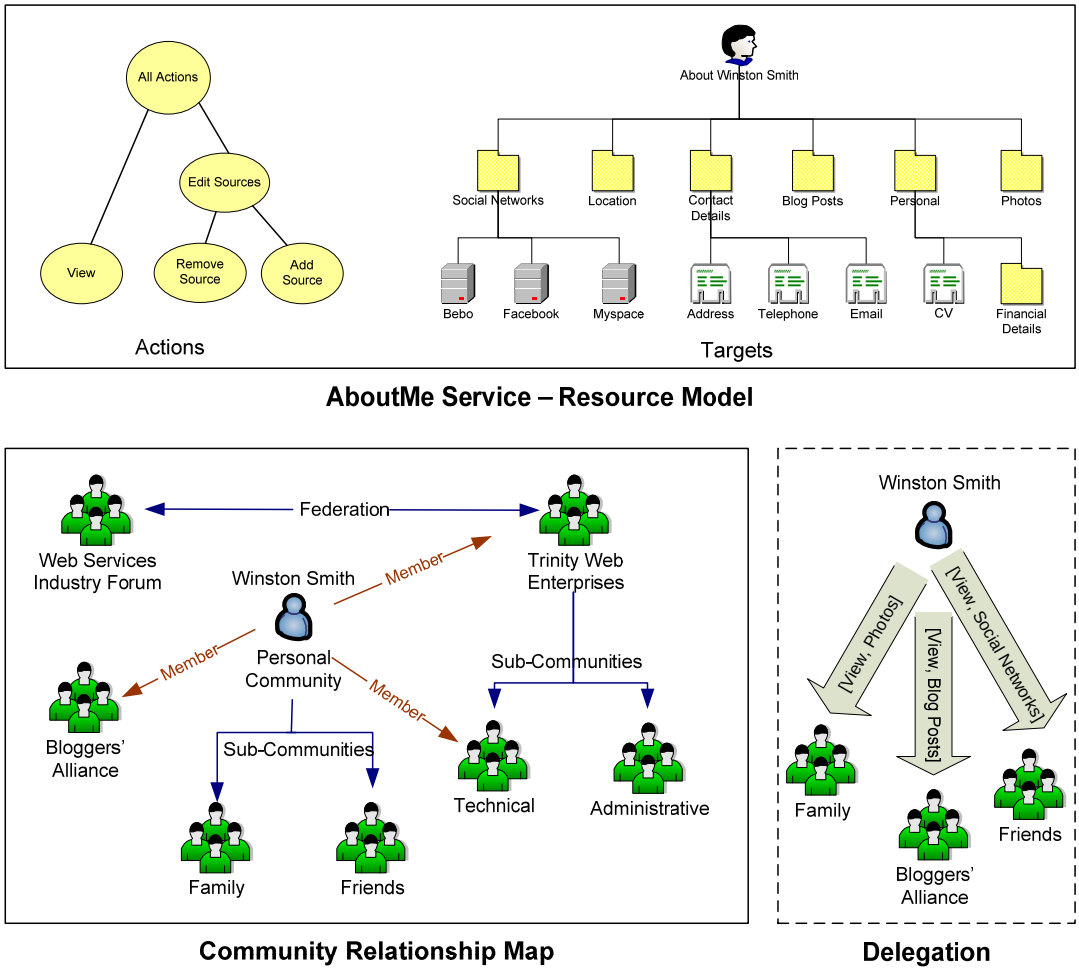

Figure 1. The CBPMS model of the AboutMe service, with capabilities sub-divided into a hierarchical resource model for each user. Pictured below is an example of a user's relationships in the community map and three examples of service-capabilities being delegated.

