POWER Prototype: Towards Integrated Policy-Based Management

M. Casassa Mont, A. Baldwin, C. Goh,
{ mcm,ajb,cng}@hpl.hp.com

Extended Enterprise Laboratory,
Hewlett Packard Laboratories Bristol,
Stoke Gifford, Bristol BS34 8QZ
United Kingdom

Abstract: A policy-based management system is only really useful if it allows not only high level description of abstract policy, but also enables such policy to be refined and eventually mapped into an appropriate configuration for controlling devices in the managed system. Such a full integration has only been discussed in the literature but not realised so far. Our approach, implemented as the *POWER* prototype, demonstrates a way towards making it a reality in practice.

keywords: policy, management, refinement, template, model.

1. Introduction

The use of policy for the management of information network and system has become very popular since the early days of the pioneers such as [Masullo93, Moffett93, Wies95]. Nowadays, many commercial products such as Checkpoint's Firewall-1, Axent's Enterprise Security Management, and Hewlett Packard's DomainGuard all use some form of policy to configure and control the way they function. In the IETF, there is an active Policy Working Group [IETF-policy] which aims at resolving issues related to policy-driven network QoS. All of these show the recognition of the importance of this approach in IT management.

Practical implementations of policy-based management are, however, mainly at a low level at present. Little has been found to bridge the gap between "business level policy" and "device level configuration information" (paraphrased) first expounded in [Wies95, Heiler96], and subsequently re-interpreted in [Goh97], and will be referred to as *policy refinement* henceforth. The **POWER** prototype –

POlicy **W**izard **E**ngine for **R**efinement is an integrated policy authoring environment developed as a realisation of the concepts in these previous works.

1.1 Motivation

The work presented in this paper was motivated by the desire to find a solution for the seemingly intractable problem of transforming an abstract policy to implementable configuration. In addition, we also find that most policy description languages are aimed at the technical operator. This does not go well with several classes of people who interact with policies, and in particular, those who make policies, but are not technically orientated [Goh98]. This, in itself, also spurred us into this work.

Although we started from the perspective of security management, it rapidly became obvious that the same approach is applicable in IT management in general, including network QoS management. This paper should therefore be read with such generality in mind.

1.2 Outline of paper

The rest of this paper is in four sections. Section 2 is an overview of the specific problem we set out to tackle, our design objectives and the philosophy underlying our approach. Section 3 is a detailed explanation of the design of the prototype, and in section 4, implementation details including screen captures as illustrations. Section 5 concludes with a summary and list of future work.

2. Overview

2.1 Issues and design objectives

The concept of policy hierarchy and transformation process presented in [Wies95], and later in [Neumair96], were the few fruitful attempts to implement the abstract policies referred to in [Moffet93, Masullo93]. The dichotomy of "human orientated" abstract policy description and machine executable configuration is still bridged by human intervention, transforming a policy from one description language to another. Often, this process takes place more than once, giving rise to the following issues:

- the human operator must have deep understanding of both the business level policy *and* domain specific knowledge such as security or network QoS;
- it is hard to check the accuracy and consistency of transformation carried out by the human operator;
- a policy author can only construct a policy by using accurate syntax in addition to having precise semantics;

- the human input must be compiled and interpreted to produce an output which is domain specific;

From these issues, we have chosen a subset to work upon, and they became our initial design objectives:

- A system should allow most, if not all, classes of policy users as categorised in [Goh98] to easily interact and understand the meaning of the policy.
- The business-driven policy maker (referred to as the *consultant* later*)* should be shielded from the need to have *deep* domain-specific technical knowledge not relevant to that person's main job.
- Using the same system, a business level, or abstract, policy can be expressed as easily as the device level, or configuration, policy.
- At all level, the policy maker can concentrate on the semantics of policy *without* being forced to be distracted by the need of syntactical precision.
- A business level policy can be refined in such a way that the policy author can, at some point, actually *deployed* it directly in devices without leaving the authoring environment.

We will review the extent to which we come close to these design objectives at the end.

