Semantic Ordering in Policy-based Management

Andrei Majidian

British Telecommunications plc
pp HWH225
Virtual Postbox (HOM-NX)
PO Box 400
London N18 1XU
UK
Andrei.Majidian@bt.com

Abstract. Policy-based management has emerged as a solution to the problem of resource management in an open distributed environment. However, adding any new rule to a large distributed system could lead to conflict and thus the integrity and consistency of the rule database within the system is one area where a solution is still sought. This paper describes how, in a given policy based system, to compute the full extent of all possible rules and how, by considering the semantics of actions/verbs within the system, the action set can be ordered to form a partially ordered set (poset), an ontology. Using the ontology, rules can be expanded to reveal hidden conflict at compile-time - the algorithm for this expansion is provided in the paper. Without this expansion, the conflict remains hidden and only emerges at run-time. The paper also provides a mathematical method of computing the minimum set of rules that can represent any given rule set in a policy based system, i.e. a way of contracting a rule set. Finally, the paper discusses two (both static and dynamic) ways of implementing the semantic ordering in a policy based management system.

1. Introduction

Modern systems are dynamic and scalable i.e. they expand and contract in terms of the functionality and range of services they offer. As a system grows the functionality on offer increases and the interrelationships between the different functions become more complex. Different parts of the system may be owned by different entities. A global knowledge of the interrelationships between the functions within this dynamic system currently resides outside the system in the people responsible for it. This complexity has created a need for management platform that is capable of adapting to the dynamic nature of these systems. The policy concept has been gaining ground as a means of handling dynamic systems access management.

1.1 Policies

Policies are used to dictate a particular set of behaviours in a distributed system, without hard coding this behaviour into the structure of the system itself. This is often achieved through a policy-based management platform. These platforms provide a management language by means of which the policies can be expressed and stated to the system [3, 5], however, the user end of the language might be presented as GUI component. In access control policy management platform as well as policy-based networking [13, 14, 15] the policies are often sets of modal operators denoting permission (authorisation) and obligation akin to the notions defined in deontic logic [6]. The sentential syntax of these languages are simple structures of the form: subject, verb and object, stating what the subject can do to the object of the sentence. There is also additional information identifying the type of the sentence i.e. whether a permission has been granted to the subject or an obligation has been placed upon the subject (usually in the event of some condition occurring). There is also a statement of mode that would indicate whether the permission or the obligation has been bestowed in a positive or a negative sense. Of course, a fully fledged language may have other construct and sophistications depending upon its usage (e.g. notion of time or object orientation facilities).

In order to demonstrate the concept of semantic ordering, we have devised a simple generic language (Joey). Joey has been designed with minimum set of constructs (retaining only those language constructs that are useful for our discussion on semantic ordering).

2. Joey Language

Joey is a language for expressing policies. Rather than selecting an existing language with a multiplicity of features and then stripping away the extra features which were not a relevant part of the language, in order to demonstrate the concept of ordering verbs, Joey has been developed with just enough constructs to demonstrate the concept of Semantic Ordering (SO). The other reason is that Joey was designed with only the core concepts of a policy-based language, so that the concepts that are demonstrated here with Joey can easily be adapted into any policy-based language [2, 3, 11].

In Joey, a policy is a set of rules. Rules are separated by a semicolon and have the following construct:

Mode Type : SubjectSet VerbSet ObjectSet;

Mode is either **positive** or **negative**
Type is either **authorisation** or **obligation**
the SubjectSet and the ObjectSet are of the form
{subject1, subject2, …, subjectM} {object1, object2,…, objectM};
the VerbSet is of the form {verb1(para1, para2, paraK), …verbJ(para1,…,paraL)};
(para are parameters) where the type of the parameters is important rather than the value.