Because each part of the community map is managed by the user or group that it represents, each community can create its own individualised view of the world, while also interacting with communities defined by others. Each segment of the community map is itself a service and, like any other service, its capabilities can be delegated to other communities. This enables the formation of collaborations with complex governance relationships. Each community can own services and can create its own policy rules to apply to the services that it owns or has been delegated. Thus, we can allocate ownership of a personalised *AboutMe* service, through the CBPMS, to each community. Each community can then make this service available to third-parties with policies specifying precise constraints on how its capabilities may be used. For example, an individual may be happy to delegate their personal

information, contact details and CV to a job-matching service, but only allow their close friends to view the detailed history of their online purchases and social networking activity. Third parties can add their own policy rules to the capabilities they have been delegated, specifying further constraints on what information will be available to their users. This decentralised policy authoring model allows each individual and group to exercise fine-grained control over their “owned” services.

We now turn to a discussion of the CBPMS architecture and how its design supports extensible and flexible PBM capabilities. This discussion is accompanied by examples drawn from our work integrating the popular Drupal web application framework with the CBPMS.

CBPMS Service Oriented Architecture

Due to the heterogeneous nature of the services provided by web-application frameworks, we must assume that they will have varied requirements in terms of what policies need to be expressed, what contextual information these policies depend upon and how the policies are specified. Therefore, flexibility, extensibility and the ability to adapt policy specification approaches to the semantics of particular services are desirable. The CBPMS architecture supports flexibility by decomposing the policy system into a number of component services, as shown in figure 2.