2.2 Philosophy and approach

A policy can be most generally looked upon as "the *constraints* and *preferences* on the state, or the state transition, of a system, and is a guide on the way to achieving the overall objective which itself is also represented by a desirable system state" [Goh97]. Developed from this rather abstract rendering, we began to use the concept of constraint for the components in policy description: the context in which a policy operates; the triggering event which kicks policy into consideration in the context; and the policy statement. We will not consider events in this work, but deal with the other two components using a constraint-based approach. The refinement of policy, therefore, consists of two aspects: the refinement of policy context by making constraints more specific, and refinement (constraining) of objects used in the policy. This philosophy and approach is reflected in our prototype and will become clearer with the example given in section 0.

3. Prototype design

In order to deal with the user-related aspect of the design objectives, we created certain concepts missing in present literature. Based on these concepts, we are able to design an architecture to support our goals.

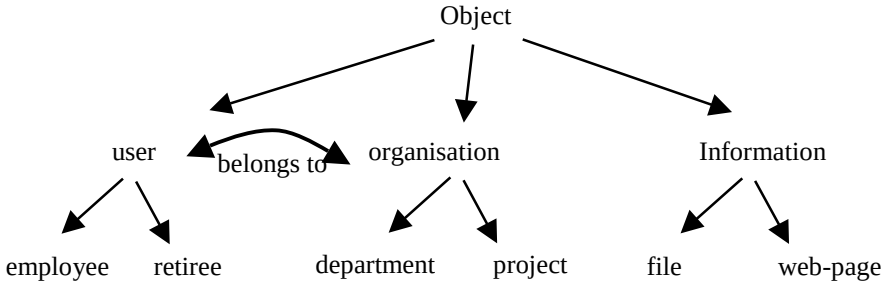
3.1 Prerequisite concepts

3.1.1 Expert and consultant

Present literature related to policy management has a good understanding of the concept of refinement [Wies95], even to the extent of identifying the user category of these policies at the various stages of refinement [Goh98]. However, a vital separation of responsibility has not been brought out explicitly: there are two types of policy making person whom we respectively call *expert* and *consultant*. The "expert" is the person with deep domain knowledge, such as that in the field of security, or network QoS related mechanisms. A "consultant" is the person who has deep knowledge of the business for which policies are to be established.

The expert deals mainly with *policy function to mechanisms mapping* and the consultant mainly deals with *business to policy function mapping*. With this separation, we can envisage the creation by the expert of *policy templates*, which will be used by the consultant to create policy according to the business needs. This process has so far been largely glossed over in template driven tools found in existing products, because templates for policy related management have been created for convenience rather than due to a deeper need for expert knowledge embodiment.

3.1.2 Information and system model (ISM)



Association: "***user*** belongs to ***department***"

Figure 1 Example of object hierarchy and object associations.

Modelling in IT management is old hat, but hitherto this concept has not been greatly exploited in policy refinement. We have created an *information and system model* (ISM) in which all policy related information is modelled and stored. The ISM essentially models objects and their relationships, which can be hierarchical inheritance or associations. This is a subset of the full model of the managed system, only sufficient for enabling policy refinement. The model covers non-system level objects to include concepts such as roles and organisations, and indeed anything else needed to describe a policy. See fig. 1 for example. The system related

part of this model links into the actual managed system, and through the interaction with the managed objects, it would be possible for policies to be deployed.

3.1.3 Domain dynamicity

The concept of domain is very powerful in describing a given space of operation, and is particularly useful for discussing policy [Sloman94]. However we have found it necessary to add dynamicity to this concept, i.e., using constraint-based description that combines with managed system model gives us a much more powerful and timely determination of the applicability of policy. We call this dynamic domain the "context" for the policy. As for dynamic sub-domains, we can find an example in the form of context that help to determine the setting off of an event trigger in case of an event. Clearly, a degenerate case of this concept will be the fixed domain idea that is well understood already in this field.

3.2 Architecture

The architecture of our prototype is shown in figure.2.