For example, following two rules are written in Joey:

```
negative authorisation : {alex, danny} {read()} {hamlet}

positive obligation : alex {send()} {hamlet}
```

2.1 Mathematical notation

If we collapse the syntax of the Joey and only consider its subject, verb and object [1, 3], we can consider policies as a sub-set of the Cartesian product of the subject, verb and object sets. Mathematically what is suggested can be expressed as follows: Let σ be the set of all subjects, υ the set of all verbs and o the set of all objects. Also let p be any arbitrary policy then $p \in \sigma \times o \times \upsilon$. Policies can be grouped together to form a set of policies. Thus, we can talk of the set of policies P where $P \subseteq \sigma \times o \times \upsilon$ for any element $p \in P$ where $p = (x, y, z)$, which means that subject x can do verb y on object z. The syntax of Joey provides a facility to group the policies together into a set of policies. We can denote this grouping as $P = \langle S, V, O \rangle$ which says that P consists of $|S| \times |V| \times |O|$ number of policies, e.g. (in positive authorisation mode) $\langle S, V, O \rangle$ means each element of S can do each verb in V on each element of O. Formally it can be stated as $\langle S, V, O \rangle = \{(s,v,o) \mid s \in S, v \in V, o \in O\}$

i.e. $\langle S, V, O \rangle = S \times V \times O$.

Notice that $\langle S, V, O \rangle \subseteq \sigma \times o \times v$.

The rule base management itself needs to be governed with meta-rules that will shape and create a framework in which the rules are issued. In the next section we will talk about some of these meta-rules.

The set of all of the singleton rules that can have as their subject an element from S, as their verb an element from *V,* and as their object an element from O, is $\wp(S \times V \times O)$, i.e. the power set of the Cartesian product $S \times V \times O$. Therefore, the total number of rules that can be defined (including the empty set $\varnothing$) and considering positive and negative mode and also considering both authorisation and obligation types is

$$\left|\wp(S \times V \times O)\right| \times 4 = 2^{|S \times V \times O| + 2} = 2^{(|S| \times |V| \times |O|) + 2}.$$

Obviously not all these rule combinations make sense and some of them will be contradictory. Some of the contradictions will be obvious and some will arise from the semantics of the rules. This paper considers a class of semantic conflict and demonstrates how to detect it automatically at compile time (i.e. statically) by supplying some extra information to the management platform.

2.2 Singleton Rule

A singleton rule form is defined as:
Mode Type : {singleSubject} {singleVerb} {singleObject};
Any rule can be converted to a set of singleton rules.
For example

```
positive authorisation : {Danny, Alex} {read()} {Ulysses};
```

can be converted to the two singleton rules

```
positive authorisation : {Danny} {read()} {Ulysses};

positive authorisation : {Alex} {read()} {Ulysses };
```

The number of singleton rules in the set of rules for a policy can be calculated as:

$$\sum_{i=1}^{n}\left(|S_i| \times |V_i| \times |O_i|\right)$$

Where n is the number of rules (before expansion to singleton rules) in the policy.

3.Imposition of a partial order on an verb set

In a rule based management system, the system can potentially contain a very large number of functionalities/verbs available. For an administrator who is responsible for managing the rule base, it might be cumbersome if not impossible to have the knowledge and awareness of all the functionality (i.e. verbs) within the system in which he is working, particularly given that the system is dynamic and subject to change. Furthermore, as we will show later, there are semantic interdependencies within the set of functionalities that the system offers. In this paper we are proposing how to imbue the system with the knowledge of the functions and their interdependencies. This will ensure the integrity of the rule database within the system and make sure the system is consistent.

For example, if we consider remote file access (FTP), one possible ordering that can be created is that shown in figure 1. In this example the verbs form a partially ordered set (poset). This paper proposes that though any graph imposed on a set of verbs is primarily a management policy (or rather a management meta-policy [4]), however any imposition of a hierarchy will result in a poset.

