



The policy continuum—Policy authoring and conflict analysis

Steven Davy^{a,*}, Brendan Jennings^a, John Strassner^b

^aWaterford Institute of Technology, Telecommunications Software and Systems Group, Cork Road, Waterford, Ireland

^bAutonomic Computing, Motorola Labs, 1301 East Algonquin Road, Mail Stop IL02-2240, Schaumburg, Illinois 60010, USA

ARTICLE INFO

Article history:

Available online 30 April 2008

Keywords:

Policy-based management
Policy continuum
Conflict analysis

ABSTRACT

The policy continuum is a fundamental component of any policy-based management implementation for autonomic networking, but as of yet has no formal operational semantics. We propose a policy continuum model and accompanying policy authoring process that demonstrates the key properties that set a continuum apart from a non-hierarchical policy model. As part of the policy authoring process we present a policy conflict analysis algorithm that leverages the information model, making it applicable to arbitrary applications and continuum levels. The approach for policy conflict analysis entails analysing a candidate policy (either newly created or modified) on a pair-wise basis with already deployed policies and potential conflicts between the policies are fed back to the policy author. Central to the approach is a two-phase algorithm which firstly determines the relationships between the pair of policies and secondly applies an application specific conflict pattern to determine if the policies should be flagged as potentially conflicting. In this paper we present the formal policy continuum and two-phase conflict analysis algorithm as part of the policy authoring process, we describe an implementation where we demonstrate the detection of potential conflicts within a policy continuum.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Policy-based network management systems are widely seen as an appropriate management paradigm to facilitate high-level, human-specified cognitive decision making in network management. The special requirements of autonomic network management such as self-configuration, self-healing, and in particular self-organization and self-management demand highly flexible and powerful management processes, for which policy-based management is well suited. The goal is to allow expensive human attention focus more on defining business logic and less on low-level device configuration processes. Reaching this goal requires the development of effective and extensible algorithms and processes for policy translation/code generation, policy analysis and policy enforcement. Part of the solution is the policy continuum for representing policy, as it can abstract the complexity associated to low level system oriented policies from business level policy authors. This enables business level concerns to be addressed at an abstraction relevant to their knowledge. There are currently two challenges facing the realization of the policy continuum:

- (1) There is currently no formal representation of the policy continuum making it difficult for standard tools to be developed for policy authors and,

- (2) policy analysis of the policy continuum during the policy authoring process is a scarcely researched domain, where there are new complexities associated to detecting policy conflict across multiple continuum level boundaries.

This paper proposes a formal model of the policy continuum to be used with an application specific information model coupled with a continuum sensitive policy conflict analysis algorithm that is part of a policy authoring process. The policy continuum delegates application specific and level specific concepts to an information model, therefore enabling it to have consistent semantics at any continuum level, and making it applicable to arbitrary application domains. We assume the presence of a system information model (typically specified using the UML) which models the structure and relationships between the system managed entities and the policies that effect their management. Given the presence of such an information model, and making minimal assumptions regarding the nature of the information model (for example, that policies are specified as event-condition-action tuples) it is possible to define a generic policy continuum model and conflict analysis algorithm.

A policy conflict is defined to occur when a set of policies are simultaneously satisfied; that is, an event has triggered the evaluation of a set of policies whose conditions have subsequently been satisfied, but the combined actions of the policies will result in the system reaching different final states depending on the order of execution of these actions. Policy conflicts can easily occur,

* Corresponding author. Tel.: +353 860505821.

E-mail address: sdavy@tssg.org (S. Davy).

especially in circumstances when there are multiple authors defining policies for a given system, such is the case when dealing with a policy continuum. Policy authoring in a policy continuum environment is an especially difficult task as the creation or modification of policy at one level can have an affect on policies defined at other levels. Therefore we outline a policy authoring process tailored for the policy continuum, and we define a policy conflict analysis algorithm to function within this process.

The policy conflict analysis algorithm is used to ascertain if a newly created or modified policy (the candidate policy) potentially conflicts with already deployed policies at the same or other levels. This algorithm is designed to be sensitive to that fact the policies may reside at different levels, where they cannot be directly compared. Candidate policies are analysed on a pair-wise basis with currently deployed policies at the same level, with the analysis being split into two phases. In the first phase the algorithm uses the information model to identify relationships between the candidate policy and the deployed policy (for example, the policies may have the same target entity). In the second phase these relationships are examined in the context of an application specific conflict pattern extracted from the information model. The separation of the second phase reflects the fact that different application domains utilize differing policy execution models and hence there is no universal criteria for how policies conflict. For example, the execution model for firewall policies differs considerably from that of access control policies.

The paper is structured as follows. Section 2 outlines our definition and model of the policy continuum. Section 3 specifies the two-phase conflict analysis algorithm, discussing how it integrates with a system information model and the policy continuum model. Section 4 discusses a prototypical implementation of a policy-based management system based on the DEN-ng information model. Section 5 describes a policy management scenario, illustrating how the two-phase algorithm is used to detect potential policy conflicts within a policy continuum. Section 6 discusses related work on policy conflict analysis, highlighting the novelty of our approach. Finally, Section 7 summarizes the paper and outlines topics for future work.

2. Policy continuum

The policy continuum as originally documented by Strassner in [3], is the concept of separating the policies related to different constituencies of policy authors into different level. Each level of the policy continuum is directly related to different aspects of the management of a communications network. The number of levels of the policy continuum may be arbitrary and application specific as depicted in Fig. 2, however Strassner recommends five, namely business, system, network, device and instance. The power of the policy continuum is associated to its ability to integrate the efforts of multiple policy authors where each has a specific expertise and part to play in the definition of policy for network management. The use of the policy continuum for autonomic network management has been discussed in [21], as part of the FOCAL autonomic network management architecture. This section discusses the formal model of the policy continuum, and associated policy authoring process and in specific policy conflict analysis. The importance of a policy authoring process is emphasized through by examining the issues related to the basic continuum modification operations.

2.1. Formal model

This section describes the formal representation we propose for the policy continuum. First we define our assumptions about the

form the information model takes. We then outline some assumptions about the policy model in use; specifically that application specific information is separated from the policy implementation by using an information model. The design of the policy continuum is then presented, where we outline some necessary operations required to interface with the continuum model. Once we have defined the policy continuum model, we then describe a policy authoring process that takes into account the intricate relationships among policies at different levels. Creating, modifying or removing policy at a specific level of the policy continuum is a tedious process that must consider the affect that these operations may have on other continuum levels. Firstly, we describe the notation used throughout the rest of this paper.

2.1.1. Notation

The notation used throughout the paper to describe formalisms is based on the Vienna development methodology (VDM) formal specification language. VDM is an implementation independent model for describing software systems. It is based on the mathematical theory of sets and maps.

Sets are used in VDM to describe different types of entities that can exist in a system and can be thought of as objects in the object oriented world. Maps are used to relate various sets together, where the maps may have specific properties (i.e. transitive, reflexive, symmetric). The notation used throughout is outlined in Table 1.

We use sets to describe the different types of entities that exist, for example, we may define a set to contain all entities of type policy, that way we have a very well defined methods of searching and transforming the elements of this set using the formal language specified. Maps are used to describe relationship between the defined entities, for example we may define a map that relates policies to events, that way we can determine those policies that are associated to specific events.

2.1.2. Information model assumptions

Our approach is based on the use of application specific information gathered from a system information model. The advantages of taking application specific information directly from an information model are threefold:

- (1) New information can easily be added to the information model should the application domain need modification;
- (2) policy can be reused for different application domains by using a different information model; and,
- (3) there is a common format for representing application specific information as opposed to developing a custom repository of information.

We make the following assumptions regarding the form and structure of the system information model (noting that they are satisfied by well known information models including DEN-ng and CIM):

- The information model can represent the basic concepts of object-oriented modeling. Specifically, it should model inheritance hierarchies, associations, operations and attributes. Additionally the information model should be able to represent invariants, as well as pre- and post-conditions relating to the constraints on values of attributes before and after operation invocations.
- The information model encodes knowledge regarding the structure of and relationships between system managed entities.
- Policies are themselves modeled in the information model, in a manner such that their relationships with each other and with managed entities can be ascertained.

Table 1
Notations

Notation	Description
$a \in S$	a is an entity belonging to the set S
$S \setminus a$	The set S less the element a
$S \cup a$	The set S union the element a
$S \subset T$	The set S is a subset of the set T
$\mathbb{P}S$	The power set of S , i.e. a set of sets
$b \in \mathbb{P}S$	b is a set of entities from the set S
$Q = S \rightarrow T$	Q is a map that maps entities from the set S to entities in the set T , i.e. S is the domain of the map, T is the range of the map
$dom \circ Q$	The domain of map Q
$rng \circ Q$	The range of map Q
$q(e)$	Map index function, if $e \in dom \circ q$, then return what e is mapped to
$Q \sqcup [s \rightarrow t]$	Like union, but instead add a new mapping to a map
$Q^\dagger[s \rightarrow t]$	Replaces an existing mapping to a map
$(R \times S \times T)$	A tuple consisting of an entity from each set specified
π^i	Tuple index function, returns the element of a given tuple at index i
$\bar{\pi}^i$	Tuple remove function, returns the tuple with the associated index removed
$(I \rightarrow f)$	A map transform function, the function f is applied to the range of the map
$(I \rightarrow \triangleleft(b))$	A map transform function, the function b is a boolean function and is applied to the range of the map, if a true is returned then the element is kept, otherwise it is discarded
$(I \rightarrow \triangleright(b))$	A map transform function, the function b is a boolean function and is applied to the range of the map, if a false is returned then the element is kept, otherwise it is discarded
$funct:(A) \rightarrow (B)$	A function specification, the parameter is from the set A and the result is from the set B
\cong	The function specification symbol
Q^{-1}	The inverse map of Q
$\forall a \in S:f(a)$	For each entity a in set S , apply function f

- Policies are modeled as event-condition-action tuples, with the semantics of “on event(s), if condition(s), do action(s)”. Also, the subjects and targets of the policies should be specified.
- Operations defined for entities within the information model correspond to actions of policies.