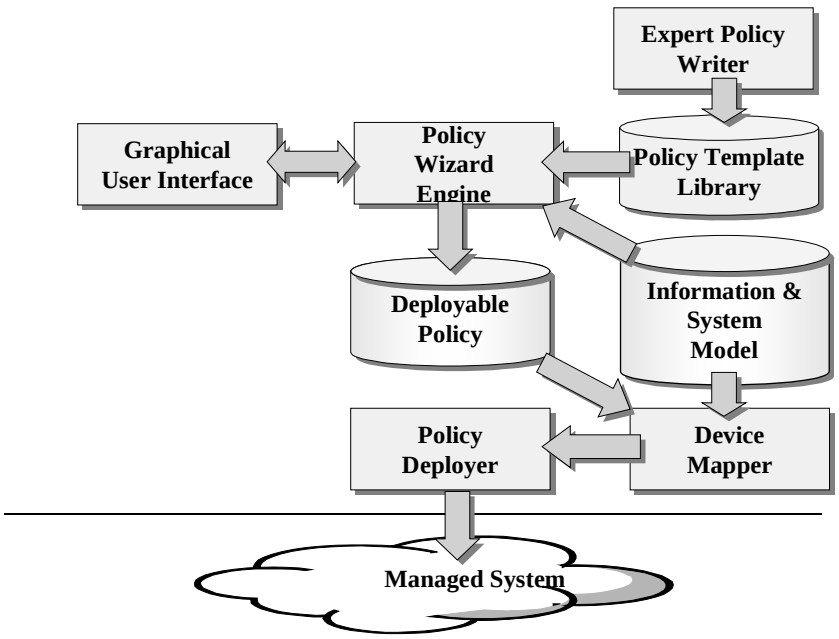


Figure 2: Policy Authoring Environment Architecture

There are six key components in this architecture.

3.2.1 Policy Template Library (PTL)

Use of template for policy is common [Wies95], but existing examples only fulfill a "form-to-fill-in" function. The main goal of our policy template is to store a generic policy description that provide information about its refinement to the Policy Wizard Engine (see section 3.2.3). For example, an abstract policy template "*people* can carry out *some operation* on *specific information*" has enough embedded information to be refined as: "*engineers* can *add entry* in a *database* that belongs to the *department*". This is achieved through referring to objects that are defined and described in the ISM. For example "people", "operation" and "information" are ISM concepts. The Policy Wizard Engine manipulates the information stored inside a template yet masking its complexity so that the consultant will have a "human readable" view of the information.

The PTL is a collection of *policy templates* which have been created by the expert of the domain that the policies are meant for. Each template is a "package" that describes the policy according to certain principle, and the way a consultant, using the authoring environment, can refine it. This is achieved through embedding the refinement steps and instructions in the template as components.

The policy template is implemented as a Prolog "fact" and can be manipulated by the Policy Wizard Engine. Its components can be classified accordingly to their usage:

- Policy Statement: The description of the policy. These are predicate logic statements with several views, one of which is "natural-language like" and is exposed to the policy user.
- Policy Context: The description of contextual constraints within which the policy will operate. The contextual information allows one to arbitrarily define a domain *dynamically* within which the policy statement is valid.
- Informational components: they provide extra information to the policy user. For example the "abstract" and the "description" contain descriptive text about the meaning of the policy.
- Procedural components: they have embedded process instructions used to drive the "refinement flow". For example the "sequence" component defines the steps the Policy Wizard Engine will lead the consultant through.

Both the policy context and policy statement can be expressed as logical predicates with AND, OR, NOT as constraints and conditions.

3.2.2 Information and system model (ISM)

As mentioned previously, the ISM models the information in the underlying environment we want to manage using policy. The concept used here may easily be implemented using the Common Information Model from the Distributed Management Task Force CIM [DMTF-CIM], with extensions in the area of business and organisation model. Included with the implementation will be the low-level

linkage of object classes to information sources that creates the mapping to managed objects. In our prototype, this is implemented as a set of Prolog statements that can be easily accessed by the Policy Wizard Engine.

3.2.3 Policy wizard engine (PWE)

The Policy Wizard Engine is the heart of the Policy Authoring Environment. It is the combination of:

- A Prolog inference engine.
- An interpreter that manipulates a policy template according to the embedded information, and provide support to the graphical user interface.
- A module that interacts with the ISM using a defined API.
- A module that deals with "deployable policies".
- Procedures that interact with the "Policy Deployer" using a defined API.

At start up, the PWE will load policy templates from the library. Through the use of a GUI, a relevant template can be selected, and by interpreting the embedded information in the template, the PWE will guide the consultant in the refinement process to ensure that:

- within an abstract policy, objects, which can be made more specific through the selection of its sub-class, can be so specified;
- legitimate additional constraint can be included as contextual information.

At the end of the refinement process the PWE will save the policy either for further refinement later or for it to be used in deployment.