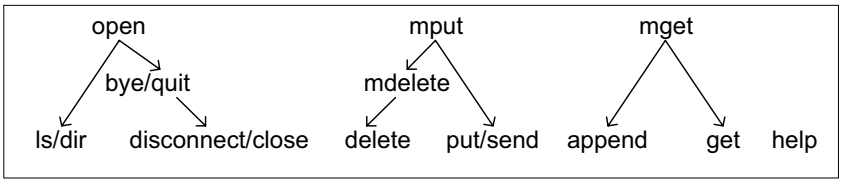

Figure 1 a possible graph for FTP

Here, we are not suggesting an alteration to the behaviour of the system or for that matter the verb set within the system. The rule-programmer in any system must be aware of these interdependencies and accordingly code the rules. Our suggestion lies in recognition of this ordering in the verb set and ways of automating the process of expansion of the rules. This requires the rule engine to be able to derive efficiently which verbs are implied by the inter-relationships between the verbs.

4 Conflict

There are three types of explicit conflict [7, 8, 15, 16]:

1. A subject is authorised and not authorised to apply the same verb to the same object.
2. A subject is obliged and not obliged to apply the same verb to the same object.
3. A subject is obliged but not authorised to apply the same verb to the same object.

4.1 Exposing hidden conflict

Semantic ordering relies on the ordering structure between the verbs. This structure is something that any rule-programmer should be aware of in the normal course of events. When rules are input into the system they can create some hidden conflict, consider the following example

```
positive authorisation : {Danny} {write()} {hamlet}        (1)

negative authorisation : {Danny} {read()} {hamlet}         (2)
```

Assuming $(read \preceq write)$ we can detect a hidden conflict[1]. In a more complex verb set it would be even more difficult to detect such a conflict. An automated system, such as the one we are suggesting, using a ontology will generate the following rule from the rule (1) :

```
positive authorisation : {Danny} {read()} {hamlet}
```

The conflict between this rule and the rule (2) above would be easy to detect at compile time.

4.2 Implied-rules: their effect on authorisation and obligation

Semantic ordering has different effects on authorisation and obligation policies. While the automatic generation of implied rules is appropriate for authorisation policies, however, in the case of obligation, consider the following example:

```
positive obligation: {Danny} {write()} {hamlet}
```

Now assume that to write implies to copy i.e.($copy \preceq write$), it would not necessarily be the intent that the following rule is also generated

```
positive obligation: {Danny} {copy()} {hamlet}
```

However, if a manager is obliged to write a file, the system can automatically generate all the necessary authorisations that come with writing the file, which means authority to write and all the implied verbs which write would generate.

In the next section we will consider a more formal approach to what has been described above.

[1] There maybe some situations that the permission to write a file does not imply the right to read that file. In such a situation the system has to provide a separate write function to cover the requirement or else not have this ordering. How each verb behaves is a meta-policy decision, an anomaly such the one described above is not created by our approach, in any system these anomalies need to be dealt with separately.

5 A meta-policy: semantic ordering of verb set

Consider a set of verbs where there is some form of semantic [12] ordering between the verbs in the set e.g. the following set of verbs in the context of file management {write, read, copy, send, print}, one could impose an ordering on the elements of this set [9, 10], see figure 2.

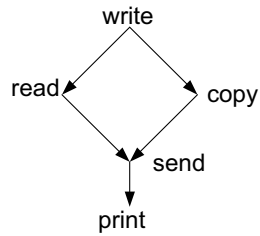

Figure 2 a graph representing the set {write, read, copy, send, print}

Using the ordering relationship one can see, for example, that if $\alpha \preceq \beta \preceq \gamma$ and the following positive policy is added to the policy set (s, γ, o) then automatically the following two positive policies namely (s, β, o) and (s, α, o) should also be added to the system. On the other hand, if the negative policy (s, α, o) is added to the system in the first place then it implies the following two negative policies (s, γ, o) and (s, β, o). This notion of an ordering relationship can be extended to cover policy sets, which are Cartesian products of the subject, verb and object sets. This means, if there is a policy set $\langle S, V, O \rangle$ we should add $\langle S, \Phi(G, V), V \rangle$ where function $\Phi(G, V)$ is the function, which will return the set of all verbs implied by V. Later we will discuss the implementation of function $\Phi(G, V)$ where G is graph representing the poset. The verb set forms a partially ordered set (poset), thus:

$\forall \alpha, \beta, \gamma, \in VerbSet$

- $\alpha \preceq \alpha$ Reflexive
- $(\alpha \preceq \beta) \wedge (\beta \preceq \gamma) \Rightarrow (\alpha \preceq \gamma)$ Transitive
- $(\alpha \preceq \beta) \wedge (\beta \preceq \alpha) \Rightarrow (\alpha = \beta)$ anti-symmetry

In other words reflexivity means each verb implies itself. Transitivity can be expressed as if verb α implies verb β and β in turn implies γ then we can infer α implies γ. Anti-symmetry can be understood as saying if verb α implies verb β and β in turn implies verb α itself then verbs α and β are the same (in the semantic ordering sense). By rendering poset structure on the verb set, we can now transfer the flat set of verbs into a directed acyclic graph. This not only will assist the systems management task as discussed above but also will provide an easy to understand pictorial representation of the verb set within the system.

6 Algorithm for finding implied verbs

Here we will give an algorithm for finding the implied verbs.

1. print the start node
2. for each child (if it is not been visited yet) consider it as a starting point and go to (1)

The recursive pseudo code of the algorithm for finding the implied verbs (findImpliedVerb)

```
solutionSet = ∅   //empty set

findImpliedVerb(currentVerb) {

   solutionSet = solutionSet ∪ {currentVerb}

   for each α ∈ set of adjacent nodes of currentVerb do

      findImpliedVerb(α)
}
```

This algorithm is a top down, depth first and it traverses the graph by moving down through neighbours (i.e. collecting all the implied verbs).

6.1 Using mirror image for implied negative policies

If we draw a graph of a partially ordered set as has been described above and draw a horizontal line under it which acts as a mirror, the graph G with respect to this mirror line is G_m. Note the directions of the edges have all been reversed. Formally we can define the G_m as following:

Let $G = (V, E)$ be a graph where V is the set of vertices and E (a binary relation on V i.e. $E \subseteq V \times V$) the set of edges. The **mirror** image of G is defined as:

$G_m = (V, E')$ such that for $(v, u) \in E' \Rightarrow (u, v) \in E$. Notice if a graph G is superimposed with its mirror image G_m the result would be an undirected graph.

Now we can use the same algorithm on G_m whenever we need to handle policies in negative mode, since G_m represents the correct hierarchy of implied verbs for negative policies.

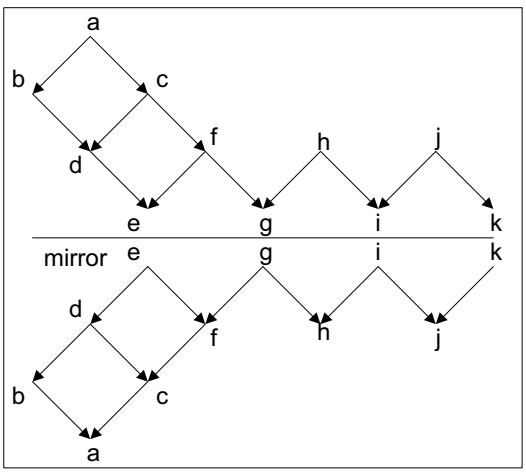

Figure 3 a graph with its mirror image

The following function heading specifies the findImpliedVerb formally:

$\phi :: (graph \times vertex) \rightarrow impliedSet$

We can extend the function to accept a set of vertices as the second element of the input tuple and define the following recursive function using pattern matching:

$$\Phi :: \left(graph \times vertices\right) \rightarrow impliedSet$$
$$\Phi\left(G, \{\}\right) = \{\}$$
$$\Phi\left(G, \{v\} \cup S\right) = \phi\left(G, v\right) \cup \Phi\left(G, S\right)$$

Now in the next section we can talk about the minimum verb set needed to express a set of rules including the implied rules.