As well as the assumptions relating to the information model itself, we assume that external applications have available to it a range of functions for querying the information model. Fig. 1 formally specifies a number of map functions for the purpose of information model querying (however we do not provide an exhaustive list here). For example, the **ObjectClass** map function returns the class type of any input managed object; given this function, we can explore the information contained within the information model concerning that object’s class. The **ClassDetails** map function maps a class identifier to a tuple of sets of operations, attributes, associations, invariants and a parent class. Similarly, when passed a specific operation identifier, the **OperationConstraints** map returns a tuple containing a set of pre-conditions and post-condition.

$$\begin{aligned}
 objCl &\in ObjectClass = Object \rightarrow Class \\
 cld &\in ClassDetails = Class \rightarrow (\mathbb{P}Operation \times \mathbb{P}Attribute \times \mathbb{P}Association \times \mathbb{P}Invariant \times Class) \\
 opc &\in OperationConstraints = Operation \rightarrow (\mathbb{P}PreCondition \times \mathbb{P}PostCondition) \\
 inv &\in Invariants \subset Constraint \\
 prec &\in PreCondition \subset Constraint \\
 postc &\in PostCondition \subset Constraint \\
 &\vdots
 \end{aligned}$$
Fig. 1. Information model interface.

2.1.3. Policy model assumptions

We assume that policy takes a generic form that is used in many popular policy-based management implementations and standards. A policy is made up of a set of events, conditions and actions that may reference a subject and target. We can represent this in a more formal notation as depicted in (1):

$$\begin{aligned}
 pm \in \mathbf{PolicyMap} &= Policy \\
 &\rightarrow (\mathbb{P}Event \times \mathbb{P}Condition \times \mathbb{P}Action \times \mathbb{P}Subject \times \mathbb{P}Target) \quad (1)
 \end{aligned}$$

The structure of policy is the same at each level of the policy continuum, where the difference among the policies are the types of entities they reference. Policies at the business level are constrained to only reference business level entities represented in the information model, whereas system level policies reference only system level entities. However there may be some overlap where some entities are relevant at the business level and the system level. Criteria that determines business level entities and system level entities is information model dependent, but an example would be that customer, billing and product entities are at the business level whereas router, firewall and access control server are at the system level.

The execution semantics of the above specified policy model may vary depending on the level it is being specified at, for example, business level policies depend on policy decision and policy enforcement points, whereas low-level system/network level policies depend on the embedded services of a network device to enforce policies. The typical enforcement model of policy is; an event is received by the subject’s policy agent, which refers to the policy continuum to determine of any policies for this subject specifies the received event as a trigger. If there is a policy triggered, this will cause a request to perform the specified actions to be sent to the target of the policy. If a policy exists that prohibits the subject from performing the actions, the actions are rejected and the system is notified. Once the subject sends the request to the target, the target’s access controller processes it. If the target’s policies permit the action to be executed, it is performed in the target. If the action is prohibited by the target’s access controller, the action is rejected and the system is notified.

2.1.4. Policy continuum description

The policy continuum [3] dictates that policies at one level of abstraction may be realized as one or more policies at a lower level of abstraction, i.e. business policies can be related to multiple system level policies. This creates dependencies among the sets of policies expressed at different levels, where modifications at one level directly affect the dependencies with the policies specified at other levels. This dependency may or may not exist depending on the policies specified, as policies may exist solely at a single level. The policy continuum is built upon the concept that there may be multiple levels of policies. The dependencies among the policies at the various levels are closely aligned to the dependencies of modeled entities as defined in the information model of

the managed system. An abstract example of a policy continuum is depicted in Fig. 2, where there are multiple policy authors contributing to policy at various levels of the policy continuum.

2.1.5. Requirements

This section focuses on the requirements of a model of a policy continuum. These requirements will be used to derive a formal specification of a policy continuum along with an associated set of base operations used to interface with the policy continuum.

2.1.5.1. Create/retrieve/update/delete policies. Initially the policies need to be created by a policy author. From this interface policies must also be retrieved/searched, updated and deleted at the various levels, respectively. This gives rise to the following requirements from the perspective of a policy author:

- Add a policy at a specific level.
- Remove a policy from a specific level.
- Update a policy at a specific level.
- Retrieve a set of policies given search criteria.

2.1.5.2. Analysing policies. Once a policy has been created it must be analysed for correctness, we assume there is a separate process for syntactic analysis. The newly created or modified policy should also be analysed for conflicts among policies specified at the same level of the policy continuum. The next section will go into more detail regarding policy conflict analysis. Provided

the policy does not conflict with policies at the same level, it must then be refined where a set of operation are performed on the lower levels to reflect the addition of the new policy. These operations may create, modify or remove polices at lower levels. A refinement process is not discussed in this paper, however the interfaces provided must be capable of supporting the operations required by such a process.

The requirements defined for a policy continuum require it to be very flexible in relating policies together, and that it be able to derive complex relationships among policies. These complex relationships among policies demand thorough and effective analysis processing before, during and after any modification to the policy continuum. These requirements are met by the formally specified policy continuum model.

Analysis of policies is not limited to conflict detection and refinement, there are other related analysis processes, such as policy optimization, policy combination, policy similarity, that are not considered in this paper but none the less should be integrated in the analysis processes defined for the policy continuum.

2.1.6. Design

The policy continuum is modeled as a tree of policy objects where the definition of a policy object is specified in the previous section. The specification detailed in Eq. (2) describes that a policy p is an element of the set policy:

$$\begin{aligned}
 p &\in Policy \\
 pr \in PolicyContinuum &= Policy \rightarrow (\mathbb{N} \times \mathbb{P}Policy)
 \end{aligned}
 \tag{2}$$

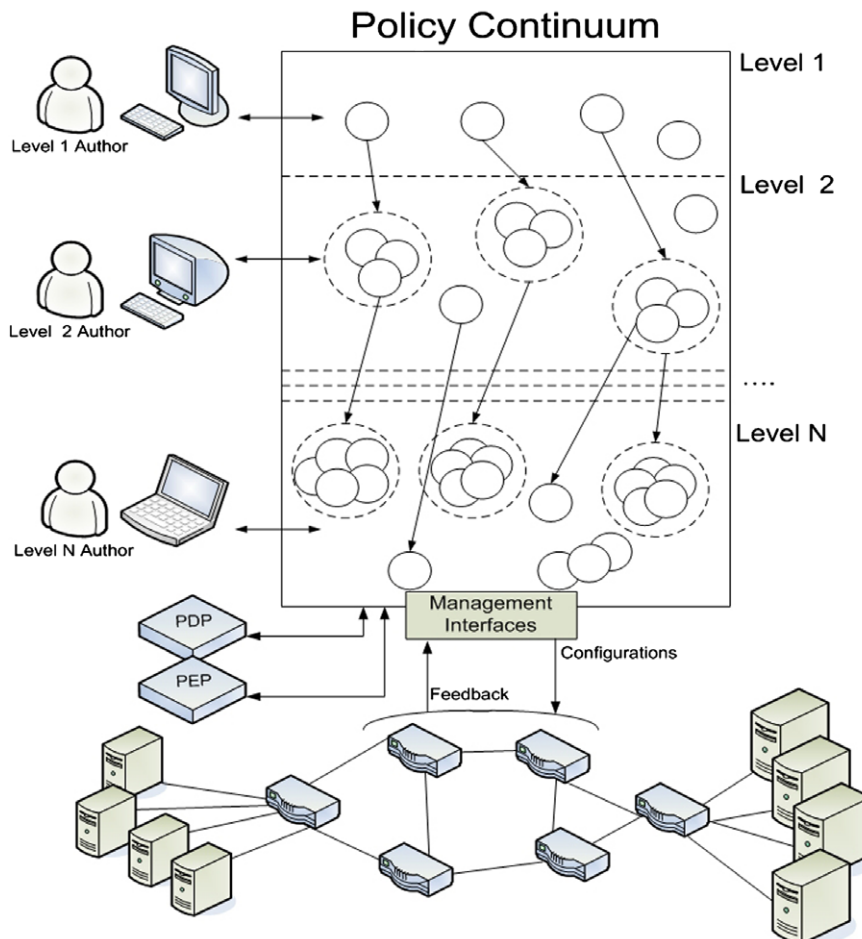


Fig. 2. Policy continuum.

The *PolicyContinuum*, pr , is a map function and the root container for all policies. The *PolicyContinuum* maps policies to a level represented by a natural number within the continuum and to a nested policy set that represents the set of policies at a lower level. The natural number representing the policy continuum level is a monotonically increasing value starting at 1. This facilitates the ability for a policy to reference policies in other levels. A policy can only be associated directly with policies defined at a lower level; also a policy at a specific level may be referenced by more than one policy at a higher level. All policies within the policy continuum must be directly accessible from the *PolicyContinuum* map.