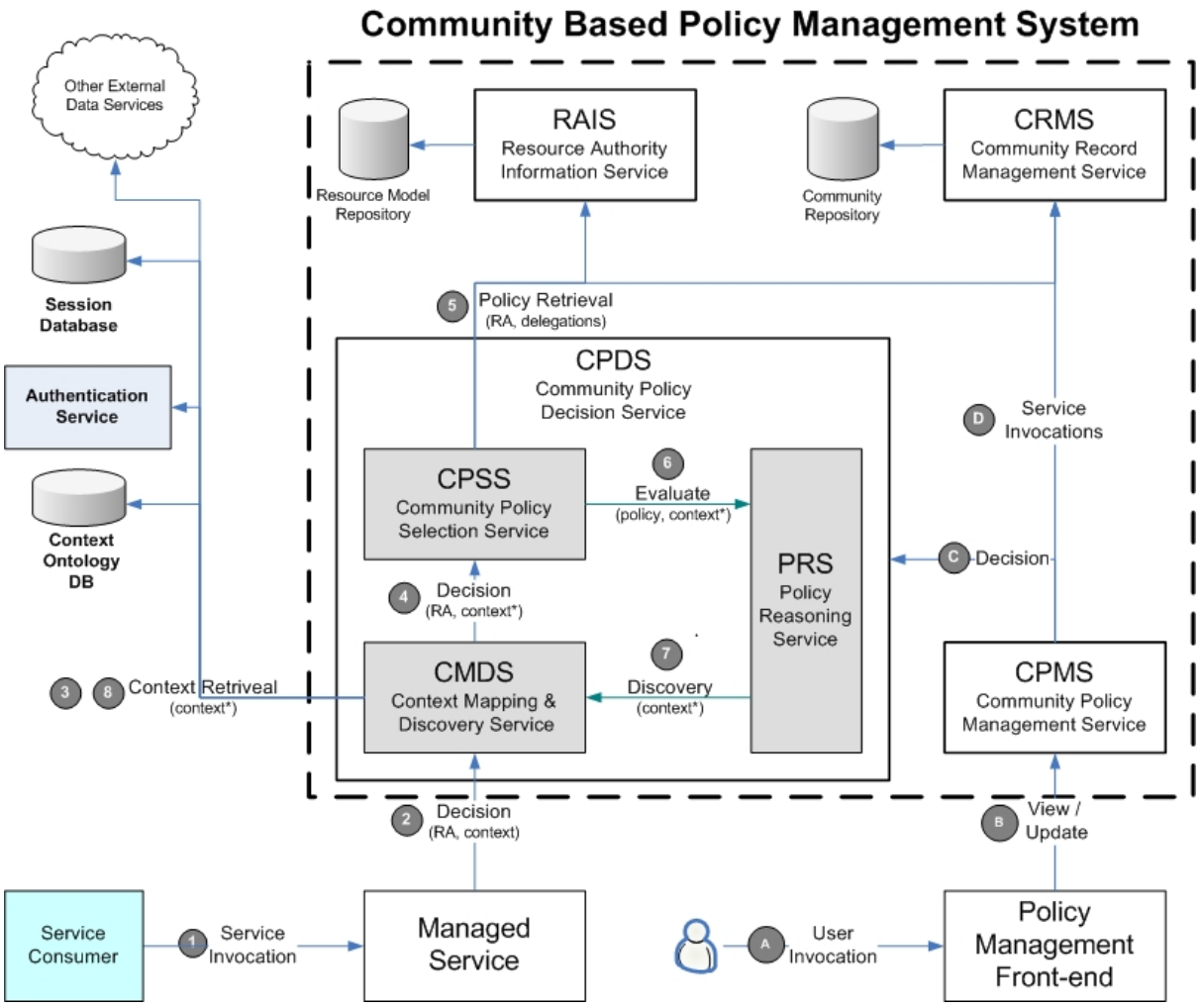

Figure 2. CBPMS service-oriented architecture. The arrows numbered 1 to 8 show the sequence of service invocations in providing a policy decision to a managed service (the state information exchanged is in brackets). The arrows marked A to D show the sequence of invocations in providing a policy management service.

Community Record Management Service

The CRMS maintains the system's community map, ownership and delegation records and policies. The map may be spread across multiple, independently managed CRMS servers. It provides 13 operations to allow individuals and groups to manipulate their part of the map, add policy rules and delegate authority. The operations were originally derived from long-term observation of the dynamics of internet communities and are designed allow the system to flexibly capture highly dynamic relationships, including major changes such as mergers and splits.

Community Policy Management Service

The CPMS serves three basic functions. Firstly, it enforces access control policy on the CRMS operations. Secondly, it provides a REST interface, with PUT and GET operations, and a common XML interchange format defined by the CBPMS XML Schema. Thirdly, as the community map may be distributed, the CPMS manages the routing of requests and assembles views of the community map across multiple, distributed CRMS servers.

Resource Authority Information Service

The role of the RAIS in the CBPMS architecture is to provide information pertaining to each service's hierarchical resource model. Technically, a resource model is a product-ordering of the Cartesian product of a set of partial orders, each of which corresponds to a dimension on which the managed service can be sub-divided into capabilities. In less technical terminology, each service is modelled as a set of trees and a resource authority is formed from a single node from each tree. Resource authorities are the units of delegation. They represent authority to access the capabilities represented by all the nodes on the trees beneath them. The RAIS provides the following function which the CBPMS uses to compare capabilities as part of its decision algorithm.

```
boolean result = impliesAuthority (ResourceAuthority ra1, ResourceAuthority ra2)
```

Two examples of the function being invoked for our *AboutMe* service and their results are shown below.

```
impliesAuthority ([View, Social Networks], [View, Facebook])  => true

impliesAuthority ([Edit Sources, Social Networks], [View, Contact Details])  => false
```

The RAIS logically encapsulates all information about the structure of the managed service. This allows the *impliesAuthority* function to be provided by the managed service itself. Many services have dynamic capabilities and their models need to be dynamic to reflect changes to the functionality offered by the service. For example, if we use the *addSource* function of our *AboutMe* service to add a new source of personal information to our profile, we need to add another node to the target tree of the service's resource model. By implementing the *impliesAuthority* function itself, the *AboutMe* service provider ensures that there is no

possibility of the resource model losing synchronisation with the service's capabilities while also retaining direct control of how the service is sub-divided into capabilities.