7 Minimum verb set

This section examines how to identify the minimum number of verbs required, to express all the rules relating a subject (or a set of subjects) to an object (or a set of objects). First, let us consider the case of a subject s that can do every verb on a object o. To do so we need to define the notion of minimal element and its dual maximal element in a poset.

Let P be a poset and B be a subset of P. An element $b \in B$ is a ***minimal element*** of B if $b \in B$ and no element $b' \in B$ exists such that $b \neq b'$ and $b' \preceq b$. Similarly, Let P be a poset and B be a subset of P. An element $b \in B$ is a ***maximal element*** of B if $b \preceq x \in B \Rightarrow b = x$. For the graph of Figure 3 the minimal elements are $\{e, g, i, k\}$ and maximal elements are $\{a, h, j\}$. In terms of the graph G of a poset and its mirror image G_m, the maximal elements of graph G are the minimal elements of graph G_m and vice versa. The function $\lambda\left(G\right)$ to return the set containing the maximal elements in the graph G is defined. Considering the function findImpliedVerb, it can be observed that $\Phi\left(G, \lambda\left(G\right)\right) = V\left(G\right)$ where $V\left(G\right)$ is the set of vertices in the graph G. In other words only the maximal elements in the graph are needed be able to generate all the verbs. Therefore $\lambda\left(G\right)$ represents the minimum set that can generate the whole graph.

It can be seen that $1 \leq \left|\lambda\left(G\right)\right| \leq \left|V\left(G\right)\right|$, when $\left|\lambda\left(G\right)\right| = 1$, all the verbs within the system form a graph with one maximal element (this would include a linear chain) and when $\left|\lambda\left(G\right)\right| = \left|V\left(G\right)\right|$, there are no semantic relations between the verbs. Therefore, unless there are absolutely no semantic dependencies within the verb set in the system then $\left|\lambda\left(G\right)\right| < \left|V\left(G\right)\right|$. However, in most cases there would be some interdependency between the verbs. After all, any system, intuitively, offers a package that in its totality solves some problem. For example, consider, file management, banking, stock market, hospital management and it is inconceivable that there is a real system where verbs are not related in a semantic sense and this in turn means the maximum number of applicable rules would be equal to $\lambda\left(G\right)$ and where $\left|\lambda\left(G\right)\right| < \left|V\left(G\right)\right|$.

Now, bearing in mind that the set of all verbs within the system would be very unlikely to form a connected graph, the most likely the graph would be a collection of disconnected components. Let us present the graph G of n disconnected components as $G = \left\langle G_1, G_2, \cdots, G_n \right\rangle$ where each component G_i is a connected graph. The number of maximal elements in the graph G, is equal to the sum of the number of maximal elements in each component i.e. $\left|\lambda\left(G\right)\right| = \sum_{i=1}^{n} \left|\lambda\left(G_i\right)\right|$. Further refinement can be achieved by considering a more realistic case, where subject s can do all verbs in set X on object o (where $X \subseteq G_f$)[2]. X can be partitioned into mutually exclusive sets

[2] $V(G) = G_f$ where G_f is the flat set that has vertices of G as its elements.

$X_1\ X_2\ \cdots\ X_m$ such that $\bigcup_{i=1}^{m} X_i = X$ and where for each X_i there is a unique G_j such that $X_i \subseteq G_j$.