The model for the policy continuum as presented above is defined so that policies created by different constituencies of users can be associated to each other. A model of the policy continuum has been typically implied in the literature to date, whereas now we can begin to deal with the issues that relate to associating policies in this way. The main issues are related to the analysis of policies for conflict, however this analysis can only be performed within the constraints of policy continuum modification operations.

2.1.7. Basic continuum operations

Once the structure of the policy continuum has been defined, some basic operations can be formally described to enable processes to make use of the model to store and retrieve policies.

2.1.7.1. AddPolicy. The first requirement for the policy continuum is the ability to add a given policy to a specific level of the policy continuum. This function therefore simply adds a policy at the given level and associates it to a null set of policies, demonstrating that the policy is not yet realized at any other level of the policy continuum. In the operation defined in (3), n refers to a continuum level, p is a policy identifier and pr is the top level policy continuum to which the policy mapping is being added. The result of the operation is a modified policy continuum:

$$\begin{aligned} \text{AddPolicy} &: (\mathbb{N} \times \text{Policy}) \rightarrow (\text{PolicyContinuum} \rightarrow \text{PolicyContinuum}) \\ \text{AddPolicy}(n, p)pr &\triangleq pr \sqcup [p \rightarrow (n, \emptyset)] \end{aligned} \quad (3)$$

This operation however does not take into account the association of policies to lower level policies and is therefore incomplete. A refinement process is required to modify the lower levels of the policy continuum so that this new policy can be realized and enforced correctly. This begins to motivate the need for a powerful authoring process that takes care of the issues related to modifying the policy continuum.

2.1.7.2. RemovePolicy. A policy should be removed using this function (4) only if it does not reference lower level policies, i.e. there are no dependencies among policies. Therefore this operation will only function in specific circumstances; the removal of a policy that is associated to lower level policies is a delicate operation and must be supported by a policy authoring process:

$$\begin{aligned} \text{RemovePolicy} &: \text{Policy} \rightarrow (\text{PolicyContinuum} \rightarrow \text{PolicyContinuum}) \\ \text{RemovePolicy}(p)pr &\triangleq pr \setminus p \end{aligned} \quad (4)$$

2.1.7.3. UpdatePolicy. Updating policies (5) involves changing the associated lower level set of policies of a given policy. Also if a policy is updated then the goals of the associated higher level policies may be jeopardized, this effectively means that while updating a policy, we must be aware that such an operation can have an affect on both higher level policies and lower level policies. Therefore care must be taken when using this function, and it should preferably be used within the confines of a policy authoring process.

Essentially, the policy p and a set of policies ps are inserted into the continuum, where the policy set ps overrides the existing policies associated to p . The policy is maintained at the same level of the continuum by copying the current continuum level number using the map index operator π^1 :

$$\begin{aligned} \text{UpdatePolicy} &: \text{Policy} \times \mathbb{P}\text{Policy} \rightarrow (\text{PolicyContinuum} \rightarrow \text{PolicyContinuum}) \\ \text{UpdatePolicy}(p, ps)pr &\triangleq pr \dagger [p \rightarrow (\pi^1 \circ pr(p), ps)] \end{aligned} \quad (5)$$

2.1.7.4. GetPolicyChildren. We can retrieve the child policies of a specific policy (6) all the way down the policy continuum. This is useful to see the impact a specific policy may have on the policy continuum. The function is recursive and is called on each of the child policies associated to a policy p . The operation terminates when there are no child policies left to call. The operation returns an enumerated set of child policies. Effectively we are traversing down the policy continuum:

$$\begin{aligned} \text{GetPolicyChildren} &: \text{Policy} \rightarrow (\text{PolicyContinuum} \rightarrow \mathbb{P}\text{Policy}) \\ \text{GetPolicyChildren}(\emptyset) &\triangleq \emptyset \\ \text{GetPolicyChildren}(p)pr &\triangleq \\ &\quad \forall p_n \in ((I \rightarrow \pi^2)pr)(p) : \text{GetPolicyChildren}(p_n) \\ &\quad \bigcup ((I \rightarrow \pi^2)pr)(p) \end{aligned} \quad (6)$$

2.1.7.5. GetPoliciesAtLevelN. We can retrieve all policies at a specific level of the policy continuum (7). This is useful when analysing policies at a specific level. The policy continuum pr which contains all policies is restricted to only those policies that are at the level equal to the level specified as an argument to the operation. The domain of this reduced map is a set of policies at a specific level:

$$\begin{aligned} \text{GetPoliciesAtLevelN} &: \mathbb{N} \rightarrow (\text{PolicyContinuum} \rightarrow \mathbb{P}\text{policy}) \\ \text{GetPoliciesAtLevelN}(n)pr &\triangleq \text{dom}(I \rightarrow \triangleleft[\pi^1 = n])pr \end{aligned} \quad (7)$$

2.1.7.6. GetAllPolicies. We can get all individual policy objects (8). Since all policies must at least be expressed in the domain of the policy continuum map:

$$\begin{aligned} \text{GetAllPolicies} &: \text{PolicyContinuum} \rightarrow \mathbb{P}\text{Policy} \\ \text{GetAllPolicies}(pr) &\triangleq \text{dom}(pr) \end{aligned} \quad (8)$$

2.1.7.7. GetPolicyParents. We can get all policies that reference a specific policy (9), i.e. those policies defined at a higher level within the policy continuum. This is useful when you need to trace up the policy continuum given a policy at a specific level. The operation firstly reduces the policy continuum pr to only those policies at a higher level to the given policy p . This map is then transformed to a policy to policy set map. The inverse of this map is a map between policies and their associated parent policies. When this new inverse map is indexed by p , the parent policies of p are returned:

$$\begin{aligned} \text{GetPolicyParents} &: \text{Policy} \rightarrow (\text{PolicyContinuum} \rightarrow \mathbb{P}\text{Policy}) \\ \text{GetPolicyParents}(p)pr &\triangleq \\ &\quad ((I \rightarrow \pi^2)\text{GetPoliciesAtView}(\pi^1 \circ pr(p) - 1))^{-1}(p) \end{aligned} \quad (9)$$

2.1.7.8. GetCommonPolicies. We can discover those lower level policies that are in common to two given policies (10). This operation makes use of the *GetPolicyChildren* operation, where the child policies of the supplied policies p_a and p_b are retrieved, the intersection of which are classed as common policies:

$$\begin{aligned}
& \text{GetCommonPolicies} : (\text{Policy} \times \text{Policy}) \rightarrow (\text{PolicyContinuum} \rightarrow \mathbb{P}\text{Policy}) \\
& \text{GetCommonPolicies}(p_a, p_b)pr \triangleq \\
& \text{GetPolicyChildren}(p_a) \cap \text{GetPolicyChildren}(p_b)
\end{aligned} \tag{10}$$

2.1.8. Summary of policy continuum

We have seen that the policy continuum is a complex concept, and we have outlined an interface to it. The interface provided enables the addition, removal and update of policies to the policy continuum. Search operations are also provided to enable us to further examine policy relationships. However the interfaces provided should be used as part of an intelligent policy authoring process so that the correct policies are added and removed as the continuum is modified. The next section outlines a continuum sensitive policy authoring process that makes use of the policy continuum model and interface.

2.2. Policy authoring process

The process a policy author must go through to create or modify a policy that is merged into the policy continuum is consistent across all levels of the policy continuum. Fig. 3 depicts the process of modifying an existing policy, from now on referred to as the candidate policy. The authoring process is split into three steps, the first step traces up the policy continuum to verify that the modification of the candidate policy does not invalidate current policy continuum semantics. The second step analyses the policies at the same level for potential conflict with the candidate policy. The third step invokes a refinement process to derive a set of lower level policies from the candidate policy that must be recursively verified and tested for conflict and validity. Fig. 4 depicts a flow diagram explaining the process from the perspective of a policy author at a particular level.

We begin by making sure the modification to p_{old} , represented by p_{cnd} (the candidate policy) still satisfies all roles that the policy

```

ModifyPolicy : (Policy × Policy × PolicyContinuum) → PolicyContinuum
ModifyPolicy( $p_{old}, p_{new}, pc$ )  $\triangleq$ 
 $\forall p_{par} \in \text{GetPolicyParents}(p_{old})pc :$ 
  if not VerifyPolicyContinuum( $p_{par}$ )pc
  then
    return pc
  if DetectPolicyConflict( $p_{new}$ )pc
  then
     $\forall p_{cnd} \in \text{PotentialConflictList}(p_{new})pc :$ 
     $\forall p_{par} \in \text{GetPolicyParents}(p_{cnd})pc :$ 
      NotifyCurrentAuthor( $p_{par}$ )
    return pc
  else
     $\forall p_{ref} \in \text{RefinePolicy}(p_{new})pc :$ 
    if  $p_{ref} \in \text{NewPolicy}$ 
    then CreatePolicy( $p_{ref}, pc$ )
    elseif  $p_{ref} \in \text{ModifiedPolicy}$ 
    then ModifyPolicy( $p_{ref}, pc$ )
CommitChange( $p_{old}, p_{new}, pc$ )

```

Fig. 3. Policy authoring process.

plays in regards to its association to higher level policies. Therefore we retrieve a set of parent policies by calling *GetParentPolicies* on p_{old} and for each parent policy we *verify* that it is still consistent when using the modified version of the policy, i.e. p_{cnd} . If each policy is satisfied then we continue, otherwise the candidate policy p_{cnd} is causing inconsistency within the policy continuum and should not be committed. Assuming the candidate policy has passed the previous test, it must be analysed for conflict against currently deployed policies at the same continuum level. If a conflict is detected, we need to retrieve the set of associated parent policies that are indirectly involved in the conflict so that more information about the source of the conflict can be relayed back to the current candidate policy author. If no conflict is detected at the current level, p_{cnd} is refined into a set of lower level modifications that may consist of create or modify instructions to lower level policies. For each policy instruction the outlined process is repeated. If refinement is not needed then the process returns with a successful commit. The process continues until all refinement operations are successful thus enabling us to commit the candidate policy p_{cnd} . In the next section we focus specifically on the process of policy conflict analysis which is an integral part of the policy authoring process. Therefore we assume the existence of policy validity operations and policy refinement operations. Related works in these areas are covered in Section 5.