However, in many other cases, services are sub-divided on relatively static dimensions and can be adequately captured by static models or simple algorithms. In these cases, there is no real advantage in requiring the provider of the managed service to go to the trouble of implementing a new function. All that is needed is an adequate means of representing the hierarchy of capabilities and an implementation of the *impliesAuthority* function. As part of the CBPMS implementation we have provided an extensible RAIS server supporting a range of representation approaches. It provides several generic types suitable for describing a wide variety of service sub-divisions and these can be extended through inheritance. As an example, one representation supported is the XGMML graph interchange standard. We can use it to specify the *AboutMe* action tree as follows:

```
<graph directed="1" id="42" name="AboutMe Actions">
<node id="1" label="All Actions"/>
<node id="2" label="View"/>
<node id="3" label="Edit Sources"/>
<node id="4" label="Add Source"/>
<node id="5" label="Remove Sources"/>
<edge source="1" target="2"/>
<edge source="1" target="3"/>
<edge source="3" target="4"/>
<edge source="3" target="5"/>
</graph>
```

Community Policy Decision Service

The CPDS is the CBPMS service that managed services invoke directly. Its interface consists of a single function, specified as follows:

```
result = decision(resource_authority, context)
```

In order for a service to be managed by the CBPMS, it must invoke this function whenever it faces a choice that is driven by policy. The CPDS then evaluates the policies that apply to the service in the given context and returns a decision. Since the CBPMS is policy-language neutral, the owner of the service can choose whatever policy language or encoding is convenient for expressing decisions for their service. As part of our Drupal implementation, we have developed a PHP client which provides convenient access to the CPDS. A snippet of PHP code, showing an example of using the CPDS to provide access control decisions to our *AboutMe* service, is shown below.

```
$AboutMe = new AboutMeListing($user_id);
$pc = new PolicyClient(“http://chewy.cs.tcd.ie/cpds.php”);
$pc->setAccess(“View”, “CV”);
$pc->setContext(“user_id”, $user_id); //the users id is considered context
$result = $pc->decision();
if($result->isPermitted()){
       $AboutMe->includeData(“CV”);
}
```

The CPDS is decomposed into three distinct services in order to facilitate the easy integration of new functionality into the system. The following sections describe these services in turn.

Policy Reasoning Service

One of the design goals of the CBPMS architecture was to enable the management of policies expressed in a wide range of specification languages and to allow new policy specification languages and reasoning engines to be added seamlessly into the system. This goal has been achieved by encapsulating the actual evaluation of policies into a Policy Reasoning Service (PRS). The rest of the component services within the CBPMS treat policies and policy decisions as sealed envelopes – the PRS is the only part of the system which needs to understand the semantics of the policy language. We can thus incorporate new policy languages into the system without affecting any other components, if they can provide the required service-interface, which consists of the following operations.

policyResult evaluatePolicy(policy, context)

boolean evaluateResult(policyResult)

boolean isSemanticallyEqual(policyResult1, policyResult2)

The *evaluatePolicy* operation tells the engine to evaluate the policy against the context and return a result. Policy evaluations may return results which signify that evaluating the policy did not produce a decision. For example, the XACML *NotApplicable* result signifies that no decision has been reached. The PRS provides the *evaluateResult* function to allow the CBPMS to distinguish such results from results that actually contain a decision that could be transmitted back to the consumer. The *isSemanticallyEqual* function allows the CBPMS to identify when two policy results are equivalent.

As part of our CBPMS implementation, we have developed a generic PRS platform which enables the rapid integration of new policy reasoners through extending simple templates.

Context Mapping & Discovery Service

A basic requirement of any policy system is that the entity evaluating the rules must have access to the context in which the decision is being made. Thus, when a managed service invokes the decision service, the system must ensure that whatever contextual information is required in order to evaluate the relevant policy rules is available. To give a simple example of what this means: if you write policy rules which depend upon a user's past actions, such as "three consecutive failed log-ins will result in the account being locked", your policy system must have access to information about the user's past actions – in this case whether they have previously failed to authenticate themselves twice in a row. What amounts to relevant contextual information, however, is very much dependant on the particular service being managed and the policy requirements of its owner. Some services may base all access control policy on the identity of the user accessing the service, while others may depend upon the state of complex, structured data drawn from a wide range of distributed services.

Not only do services' context models vary widely, but the context may depend on information of widely varying scope. Some of the required context may be application specific, depending on state information embedded in the managed service, while other aspects may depend on the wider environment and organisation in which the policy system is deployed. For example, in many environments, user ids and authentication certificates are managed by independent services and are relevant across multiple managed systems. To make context modelling as flexible as possible, the CBPMS does not define any specific context model - it provides support for the definition of application-specific context models, specified by consumers, and application independent models specified within its CMDS component.