3.2.4 Graphical User Interface (GUI)

The Graphical User Interface is an important part of the architecture. It hides the low-level policy details, such as the policy template infrastructure and Prolog programming language, from the consultant in order to present an easy and simplified way to access the system functionality. The consultant is always in control while using this highly interactive GUI, but is restricted by what is legitimate, a constraint established by the combination of the expert's input and the information modelled by the ISM. In this way, the consultant will not be able to stray from the permissible state of the managed system. The errors that will result from the policy refinement process will be concentrated in a smaller number of the POWER system and can be more easily rectified.

3.2.5 Deployable policies database

A policy is deployable only when, through the use of the ISM, a set of real world system objects can be found and for which configuration specified. The system stores those policies in order to perform two possible future activities:

- to be uploaded by the "Policy Deployer" and be deployed;

- to be available to the consultant or other system modules for further manipulations.

While the Policy Template Library is a knowledge base largely independent of the underlying system, the policies in this database have hooks to the real word by referring to entities described in the “Information System Model”. Moreover, the “deployable policies” must be “understood” by the “Policy Deployer” in order to be really deployed in the real word. Depending on the implementation, it is possible to create only a half-way house database for “refined policies” and for further policy manipulation alone.

3.2.6 Device mapper

The crucial step in making policy refinement an integrated process is to be able to pass the information stored in the refinement policy to a component that can transform automatically the information into configuration details. The device mapper is such a component, capable of using the information contained in the ISM to convert from a policy description in the form of a policy statement and context containing variables into a series of system specific function calls. Clearly the algorithm used is very dependent on the device to be configured.

The example we used in the prototype is access control configuration representing the relationships between users, operations and resource objects that are to be secured. A way to implement it in the common operating systems such as NT and Unix OS will be for the mapping function to create groups of people who have the same ability to access particular objects. The policy mapper starts by evaluating the context with respect to each resource type so that a list of users can be identified for each identifiable resource. These user lists are then combined to form group definitions.

Using the linkage in the ISM where the managed system information can be obtained, we are able to bind the unbound variables in the refined but still abstract policy, such that the users corresponding to individual resources can be identified. This data is reorganised into a set of group definitions and members. The process is then optimised through heuristics to remove replicated groupings. The resource enumeration carried out to generate the group structures can then be revisited and the given groups are matched against the operations being performed on each resource. This thus provides a number of entries in an access control list for that policy.

3.2.7 Other components

Not all components of the architecture outlined in the fig. 1 has been implemented. They include the following:

- Expert Policy Writer. Given that a policy template is not only an abstract representation of context and policy statement, it is also a directive for the refinement process, the expert needs a good authoring environment in order to

create such templates. As it is not the most important part of our prototype for demonstrating our concepts, it has been delayed and will form part of future work.

- Policy deployer. This is a unit that depends entirely on the network and management system. Depending of the domain, it could be a mechanism in HP's OpenView IT Operation product, or just the depositing of information in a directory for DEN (directory enabled network).

It is our belief, however, that a usable system must include these two components which we would work on in the future.

4. Implementation

In our implementation, we created a prototype based on a security policy refinement scenario, which allows the steps in a refinement process to be carried out.

4.1 Scenario

As mentioned previously, we started working on this problem with a scenario of security related management and found it equally applicable to other domains. The scenario is expressed as follows:

> *We have a system containing objects that belong to different users, how do we create a policy through refinement to control the access of these objects based on the attributes of the organisation, roles and user identity?*