3. Conflict analysis algorithm

We have seen that the policy authoring process makes extensive use of two algorithms that work independently of each other but are used together to deliver an effective solution. Specifically the algorithms are *DetectPolicyConflict* and *RefinePolicy*. The rest of this section focuses explicitly on *DetectPolicyConflict*. We therefore assume an existing algorithm for policy refinement of which there exist many that will be reviewed in the section focused on related work. We will outline how the conflict detection algorithm integrates into the policy authoring process of the policy continuum, and we specify our policy conflict analysis algorithm.

3.1. Continuum integration

In order to incorporate the algorithm to work in a continuum we need to describe how it works when there are multiple policy authors at multiple levels. From the description of the policy continuum we have seen that we can trace up the policy continuum given a specific policy using *GetPolicyParents*. Also we can trace down the policy continuum using *GetPolicyChildren*. We can make use of these operations to traverse the policy continuum when we need to. As a consequence there are multiple entry and exit points associated to the algorithm. Policy modification instructions may be initiated at a particular continuum level, or may be propagated down from higher continuum levels. Also feedback from successful conflict detection may be fed back to the policy author or propagated up the policy continuum to a higher level policy author.

3.2. Specification

Fig. 5 provides an overview of our policy conflict analysis algorithm, depicting how it integrates with the policy authoring process. When a policy is newly created, or an already deployed is policy modified, this “candidate” policy is analyzed for potential conflict on a pair-wise basis with all the other deployed policies. For a given candidate and deployed policy pair, if any potential conflicts are detected the user receives appropriate feedback via

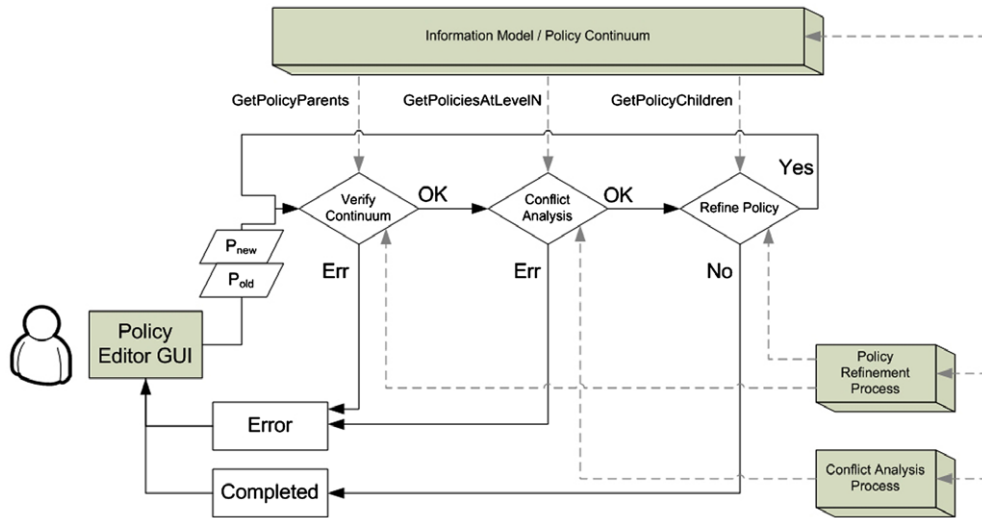


Fig. 4. Policy authoring flow chart.

the policy authoring GUI; we assume that the user then makes the decision on whether to ignore or attempt to resolve the potential conflict. The algorithm makes extensive use of the information model to access application specific information. This makes the algorithm adaptable to the different applications. The two phases of the algorithm separate the relationship analysis between policies, from the identification of those specific relationships among policies that may lead to a potential conflict. The determination of relationships that lead to potential conflict is also application specific, and is therefore separated from the conflict detection algorithm. We now provide a detailed description of the two phases.

3.2.1. Algorithm phase 1: policy relationship analysis

The algorithm initially creates a matrix that encodes the various types of relationships between the pair of policies; this matrix represents a policy relationship “pattern.” As described later, different relationship patterns constitute different forms of potential policy conflict, depending on the particular semantics of policy enforcement in the application domain in question. Construction of the relationship matrix is done by the **PolicyRelationship** function, which examines the two policies in order to discover whether specific relationships exist between them. As depicted in Fig. 6 the initially zeroed matrix is modified by a set of operations that populate the matrix with ones depending on their result.

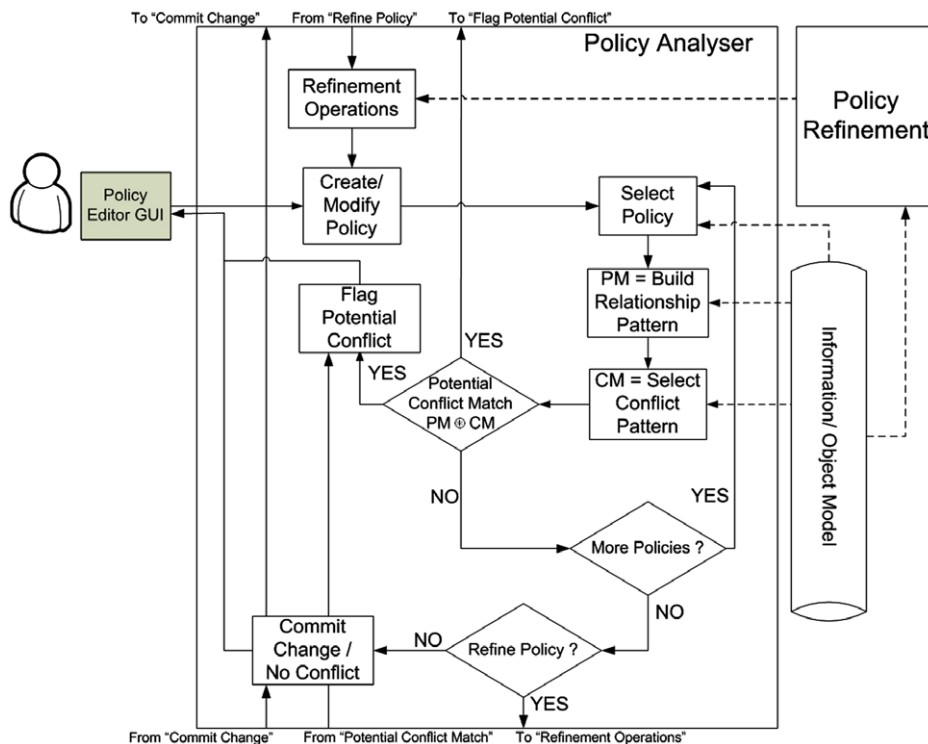


Fig. 5. Policy conflict analysis flow chart.

nents. The **associatedByAction** operation checks if there are relationships between actions defined in different policies. The **isActionContradiction** operation examines if any of the actions specified in the first policy contradicts with any of the actions specified in the second policy.

A policy conflict may not exist even if the actions of two policies contradict. To ascertain if they do contradict the operation **isActionContradiction** (specified in Fig. 6) is used, which consults the information model to ascertain if

- (1) Two operations are acting on the same type of object,
- (2) their pre-conditions hold true, and
- (3) their post-conditions are ‘incompatible’.

Only if all three conditions are true are the actions considered contradictory. The pre-condition specifies the constraints on the attributes of the object before the operation can be performed and the post-condition specifies the constraints on the attributes of the object after the operation has completed. By ‘incompatible post-conditions’ we mean that the object cannot satisfy all constraints expressed by one or more post conditions at the same time. Invariants that are defined must also be upheld after the operations have completed.

3.2.2. Algorithm phase 2: conflict pattern matching

The manner in which policies are analyzed for potential conflict is highly dependant on the application for which the policies are being defined. For example, when determining if two access control policies conflict, there must be an overlap among the subjects, targets and actions. Therefore, we can associate policies by subject, target and action as a first step, and those policies that can be seen to overlap after this step can be flagged to signify potential conflicts. On the other hand, for filtering policies (such as firewall rules) we may not be interested in subject, target, action overlap, but instead in event, condition, action overlap, in particular only when the target entities are identical. Clearly the pattern of relationships between policies gives a clear indication of the potential for conflict; however, different relationships are relevant in different application contexts.