As part of our CBPMS implementation, we have developed an extensible CMDS framework, which eases the integration of managed services by providing a suite of extensible templates, which can be composed to produce a context model suitable for each service's policy requirements. For example, we have developed CMDS handlers which add support for sessions, database queries and several user authentication and identification schemas. They serve as building blocks from which a structured context model can be dynamically composed. When the CMDS receives a request from a consumer, it launches context mapping handlers of the types associated with the service. These handlers integrate the

system-wide contextual model into the consumer's context map to provide a custom context model tailored to the needs of each application.

There are many situations in which it may not be desirable for the policy system to attempt to fully populate the context model in advance of policy evaluation. Certain rarely-evaluated policies may require access to large volumes of information and it may not be practicable to populate this context in every situation. The contextual information that some policy rules depend upon may also only become clear when the rule is actually evaluated. Therefore, the CMDS also provides a *discovery* function to allow policy reasoning engines to discover information when some required contextual information is missing.

Community Policy Selection Service

The Community Policy Selection Service (CPSS) implements the core policy selection and conflict resolution algorithms that the CBPMS depends upon. It serves as the hub of the architecture, orchestrating the distributed component services that are involved in reaching a decision. It uses the CRMS and RAIS to identify which policy rules in the system are relevant to any particular service invocation. It first retrieves policies from the community that owns the service, and sends them to the PRS to be evaluated. If one or more of the evaluated policies produces a decision, according to the *evaluateResult* function of the PRS, the search ends. Otherwise relevant policies are retrieved from communities that have been delegated a resource authority for the service and sent to the PRS for evaluation in turn. The process continues until a decision is produced or no more delegations or policies remain to be evaluated. This algorithm is responsible for a key feature of the CBPMS model – policy precedence mirrors the delegation chain for each service. The system takes a pragmatic approach to avoiding cyclical delegations and network failures, by allowing policies to be specified which limit delegation chains or define timeouts.

The CPSS is also responsible for the resolution of policy conflicts. An example of a policy conflict, would be if we specified a *deny* access control policy to apply to the *[View, Personal Details]* capability of the *AboutMe* service and a *permit* access control policy to apply to the *[View, CV]* capability. The CPSS uses the *isSemanticallyEqual* function of the PRS to identify such conflicts and it allows a wide range of algorithms to be applied to resolve them – particular communities, policy rules and delegations can all be given relative precedences.

If no conflict resolution algorithm is specified, the CPSS can identify the community in which the conflict must be resolved and alert its owner.

Policy Management Front End

The CBPMS is composed of the services described above and does not include a user interface. Nevertheless, the viability of the system depends, to a large extent, on the availability of tools to help users and service managers to translate their high level goals into concrete policy rules, while helping them to understand the consequences of policy changes. Presenting such potentially complex information to users in an accessible way is a difficult problem. To address it, we have taken a two-pronged approach. As part of our Drupal integration, we have developed a module which provides a graphical map of the chain of delegations of each service. Management policies for each community are set through Drupal's standard management interfaces, using templates consisting of checkboxes and other simple user interface elements that map to parameterised rule sets. This allows us to progressively increase the complexity of the choices offered to trial-users and to monitor how this impacts upon their use of the system.

In parallel with our Drupal development, we have been exploring the user interaction concerns of more complex rule authoring. This research is based on creating prototype models using the Graphical Modelling Framework (GMF) on the Eclipse platform to perform user-centred studies using the underlying CBPM services and data, but with different direct graphical manipulation designs to capture the complexities of policy authoring. Ultimately we aim to port the more successful designs to a rich web application interface using Ajax and advanced graphical capabilities, such as those offered by the 'Processing' graphics library from processing.org.