Let $S \subseteq \{1, \cdots, n\}$ be a set such that $\forall\ s \in S\ (\exists\ i \in \{1, \cdots, m\}\ \textit{such that}\ X_i \subseteq G_s)$ also notice that for each i there is a unique s which additionally means $|S| = m \leq n$. This would define a bijection between the elements of set $\{1, \cdots, m\}$ and the elements of set S. This means, the elements of set S can be enumerated by members of the set $\{1, \cdots, m\}$ i.e. the set S can be presented as $\{s_1, s_2, \cdots, s_m\}$. Now remembering that every subset of a poset is also a poset then each X_i can be represented by a graph. Indeed as $X_i \subseteq G_j$ for some $j \in S$ then G_j^i (using the bijection, G_j^i can be represented as $G_{s_i}^i$) represents the graph form of the set X_i and it is a sub-graph of G_j. Now whereas each G_j is a connected graph the sub-graph $G_{s_i}^i$ might not be a connected graph and can be represented as a vector of connected components $G_{s_i}^i = \langle G_{s_i 1}^i, G_{s_i 2}^i, \cdots, G_{s_i p}^i \rangle$ where each $G_{s_i k}^i$ is a connected graph. The set X can be represented by the irregular shaped two-dimensional matrix (notice, the number of components for each $G_{s_i}^i$ is different and the term matrix is used in a loose sense, hence, the adjective irregular). Let G_X be the graph representation of the set X. Also, let the sequence $p_1, p_2, \cdots, p_m$ represent the number of components in each $G_{s_i}^i$ then:

$$G_X = \left\langle \begin{matrix} G_{s_1 1}^1\ G_{s_1 2}^1\ \cdots\ G_{s_1 p_1}^1 \\ G_{s_2 1}^2\ G_{s_2 2}^2\ \cdots\ G_{s_2 p_2}^2 \\ \vdots \\ G_{s_m 1}^m\ G_{s_m 2}^m\ \cdots\ G_{s_m p_m}^m \end{matrix} \right\rangle$$

The minimum set that can generate the same set of verbs as X can do is:

$$\lambda(G_X) = \lambda \left\langle \begin{matrix} G_{s_1 1}^1\ G_{s_1 2}^1\ \cdots\ G_{s_1 p_1}^1 \\ G_{s_2 1}^2\ G_{s_2 2}^2\ \cdots\ G_{s_2 p_2}^2 \\ \vdots \\ G_{s_m 1}^m\ G_{s_m 2}^m\ \cdots\ G_{s_m p_m}^m \end{matrix} \right\rangle$$

$$\lambda(G_X) = \bigcup_{i=1}^{m} \bigcup_{q=1}^{p_m} \lambda\left(G_{s_i p_q}^i\right)$$

While the size of the set of maximal elements in graph G_X can be calculated as;

$$\left|\lambda(G_X)\right| = \left| \bigcup_{i=1}^{m} \bigcup_{q=1}^{p_m} \lambda\left(G_{s_i p_q}^i\right) \right|$$

Which means:

$$= \sum_{i=1}^{m} \sum_{q=1}^{p_m} \left|\lambda\left(G_{s_i p_q}^i\right)\right|$$

The above quantity represents the size of the minimum set of verbs that is required to express the set of rules relating subject s to object o, that would give the same result as using the set X of verbs.

8 Ways of implementing SO in a PBMS

At this point we will present a couple of ways of implementing the poset in rule based management systems. These implementations, broadly speaking, are divided into two different categories of static and dynamic approaches.

- A static approach: Rules are input in a rule-based language. The relationships between the verbs are also input, using OWL to represent the ontology. The relationships between the verbs obtained from the ontology can be used to expand the rules. The full set of rules is examined for any conflict (see figure 4).

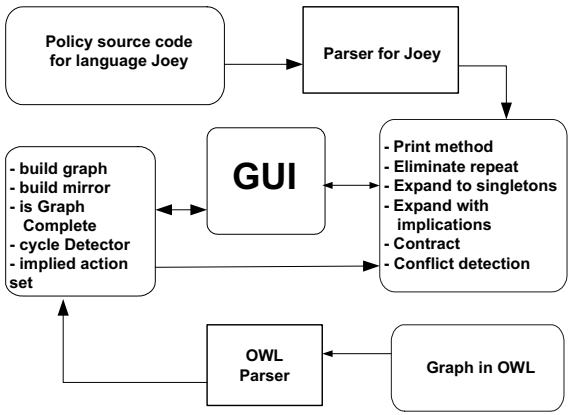

Figure 4 a static approach

- A dynamic approach: Rules are input in a rule-based language. The relationships between the verbs are also input, using OWL to represent the ontology. However, no expansion is done. Instead, whenever a subject requests to invoke a verb, on an object, the system will try to find out if the verb is permitted, by consulting the rules (to see if the rule was explicitly there). If no match is found, it will compare verbs of the explicit rules having the same subject and object with the ancestor verbs in the ontology. If there is a match, then permission is granted, otherwise permission is denied.