In phase 2 of our algorithm a matrix containing the application domain specific conflict pattern (extracted from the information model) is used to make the decision whether to flag the policies as potentially conflicting. We use an operation that matches the relationship matrix produced in phase 1 with a generic conflict matrix that represents the set of relationships that may or must hold for conflict. The matrix combination operation combines the values of two matrices by using AND and OR operations and is specified in Fig. 7. The conflict matrix specifies the pre-conditions for conflict. Thus, the specification of conflict is decoupled from the detection algorithm and can accommodate disparate conflict definitions.

Fig. 8 depicts a sample conflict pattern matrix that is examined in this phase. Each row represents the relationships of a component of policy. To describe a condition for conflict, a one is placed in the associated position in the conflict pattern matrix. For example, if a specific type of conflict requires subject subset membership (*ssb*), then a 1 would be placed in the upper left position of the matrix. The codes used in Fig. 8 represent the associated relationships considered so far.

The codes used in Fig. 8 represent the associated relationships considered so far. The first letter of the codes indicates which component is being analysed, s for subject, t for target, and similarly for event, condition and action. The tail of the codes indicate the type of relationship being established, sb for subset, sp for super set, eq for equal, cor for correlation, mux for mutually exclusive and ctd for contradict. It is important to note that the list of relationship types given in the matrix below can be expanded upon. In this pa-

$$(M \otimes M)_{i,j} \rightarrow B$$

$$(a \otimes b)_{i,j} \triangleq \bigwedge_{p=0}^i \bigvee_{q=0}^j (a_{p,q} \wedge b_{p,q})$$

Fig. 7. Matrix comparison operator.

<i>ssb</i>	<i>ssp</i>	<i>seq</i>	<i>scor</i>	<i>0</i>
<i>tsb</i>	<i>tsp</i>	<i>teq</i>	<i>tcor</i>	<i>0</i>
<i>esb</i>	<i>esp</i>	<i>eeq</i>	<i>ecor</i>	<i>emux</i>
<i>csb</i>	<i>csp</i>	<i>ceq</i>	<i>ccor</i>	<i>cmux</i>
<i>asb</i>	<i>asp</i>	<i>aeq</i>	<i>acor</i>	<i>acld</i>

Fig. 8. Phase 2 conflict matrix.

per we consider a set of popular relationship types among policies, but as there is certainly more ways to relate two policies we have decided to leave the algorithm extensible. Both the conflict pattern and the policy relationship pattern can be increased in the number of columns and rows so that new ways of relating policies can be developed.

It is important to note that the list of relationship types given in the matrix can be expanded upon. In this paper we consider a set of popular relationship types among policies, but as there are certainly more ways to relate two policies we have decided to leave the algorithm extensible. For example we have not describe the relationship represented by *emux*, which associates policies if their events are mutually exclusive meaning that the events associated to two policies cannot occur simultaneously. More importantly, both the conflict pattern and the policy relationship pattern can be increased in the number of columns and rows so that new ways of relating policies can be developed in the future.

A possible extension to the matrix is to support relationships that between policies that are related to the types of policies. For example, the Ponder policy language specifies five distinct policy types, they are namely, positive and negative authorization, obligation, refrain and delegation. A new row in can be easily defined to handle these types of relationships, indeed to detect specific forms of policy conflict this row may be required. In this paper we so not mandate such rows as different policy languages have quite different notions of policy. The relationships we present in the matrix are typical of the majority of current policy implementations.

4. Prototype implementation

For our prototype implementation we use the DEN-ng information model, as described in [1]. The instantiation of the information model, policy authoring tools and policy enforcement facilities are based on the test bed described fully in [2]. We now provide a brief overview of DEN-ng and of our test bed implementation.

4.1. DEN-ng policy model

DEN-ng is a comprehensive information model for communications networks, capturing everything from business concepts (e.g., products, service level agreements, and customers) down to low-level device functionality (e.g., packet marking, forwarding, and queuing). It is designed so that it can be readily augmented with vendor specific information and data models; it thereby provides a highly flexible and extensible information modeling solution. Here, we focus solely on the components of a management policy rule as expressed in the simplified model and depicted in Fig. 9.

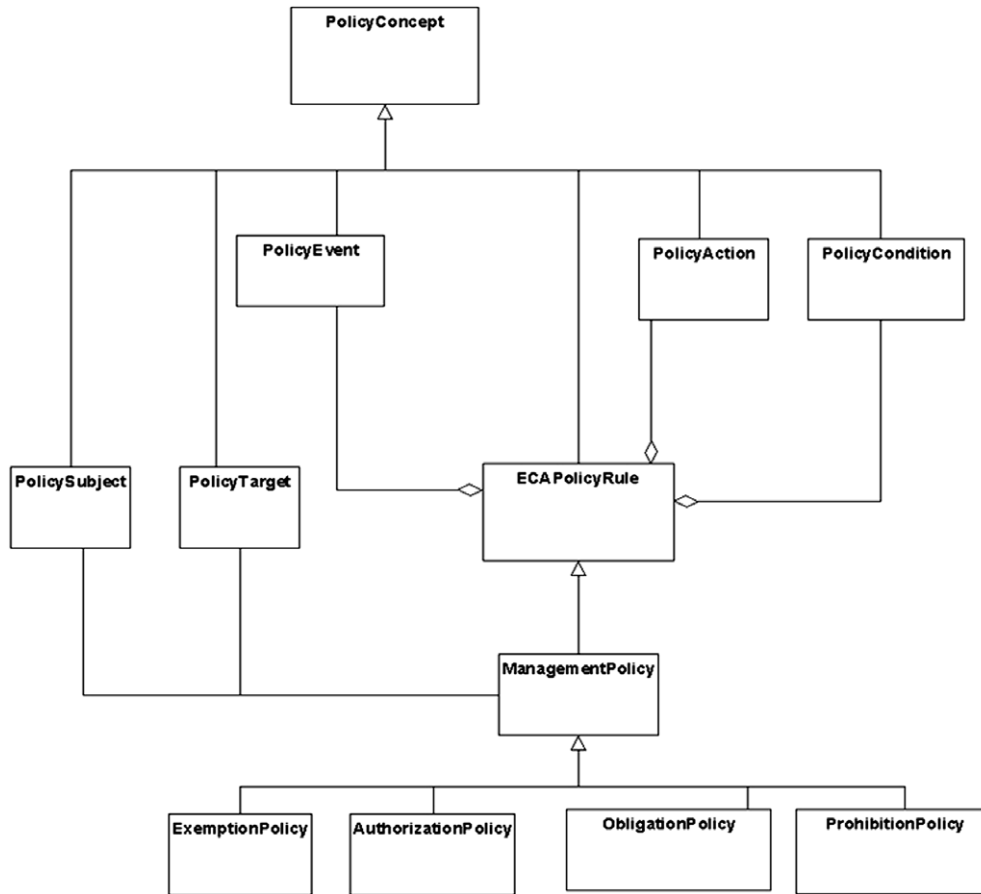


Fig. 9. Simplified DEN-ng policy model.

A DEN-ng management policy is based on the event-condition-action (ECA) rule format, and can be associated to a policy subject and a policy target. The DEN-ng also fully supports the policy continuum by allowing policy authors to represent policies defined for managed entities at different levels of granularity. To demonstrate the abilities of the DEN-ng policy model, we explain how three example policy types can be expressed. Access control policies can be expressed easily using the DEN-ng policy model. The main components required are subject entities and target entities; also, deontic concepts are supported by the model. Network traffic filtering policies can be expressed by associating a policy target to an interface on the networking device; also the conditional component of the policy can support IP traffic match criteria expressions. The explicit ordering required by firewall filtering policies is expressed within the PolicyRuleSpecification class (not shown in the above figures). Additionally, policies can be expressed to manage behaviour at a high level of abstraction, such as for what duration of the day a customer is awarded gold Internet service package. Time intervals can be represented in the conditional component of policy, whereas complex entities such as customer and internet service can be represented in the information model.

4.2. Policy test bed

The test bed is an initial implementation of a policy-based management system for a communications network, described in detail in [2], it includes a policy continuum. The test bed emulates management of a set of Internet services for a set of customers subscribed to an Internet service provider. The net-

work resources being managed are simulated in the OPNET network simulator. The work discussed in this paper is strictly focused on the policy editor component, the policy analysis component, and the information model interface, depicted in Fig. 10. The information model is distilled from DEN-ng and transformed into an ECore representation. ECore is a modelling technology, built on the eclipse platform. The interface to the information model is provided using an API made available through open architecture ware (oAW). They have developed a model query language called xTend that can query any ECore based information model. Using this query language we can develop functions that can retrieve specific class based information such as attribute names, associated classes or class hierarchies.

The conflict analysis algorithm is implemented in Java and makes extensive use of oAW's xTend query language as the interface to the information model. The algorithm is enhanced to store objects in the matrix so that more information about the relationships among policies can be returned to the user should a conflict occur. Policies are authored in the policy editor as depicted in Fig. 11. In the editor there is an analysis button that invokes the processing of the algorithm; if any potential conflicts are detected the information populating the matrix is used to describe the conflict in a dialog box displayed to the user.