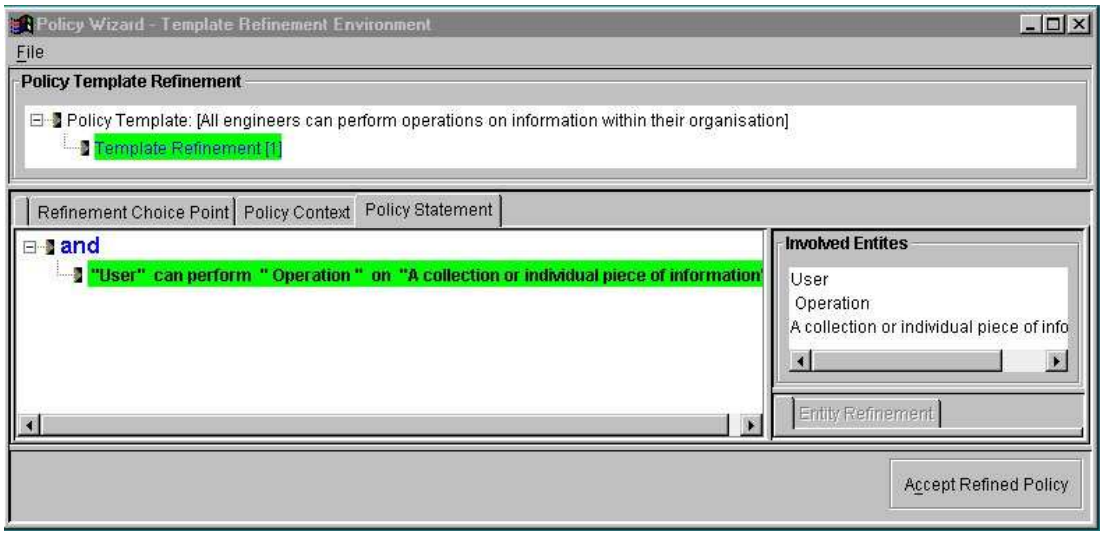


Figure 3: Policy Statement

4.2 Prototype views

We created by hand a set of policy templates which are accessible by the PWE, and an information base to represent the data in the ISM containing hierarchies of classes of object and associations of objects. With such information, we are able to use our prototype, which is made up of the components described in the architecture, to provide the following functions to the *consultant* via the GUI:

- Selection of policy template set using either keyword combinations or policy categories, thuskicking off the refinement process. Fig. 3 shows the beginning of the refinement of a policy "All engineers can perform operations on information within their organisation.". The policy statement, found in the lower left panel, highlight the objects that could be refined: "User", "Operation" and "A collection or individual piece of information". They represent objects of the managed domain, and are linked directly to the ISM and can be made more specific.

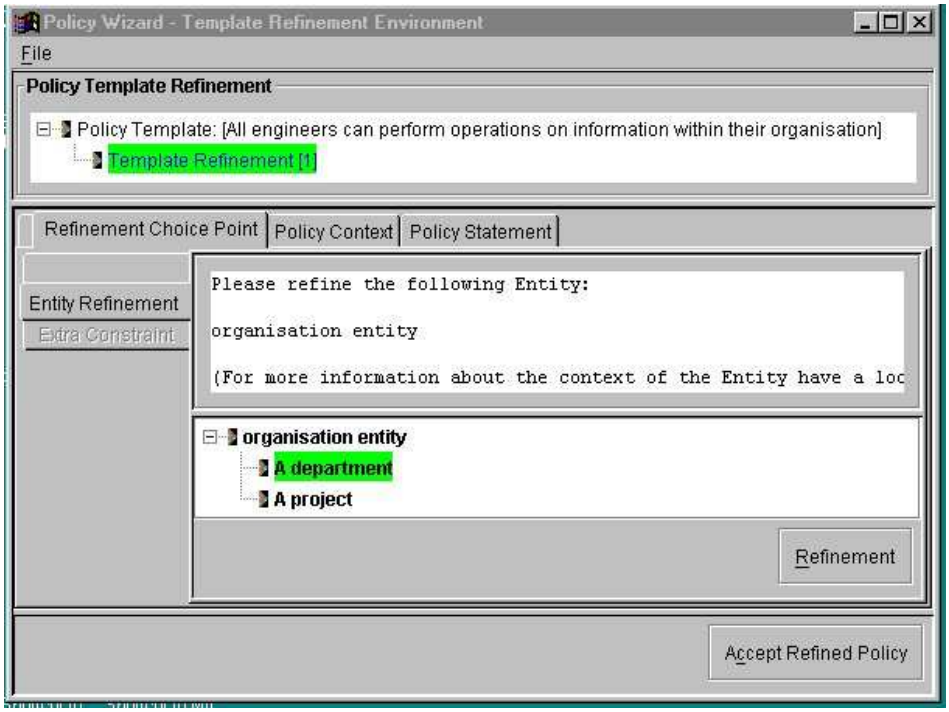


Figure 4: Object Refinement suggested by the Policy Wizard Engine

- Refinement through object subclass selection suggested by the PWE. The policy we have at this stage is very generic and frankly, rather useless. By allowing ourselves to be guided through the "Refinement Choice Point" panel, as shown in fig.4, which shows *organisation* as the object to refine, we can

choose the appropriate object subclass for the policy using the bottom panel. Here, the object hierarchy tree—with “department” and “project” as example of subclasses—is presented.