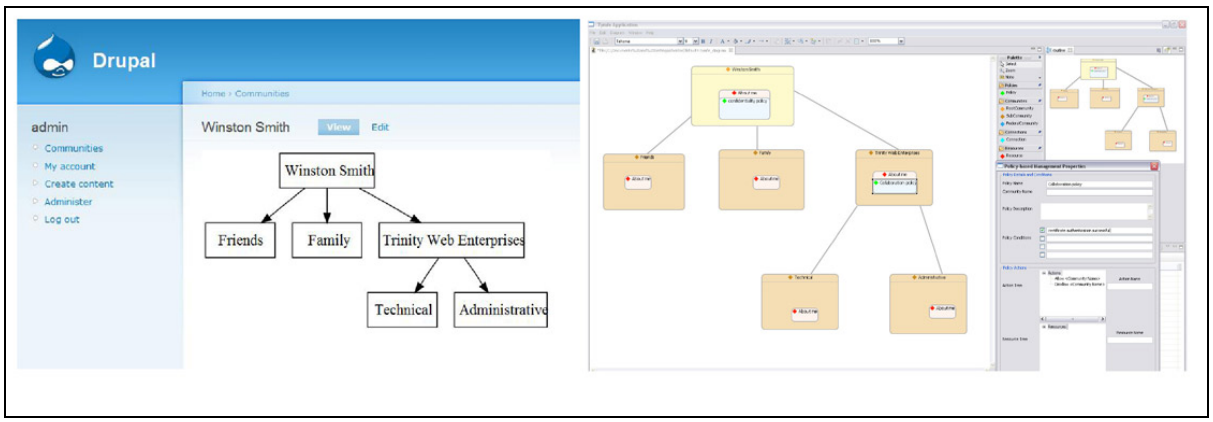

Figure 3. Twin-pronged approach to HCI, with the simple graph interface of our Drupal module pictured alongside the Eclipse-based GMF model prototyping system. Both depict the same underlying delegation chain.

Conclusions & Future Work

This article has described how the CBPMS can be employed to provide a flexible and extensible framework for the dynamic management of user-services provided by web application frameworks. We have developed a fully-featured CBPMS system and integrated a range of services which are driven by CBPMS policy decisions. Work on integrating the Drupal web application framework with the CBPMS is proceeding through progressively examining how the addition of various plug-ins, new services and new more complex policy authoring forms to the system affects its resource model, with the aim of developing an extensible suite of service resource-model templates. This platform is a test-bed for the evaluation of the limits of management decentralisation. We are also examining how semantic enrichment of resource models can be used to model and help constrain the dynamic nature of service capabilities and context models.

Acknowledgements

This work is partially supported by Science Foundation Ireland under Grant 03/CE3/1405 as part of the Centre for Telecommunications Value Chain Research (CTVR).

Links

More about the CBPMS: http://kdeg.cs.tcd.ie/cbpms

References

[1] Story, L. and Stone, B. "*Facebook Retreats on Online Tracking",* New York Times, November 30, 2007 retrieved from http://www.nytimes.com/2007/11/30/technology/30face.html

[2] Boutaba, R. and Aib I. *"Policy-based Management: A Historical Perspective",* Journal of Network and Systems Management, vol. 15, no. 4, November 2007, Springer Netherlands.

[3] Moses, T. (Ed), "*eXtensible Access Control Markup Language (XACML), Version 2.0"*, OASIS Standard, 1 Feb 2005. Retrieved from: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[4]Damianou, N., Dulay, N., Lupu, E. and Sloman, M.: "*The Ponder Specification Language*", Proceedings of . Policy 2001: IEEE Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 18-39.

[5] IBM, BEA, Microsoft, SAP, Sonic Software, & VeriSign. (2004). "*Web Services policy frame-work (WS-Policy)*". Retrieved from http://www-128.ibm.com/developerworks/library/specification/ws-polfram/

[6] Della-Libera, G. et al. (2002). *Web Services security policy language (WS-SecurityPolicy).* Retrieved from http://www-106.ibm.com/developerworks/library/ws-secpol/

[7] Brown, W., Moore, G., Tegan, W. *"Soa Governance – IBM's approach"* IBM White Paper, August 2006. Retrieved from ftp://ftp.software.ibm.com/software/soa/pdf/SOA_Gov_Process_Overview.pdf