5. The conflict analysis scenario

In this section we illustrate the operation of conflict analysis for a policy continuum via a policy scenario implemented in our

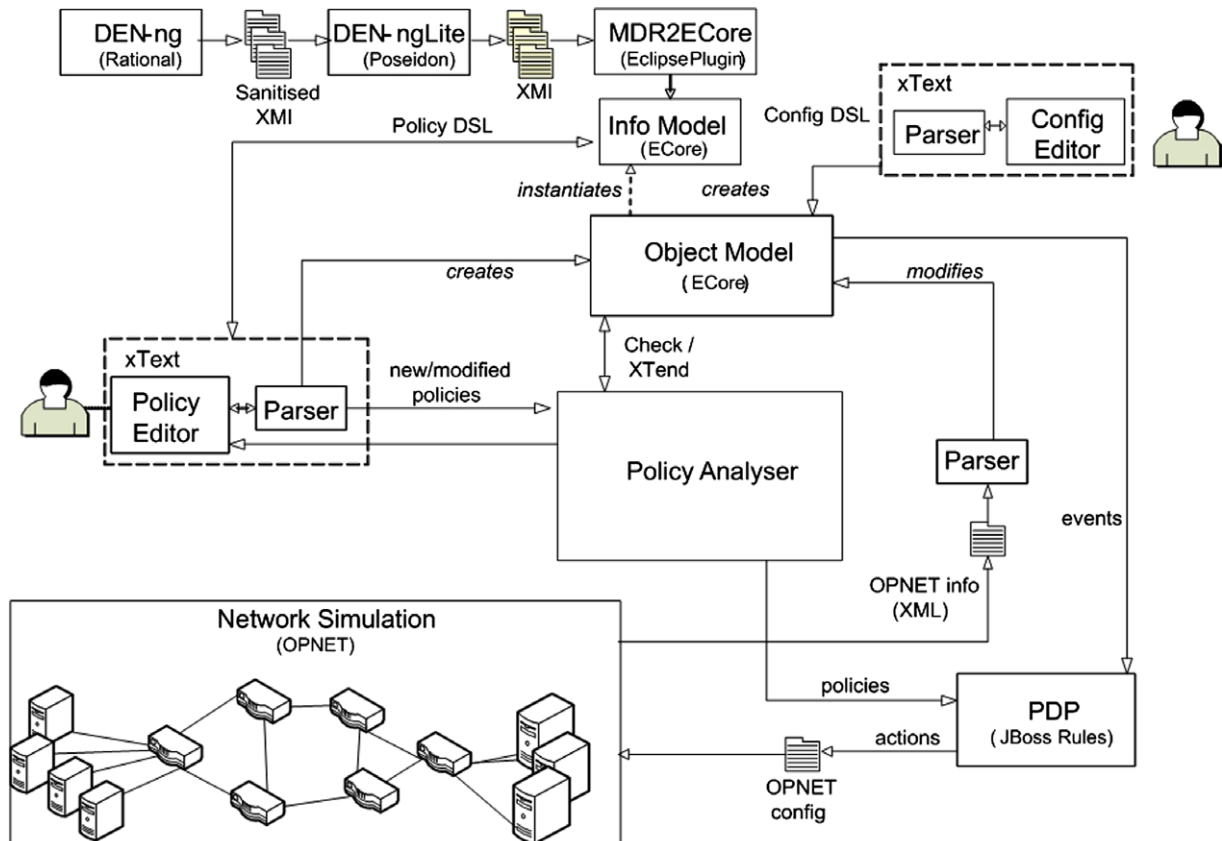


Fig. 10. Test bed architecture.

test bed. The scenario collectively relates to an ISP defining policy to manage the provision of internet service products across its communications network. Policies are defined from the perspective of two management domains that have different concerns about the organization and its services and resources. The ISP's network administration users define policy to effect management of network services and resources. Whereas the ISP's business oriented sales users define policy to effect the provision of products based on internet service grades. The management domains of users and services are as depicted in Fig. 12. There are two cases examined, the first case examines the interaction of firewall filtering policies with service provision policies defined at the business level. This case demonstrates the interaction between policies authored of different levels. The second case examines the interaction of access control policies defined at the business level but where a different type of conflict pattern is used, this demonstrates the versatility of the conflict analysis algorithm.

5.1. Business level/system level – filtering policy conflict

At the business level, a policy author defines a set of policies that relate to the grade of internet service package that a customer is assigned, and to which services it can have access. Table 2 depicts a set of current business policies. For this scenario we define policy in a pseudo policy language for the sake of clarity, for a more in-depth analysis of the policy language used, the reader is referred to [2]. We assume that these policies have already been added to the policy repository and that there were no potential conflicts detected. Subsequently a refinement process has derived a set of related system level policies that can fulfil the network resource access requirements, and packet fil-

tering requirements set out by the business level policies. These are also depicted in Table 2. The network administrator needs to define a policy that will restrict video on demand traffic, so that policies can be installed to perform essential file backup during off-peak hours. The candidate system level policy is defined at the bottom of Table 2 and will be analysed against all appropriate system level policies. The policy describes that video on demand service access must be marked with diffserv code point of AF31, thus reducing its priority and its ability to affect network congestion. The reason for this is so that the system administrator knows that he can reserve emergency bandwidth marked as AF21 for file backup. The conflict patterns that are tested relate to detecting conflict relationships specifically among network filtering policies, where we pay particular attention to the conditions overlap of the IP Header match criteria of the deployed policies.

From examining the policies we see the policy installed by the network administrator may potentially conflict with two deployed policy. The policy relationship pattern between the candidate policy and policy ID3.1 is depicted in Fig. 13. The conflict pattern used to determine a case for conflict is chosen from a set of conflict patterns that represent well known conflict types that can occur among firewall filtering policies. The conflict patterns represent equal policy subject and target, identical events, subset, equal or correlated conditions and contradicting actions. When the first potential conflict is detected, the conflict analysis algorithm traces up the policy continuum to investigate which business level policies this deployed policy is related too. As it turns out the policy with ID 3 at the business level is a parent policy of the problematic deployed policy. Subsequently an alert is presented to the system level policy author describing the potential conflict detected and a list of higher level policies that the deployed policy is related. By

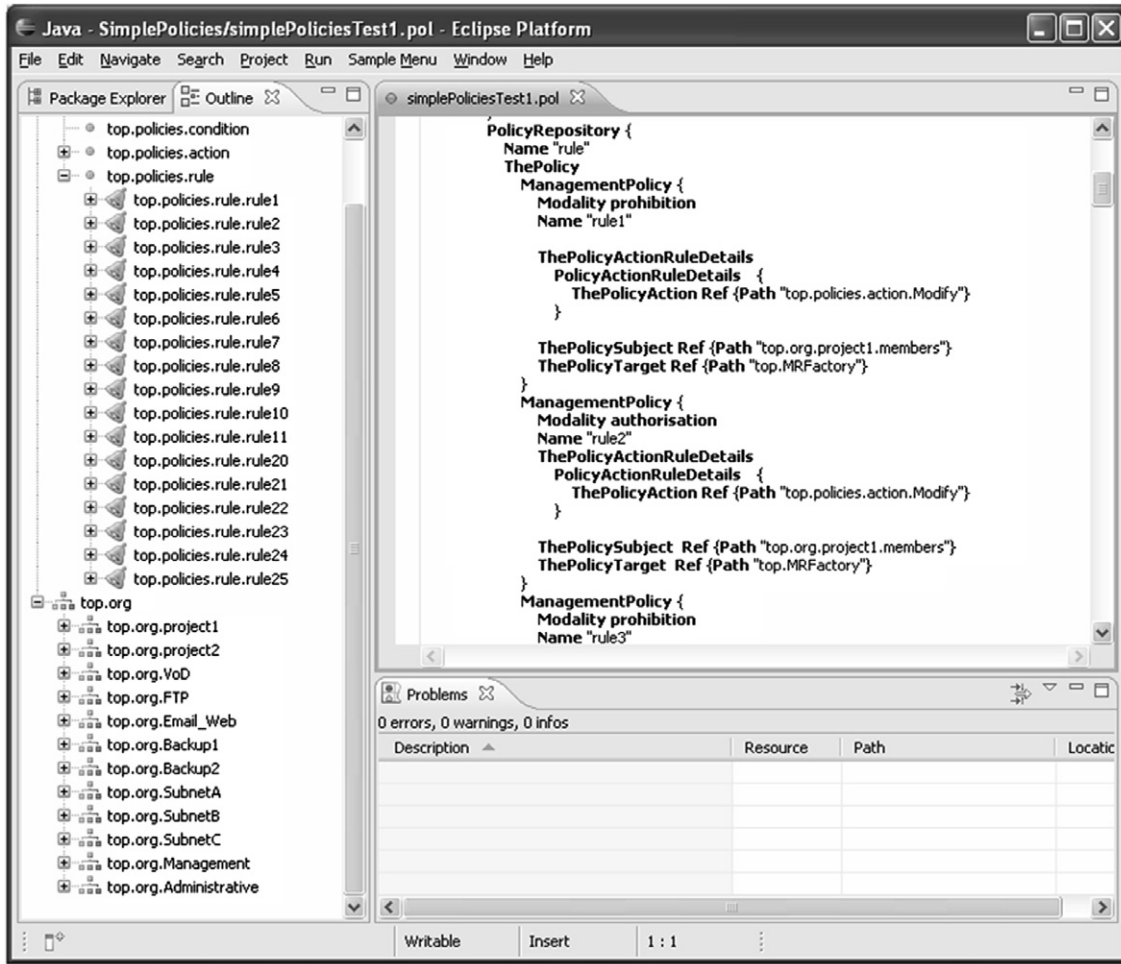


Fig. 11. Policy editor GUI.