In this way, each object that can be made more specific will be presented one at a time for the consultant to consider, until all objects are covered.

- Refinement of context suggested by the PWE. As mentioned previously, the refinement process may be carried out according to object and according to context. Context refinement is, as per our prototype, the addition of constraint to form a conjunction with the existing constraints. See fig. 5.
- The consultant can, at any point, carry out either form of refinement, even though a methodical approach following the embedded process in the template is recommended. When finished, the consultant will be returned to the beginning to construct another policy from the template, or ask the system to “deploy”. The deployment is done behind the scene, with the output in the form of a configuration file that indicates the completion of the policy-to-configuration transformation process.

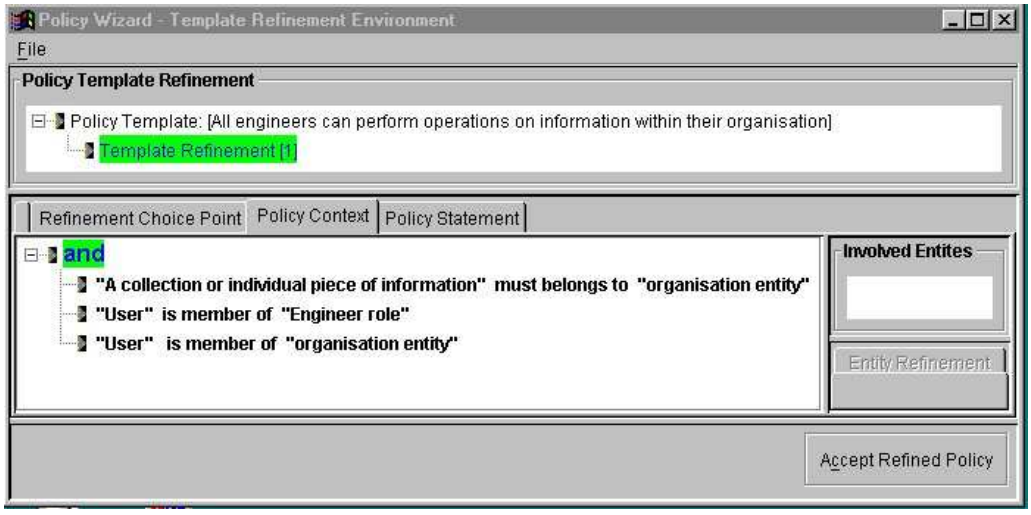


Fig 5: Refinement by Context

4.3 Policy template

The policy template supplies the cleverness that the PWE can use to guide the consultant. Shown in fig. 6 is an example of a policy template that is presented through the GUI in the last section. The example is self-explanatory, in that the Prolog facts include the necessary header information such as “keywords”, “category”, “abstract” and so on.

```
template(t3,
        [[ c0, keywords, [$engineer$,  $information$, $organisation$ ]],
         [ c1, category, $Access to Information$],
         [ c2, abstract, $All engineers can perform operations on information within their organisation$],
         [ c3, description, $Users that are Engineers can perform operations on  \r\n information that belong to\r\nthe same
              organisation they belong to].$],

         [ c4, expiration-date, $01/01/1999$],
         [ c5, deployable, $deployable$],
         [ c6, start, c7],
         [ c7, sequence, [c8, c12, c13, c16, c18]],

         [ c8, context, [internal: [and([belongsTo(information,orgUnit(U)),
                                  isMember(user(Un,UId), engineer),
                                  isMember(user(Un,UId), orgUnit(U))])],
                                  refinementBy: [[information,c10],
                                                 [orgUnit(U),c10]]]],
         [ c10, refinementDetails, [category: ism,
                                    condition: [],
                                    refinementBy: [class]]],

         [ c12, policyStatement, [category: deployable,
                                  internal: [and([canAccess(user(Un,UId), operation, information)])],
                                  condition: [],
                                  refinementBy: [[user(Un,UId),c10],
                                                 [information,c10]]]],
         [ c13, classRefinementChoice, [class: [orgUnit(U),c10]]],

        [ c16, constraintChoice, [constraint:[and([about(information,user(Un,UId))])],
                        choices:[accept:c18, ignore:c18]]],
        [ c18, end, []] ]).
```