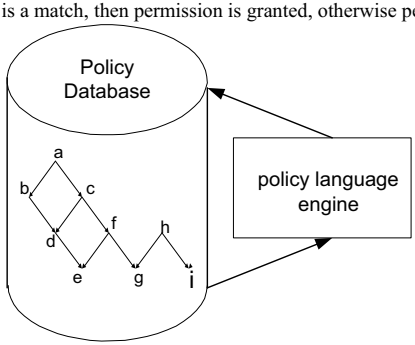

Figure 5 a static approach

However, the static approach, is more suitable for conflict analysis.

9 Conclusion

In this paper we have described how to use ontology to augment the rule system in a policy-based management system. The use of ontology (implemented in RDF or OWL) will not only help to expand rules to their full implication according to their semantics, but also expose the hidden semantic conflict which can be detected at compile-time. Additionally, the rules can be contracted using the minimum set.

Here in BT we have implemented the static approach as described above. However, for future and further work it would be interesting to apply the techniques in a more complex system to examine further the issues associated with scaling up the system.

10 References

[1] Majidian A., April 1999, A Model for Static Conflict Analysis of Management Policies, BTTJ Vol17 No. 2, April 1999

[2] Moore, B, et al Policy Core Information Model VI, IETF Internet draft, Available from http://www.ietf.org, May 2000.

[3] Davey, B. A. and Priestly, H.A. Introduction to Lattice and Order Cambridge University Press 1994

[4] Kolari, P et al Enhancing Web Privacy Protection through Declarative Policies, Proceedings of the IEEE Workshop on Policy for Distributed Systems and Networks(POLICY 2005), June 2005.

[5] Cornwell, M, et al Policies for Autonomy in Open Distributed Systems, Conference on the Human Impact and Application of Autonomic Computing Systems (CHIACS2)

[6] Uszok, A, et al Kaos policy and domain services: Toward a description-logic approach to policy representation, decon_iction, and enforcement. In *IEEE 4th* International Workshop on Policies for Distributed Systems and Networks(Policy 2003), 2003

[7] Lupu E and Sloman M: 'Conflict analysis for management policies', Proceeding of Vth International Symposium on Integrated Network Management IM'97 (formally known as ISINM), San-Diego, Chapman & Hall (May 1997).

[8] Bradshaw, J, et al Toward Semantically-Rich Policy Representation: Issues and Applications, Conference on the Human Impact and Application of Autonomic Computing Systems

[9] Hammond, B.; et al. 2002. Semantic enhancement engine: A modular document enhancement platform for semantic applications over heterogeneous content. In Real World Semantic Web Applications. IOS Press. 29–49.

[10] Aleman-Meza, B.; et al 2003. Context-aware semantic association ranking. In *First International Workshop on Semantic Web and Databases.*

[11] Damianou, N, et al The Ponder Policy Specification Language Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks,Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 18-39

[12] Berners-Lee, T, et al The semantic web. Scientic American, 279(5):34.43, May 2001.

[13] Strassner, J Management, MTP, ISBN 1578702259, 2000

[14] Verma, D Policy-Based Networking Architecture and Algorithms, ISBN 1578702267, 2001

[15] Kosiur, D Understanding Policy-Based Networking, ISBN 0471388041. 2001

[16] Mournes, G Detecting and Resolution of Modality Conflict in Policy-Based Mangement System M.Phil London Feb 2002.