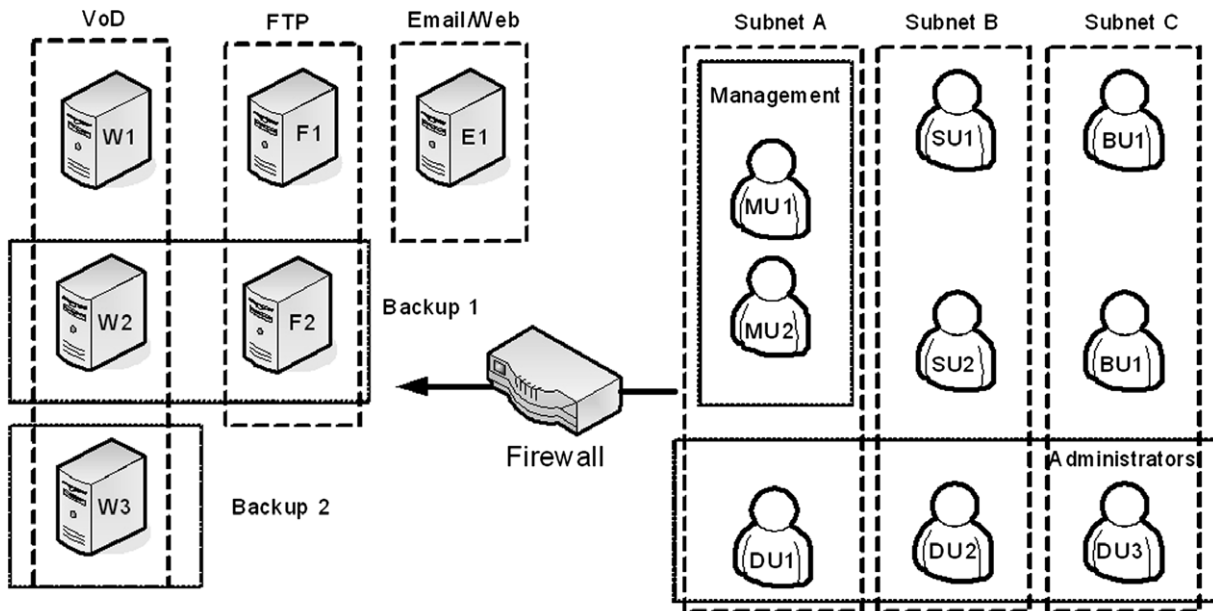


Fig. 12. Conflict analysis scenario.

providing a list of higher level policies the policy author is given more context as to why the deployed policy exists and who to

coordinate with in order to find out more information about the policies.

Table 2
Business level to system level policies

Business view (1)	System view (2)
ID 1 Subnet_A <i>canAccess</i> {VoD, FTP, Web}	ID 1.1 On IPPacketRecieved at AccessRouter. IF0, Condition: SourceIP in 64.10.10.0/24, DestIP in 64.11.1.0/24, DestPort equals 2000, Action: Forward ID 1.3 On IPPacketRecieved at AccessRouter IF0, Condition: SourceIP 64.10.10.0/24, DestIPin 64.11.2.0/24, DestPort equals 21, Action: Forward
ID 2 Subnet_B <i>canAccess</i> {FTP, Web}	ID 2.1 On IPPacketRecieved at AccessRouter. IF0, Condition: SourceIP in64.10.11.0/24, DestIP in 64.11.2.0/24, DestPort equals 21, Action: Forward ID 2.2 On IPPacketRecieved at AccessRouter. IF0, Condition: SourceIP in 64.11.3.0/24, DestIP in 64.10.11.0/24, DestPort equals 80, Action: Forward
ID 3 Subnet_B <i>canNotAccess</i> {VoD}	ID 3.1 On IPPacketRecieved at AccessRouter IF0, Condition: SourceIP in 64.10.11.0/24, DestIP in 64.11.1.0/24, DestPort equals 2000, Action: Drop
ID 4 Subnet_A <i>isAssignedProduct</i> GoldInternet	ID 4.1 On IPPacketRecieved at AccessRouter. IF0, Condition: SourceIP in 64.10.11.0/24, DestIP in 64.11.1.0/24, DestPort equals 2000, Action: Mark AF21
ID 5 Subnet_B <i>isAssignedProduct</i> SilverInternet	ID 5.1 On IPPacketRecieved at AccessRouter. IF0, Condition: SourceIP in 64.10.11.0/24, DestIP in 64.11.1.0/24, DestPort equals 2000, Action: Mark AF31
ADMIN POLICY	ID 0.1 On IPPacketRecieved at AccessRouter. IF0, Condition: SourceIP in 64.10.0.0/16, DestIP in 64.11.1.0/16, DestPort in 2000, Action: Mark AF31

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \oplus \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{1}$$

Fig. 13. Phase 2 computation for business/system conflict.

5.2. Business level policies – access control conflict

This case details a policy conflict that exist solely at the business level where there is no need to refine the policies to the system level in order to ascertain that a potential conflict exists. Therefore, if we detect that a potential conflict may exist we do not need to refine the candidate policy and the policy author can be notified straight away.

The business level policy author defines policies that control access to a set of services available to a set of customers. The currently deployed policies are as outlined in Table 3, and a new candidate policy is defined at the bottom of Table 3. The candidate policy defines that users within the Administrators managed domain are granted access to all services during the interval 17:00–23:00. The first phase of the algorithm compares this candidate policy with all deployed policies to ascertain the appropriate relationships that exist among the policies. The second phase then examines the derived relationships patterns and discovers that a potential conflict exists among the candidate policy and the deployed policy with ID 3. A conflict exists because there is an overlap between the subjects of the policies (i.e. customer DU2 is shared between Subdomain B and Administrators), there is a target overlap in that both policies reference the VoD service, and there is a correlation among the conditions meaning that they temporally overlap. The existence of these relationships indicates a potential conflict as specified in the conflict matrix depicted in Fig. 14. Once the potential conflict is detected it is relayed back to the candidate

Table 3
Business level access control policies

ID 1 Subnet_A <i>canAccess</i> {VoD, FTP, Web} during interval (09:00–18:00)
ID 2 Subnet_B <i>canAccess</i> {FTP, Web} during interval (09:00–18:00)
ID 3 Subnet_B <i>canNotAccess</i> {VoD} during interval (09:00–18:00)
* ID 4 Administrators <i>canAccess</i> {Vod,FTP,Web} during interval (17:00–23:00)

policy author with information pertaining to the detected relationships between the two analysed policies.

6. Related work

The work presented in this paper builds on previously published research by the authors. Specifically Davy et al. presented a preliminary model of the policy continuum in [25]. The policy continuum model presented in that paper is done so without the association to an associated policy authoring process or policy conflict analysis process, although they do indicate that the modification of the policy continuum is non-trivial. Davy et al. present a policy conflict analysis algorithm in [24], while this algorithm is application domain independent, it use with policies defined in a policy continuum is not discussed. Next we discuss related works divided into two areas associated to the work presented in this paper. The areas are policy continuum work and policy conflict detection work.

6.1. Policy continuum

The policy continuum as described in [3] can only be automated when there exists a policy refinement process. The concept of a policy-based network management hierarchy was introduced by Moffett and Sloman [4], where they identified a need for high-level business policies to be translated or refined into lower level policies that carry out the high level objectives. Policy hierarchies are also investigated in [5], where they specifically identify a need for a refinement process to automate the task of associating and creating effective policy hierarchies. The main difference between the hierarchies of policy defined in [4,5], and the policy continuum as defined in [3] is that they do not consider the policy authoring process occurring for multiple levels of policy. Is this circumstance there are non-trivial issues to be aware of, such as those

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \oplus \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{1}$$

Fig. 14. Phase 2 computation for business level policies.

highlighted in this paper. Specifically, creating or modifying policies at low levels can affect the objectives of higher level policies.

More recently, a policy refinement process has been investigated by Bandara et al. [6] and Rubio-Loyola et al. [7,8]. In [6] they investigate the use of event calculus to derive a set of lower level goals that can be used to satisfy the goals of a higher level policy. In [7] they use model checking and requirements engineering to derive a set of low-level goals from high-level policies. The two approaches achieve very similar objectives, and can be integrated into the policy authoring process outlined in this paper. However [6–8] do not consider the consequences of detecting policy conflicts within the derived low-level policies, whereas our approach does. Therefore it is possible to expand on the functionality of existing approaches to policy refinement by incorporating the process into a holistic policy authoring process that is capable of alerting the policy author when conflict has occurred as a consequence of policy refinement.

Until now, there exists no formal model of the policy continuum that enables different levels of policy to be related to each other that is sensitive to the intricate relationships that exist among policies at multiple levels.

6.2. Policy conflict analysis

Popular policy languages such as Ponder [9], Rei [10], KAOs [11], and XACML [12] which are used primarily for access control, all have concepts of event, condition and action. They also are defined over subject and target managed entities. Our assumptions on the required components of a compatible policy language for the analysis algorithm are therefore satisfied by these policy languages. The assumptions we placed on the information model capabilities are also satisfied by the two most popular information model standards available, CIM and DEN-ng. An XML schema based information model is also compatible with our algorithm as it can meet the minimum requirements outlined in Section 2.