Fig 6: Policy Template example

4.4 Policy mapper output

The policy mapper is includes an algorithm that transform a refined policy into configuration information. At the time of writing, the information is created and written to a text file. The information takes the following form:

```
// Policy  pol1  generates Access control functions
// functions take form Function( resouces, access,
Operation group/uers list
// Configuring resources[H69483,1,H54558]
setCalEntryACL([H69483,1,H54558],            create,
[europe1\ruby])
// Configuring resources[H69878,2,H54574]
```

```
setCalEntryACL([H69878,2,H54574],                create,
[europe1\jake])
// Configuring resources[H70273,3,H54590]
setCalEntryACL([H70273,3,H54590],                create,
[europe1\megan])
```

5. Summary and future work

The POWER prototype is an implementation that demonstrates the conceptual and philosophical intent of some of the predecessors working in the policy-driven management area. It shows for the first time a way to seamlessly integrate policy refinement with policy-based configuration generation. The strength of this prototype is the achievement of our design objectives, leading to:

- a departure from the "technician-centric view" to a "multi-user view" using the separation of responsibility for "expert" and "consultant", and enabling easy policy authoring;
- the exploitation of prevailing modelling paradigm to enable policy refinement, reification and transformation into device-meaningful configuration.

This prototype still leaves much to be worked on. In addition to implementing the missing components in the architecture, i.e. a expert template creation environment and a policy deployer linkage, future work includes, but not is not limited to, the following:

- Additional functionality in the Policy Template Language and Policy Wizard Engine:
  - fuller descriptive power for constraint specification;
  - events;
  - nesting or "template hopping";
  - management of iteration in which the same refinement process could be repeated more than once, for example, for all the sub-classes of an object.
- Additional modules to the architecture:
  - consistency and conflict analysis;
  - meta-policies management.

6. References

[DMTF-CIM] Distributed Management Task Force, *Common Information Model (CIM)* http://www.dmtf.org/spec/cims.html

[Goh97] Goh, C., *A Generic Approach to Policy Description in System Management*, Proceedings of the 8th IFIP/IEEE international workshop on Distributed

Systems Operations & Management (DSOM '97), Sydney, Australia, 21-23 October 1997.

[Goh98] Goh, C., *Policy Management Requirements*, Proceedings of the HP OpenView University Association Workshop (HP OVUA '98), France, 21-23 October 1997.

[Heiler96] Heiler,K,, Wies, R., *Policy Driven Configuration Management of Network Devices,* Proceedings of the IFIP/IEEE Network Operations & Management Symposium, Kyoto, Japan, April 1996, pp 674-689.

[IETF-policy] IETF Policy WG, *http://www.ietf.org/html.charters/policy-charter.html.*

[Koch95] Koch, T., Kra"mer, B., Rohde, G., *On a Rule Based Management Architecture*, Proceeding IEEE 2nd International Workshop on Services in Distributed and Networked Environments, Whistler, BC, Canada, June 1995, pp68-75.

[Masullo93] Masullo, M., Calo, S., *Policy Management: An Architecture and Approach*, Proceeding of IEEE 1st International Workshop On System Management, Los Angeles, April 1993

[Moffett93] Moffett, J.D., Sloman, M., *Policy Hierarchies for Distributed Systems Management*, IEEE JSAC Special Issue on Network Managemnet, Vol 11, No 9, December 1994.

[Neumair96] Neumair, B.; Wies, R.; *Case study: applying management policies to manage distributed queuing systems*, Distributed System Engineering, 1996, pp96-103

[Sloman93] Sloman, M., *Specifying Policy for Management of Distributed Systems*, Proceedings of the 4th IFIP/IEEE international workshop on Distributed Systems Operations & Management (DSOM '93) 1993

[Sloman94] Sloman, M., *Policy Driven Management for Distributed Systems*, Journal of Network and Systems Management, vol. 2 part 4, 1994, pp333-60.

[Wies95] Wies, R.,*Using a Classification of Management Policies for Policy Specification and Policy Transformation,* Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, CA, USA, May 1995.