An access control policy conflict as described by Lupu and Soman [13], occurs when a set of simultaneously applicable policies result in multiple decisions being equally applicable. A simple case is when two policies of incompatible modality need to be enforced in the same situation. A pre-condition for access control policy conflict as discussed in [13] and later in [14], is that the subjects, the targets and the actions must overlap. When this precondition is met among two or more policies of incompatible modality, then a policy conflict may occur. The problem is exacerbated when subjects and targets are grouped into multiple overlapping domains, or may take on multiple roles. Research defined in [13,14] ignore the fact that policies can be associated to specific external events and conditions that implicitly prevent the concurrent execution of two policies. This fact will inherently prevent a set of apparently conflicting policies from being applied at the same instance. This was addressed in later research by Bandara et al. [15], who describes a formal model of policies based on event calculus that takes into account simple policy events and conditions. Policy conflicts are described in the logic programming language prolog that compares all policies together and if two policies are deemed to breach a constraint then a conflict is detected. Information about the system is encoded into logical assertions derived from state machines. Our approach makes use of an interface to any compatible information model to hold application specific information, therefore reducing the need for custom logic programming. Also our approach decouples the definition of a conflict and reduces it into a matrix pattern of ones and zeros, making it easier to store and understand.

Situations can occur when the preconditions of conflict as previously outlined are not met, but inconsistent behaviour is still observed. Application specific conflict such as ‘conflicts of duty’ and

‘separation of concerns’ can arise when specifying authorization and obligation policies. Whether two policies instantiate a conflict of duty is application specific and its detection requires domain knowledge. The drawback is that each application specific conflict must be described individually and discovered by matching each policy against each other. The research carried out in [15] encodes application specific conflict as logical assertions. Each pair of policies are compared against the logical assertion, if the assertion succeeds, then a conflict is detected. This work was extended by Charalambides et al. [16] to detect application specific conflict in the network management domain, specifically in the area of quality of service management. However, their work could be applied to a wider range of application domains if the application specific logic were contained in a generic information model. Our work does not need to be customised per application domain as we have designed it to be application independent since it takes input from application specific information models.

A taxonomy of policy conflicts among firewall filtering policies [17,18] and among network security filtering policies [19] show that there is a set of repeating conflict scenarios that can occur. The most commonly discussed forms of policy conflict in network configuration are that of shadowing, correlation, generalization and redundancy. Packet filtering conflicts are discovered not by matching subject/target/action, but by examining the order of the policies and the overlap in the conditions of the policies in a single device interface. Golnabi et al. [20], describe firewall anomalies or conflicts in terms of subset, superset and correlation relationships among the match criteria of IP packets. Although this work produced useful results, the concept can be expanded to cover all components of policy, i.e. subjects, targets, events and actions, as we demonstrated in this paper.

7. Conclusion

We present a formal model of the policy continuum, with accompanying policy authoring process and a two-phase application independent policy conflict analysis algorithm. Operational semantics for basic policy continuum operations are provided that should be used in coordination with an intelligent policy authoring process that is capable of maintaining consistency among policy levels. We focus on an application domain independent conflict analysis algorithm as part of the authoring process that is sensitive to multiple policy continuum levels. We achieved application independence by separating the relationship analysis of policies from the conflict description where both phases make use of an application specific information model. We demonstrated this in a scenario with two cases, examining access control policies and firewall filtering policies. The algorithm can be applied to arbitrary domains so long as the assumption related to the policy language used and the interface to the information model used are satisfied. The algorithm is extensible in that the types of relationships formed among the candidate and deployed policies can be added to. As the conflict pattern is represented as a bit matrix, it can be easily extended to be made aware of new types of policy relationships. As the algorithm makes use of a generic interface to an information model, the model can be changed depending on the domain. In future work we investigate reducing the search space for analysing candidate policies by reusing previous analysis patterns among deployed policies. We intend on incorporating ontologies to augment the information model, so that we can represent relationships that are not otherwise possible. Ontologies will enable us to detect a greater range of conflict among policies. The complexity associated to ascertaining relationships among policies will be further investigated and is a critical issue when analysing policy conflict for more expressive policy languages such as those derived from ontologies.

Acknowledgement

This work has received support from the Science Foundation Ireland under the Autonomic Management of Communications Networks and Services programme (Grant no. 04/IN3/1404C).

References

- [1] J. Strassner, J.N. de Souza, D. Raymer, S. Samudrala, S. Davy, K. Barrett, The design of a new policy model to support ontology driven reasoning for autonomic networking, in: Proceedings of the Fifth Latin American Network Operations and Management Symposium LANOMS, 2007, pp. 114–125.
- [2] K. Barrett, S. Davy, B. Jennings, S. van der Meer, J. Strassner, A model based approach for policy tool generation and policy analysis, in: Proceedings of the IEEE Global Information Infrastructure Symposium, 2007, pp. 99–105.
- [3] J. Strassner, Policy-based network management, Morgan Kaufmann, ISBN 1-55860-859-1, 2003.
- [4] J.D. Moffett, M.S. Sloman, Policy hierarchies for distributed systems management, IEEE Journal on Selected Areas in Communications 9 (1993) 1404–1414.
- [5] R. Wies, Using a classification of management policies for policy specification and policy transformation, in: Proceedings of the Integrated Network Management IV, vol. 4, 1995, pp. 44–56.
- [6] A. Bandara, E. Lupu, J. Moffet, A. Russo, A goal-based approach to policy refinement, in: Proceedings of the Fifth IEEE Workshop on Policies for Distributed Systems and Networks, 2004, pp. 229–239.
- [7] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G. Pavlou, A. Lafuente, Using linear temporal model checking for goal-oriented policy refinement frameworks, in: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005, pp. 181–190.
- [8] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G. Pavlou, A functional solution for goal-oriented policy refinement, in: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks, 2006, pp. 133–144.
- [9] N. Damianou, N. Dulay, E. Lupu, M. Sloman, The ponder policy specification language, in: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, 2001, pp. 18–38.
- [10] L. Kagal, T. Finin, A. Joshi, A policy language for a pervasive computing environment, in: Proceedings of the Fourth International Workshop on Policies for Distributed Systems and Networks, 2003, pp. 63–74.
- [11] A. Uszok, J.M. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M.R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, J. Lot, KAoS policy and domain services: toward a description-logic approach to policy representation, deconfliction, and enforcement, in: Proceedings of the Fourth IEEE International Workshop on Policies for Distributed Systems and Networks, 2003, pp. 93–96.
- [12] S. Godik, T. Moses, et al., eXtensible Access Control Markup Language (XACML) Version 1.0, OASIS Standard, February 2003.
- [13] E. Lupu, M. Sloman, Conflicts in policy-based distributed systems management, IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management, 25 (6) (1999) 852–869.
- [14] N. Dunlop, J. Indulska, K. Raymond, Dynamic conflict detection in policy-based management systems, in: Proceedings of the Sixth International Conference on Enterprise Distributed Object Computing, EDOC'02, 2002, pp. 15–26.
- [15] A.K. Bandara, E.C. Lupu, A. Russo, Using event calculus to formalize policy specification and analysis, in: Proceedings of the Fourth IEEE Workshop on Policies for Distributed Systems and Networks, 2003.
- [16] M. Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, J. Rubio-Loyola, Policy conflict analysis for quality of service management, in: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005, pp. 99–108.
- [17] E. Al-Shaer, H. Hamed, R. Boutaba, M. Hasan, Conflict classification and analysis of distributed firewall policies, IEEE Journal on Selected Areas in Communications 23 (10) (2005) 2069–2084.
- [18] C. Lin, C. Xue, L. Zhitang, Analysis and classification of ipsec security policy conflicts, in: Proceedings of the Frontier of Computer Science and Technology, Japan–China Joint Workshop on FCST '06, 2006, pp. 83–88.
- [19] H. Hamed, E. Al-Shaer, Taxonomy of conflicts in network security policies, Communications Magazine IEEE 44 (3) (2006) 134–141.
- [20] K. Golnabi, R. Min, L. Khan, E. Al-Shaer, Analysis of firewall policy rules using data mining techniques, in: Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium, 2006, pp. 305–315.
- [21] J. Strassner, N. Agoulmine, E. Lehtihet, FOCALE – a novel autonomic networking architecture, in: Proceedings of the Latin American Autonomic Computing Symposium, LAACS, Brazil, 2006.
- [22] D. Agrawal, J. Giles, K. Lee, J. Lobo, Policy ratification, in: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005, pp. 223–232.
- [23] D. Lin, P. Rao, E. Bertino, J. Lobo, An approach to evaluate policy similarity, in: Proceedings of the 12th ACM symposium on Access control models and technologies, SACMAT'07, ACM, 2007, pp. 1–10.
- [24] S. Davy, B. Jennings, J. Strassner, Application domain independent policy conflict analysis using information models, in: Proceedings IEEE/IFIP Network Operations and Management Symposium, NOMS, 2008, pp. 17–24.
- [25] S. Davy, B. Jennings, J. Strassner, The policy continuum – a formal model, in: Proceedings of the Second IEEE International Workshop on Modelling Autonomic Communications Environments, MACE, 2007, pp. 65–79.