Application Domain Independent Policy Conflict Analysis Using Information Models

Steven Davy,[1] Brendan Jennings,[1] John Strassner[2]
[1]Telecommunication Software & Systems Group
Waterford Institute of Technology, Cork Rd., Waterford, Ireland
{sdavy, bjennings}@tssg.org
[2]Motorola Labs
Schaumburg, IL, USA
john.strassner@motorola.com

***Abstract*— A key part of the policy authoring process is analysis of the potential for newly created or modified policies to conflict with already deployed policies. We propose an approach for policy conflict analysis in which candidate policies (either newly created or modified) are analyzed on a pair-wise basis with already deployed policies, with potential conflicts between the policies being notified to the policy author. Central to the approach is a two-phase algorithm which, querying an information model, 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. The algorithm is generic in the sense that all application specific information is encoded in the information model; as long as a minimal set of assumptions regarding the policy model are adhered to it can be applied in arbitrary application domains. In the paper we present the two phase algorithm and describe an implementation in which it is used to detect potential conflicts for both access control and filtering (firewall) policies.**

Keywords-Policy Based Management, Conflict Detection, Information Model.

I. INTRODUCTION

Policy-based network management systems are widely seen as an appropriate paradigm to facilitate high-level, human-specified cognitive decision making in network management. 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, conflict analysis and policy enforcement. In this paper we focus on detection of potential conflicts between policies. A policy conflict occurs when 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, especially in circumstances when there are multiple authors defining policies for a given system.

The goal of our work is to define a generic and extensible policy conflict analysis approach that is independent of the application domain to which the policies relate. We assume the presence of a system information model (specified using the UML) that models the structure and relationships between the managed entities and the policies that effect their management. Given the presence of such an information model, and making certain assumptions regarding its nature (for example, that policies are specified as event-condition-action tuples) we define a generic policy conflict analysis algorithm. In this algorithm, candidate policies are analysed on a pair wise basis with currently deployed policies, with the analysis being split into two phases. In the first phase the information model is used 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 processes and hence there is no universal criteria for how policies conflict. For example, the execution process for firewall policies differs considerably from that of access control policies.

The paper is structured as follows. Section II specifies the two-phase conflict analysis algorithm, discussing how it integrates with a system information model. Section III discusses a prototypical implementation of a policy based management system based on the DEN-ng information model. Section IV describes three separate policy management scenarios, illustrating how the two-phase algorithm is used to detect potential policy conflicts. Section V discusses related work on policy conflict analysis. Finally, Section VI summarizes the paper and outlines topics for future work.

II. CONFLICT ANALYSIS APPROACH

In this section we specify the two-phase conflict analysis algorithm. This algorithm depends on the presence of an information model embodying the knowledge required to analyze policies for potential conflict. Before specifying the algorithm we discuss our assumptions regarding the form of the information model and the available information model query/search interface.

A. Information Model Assumptions

Our approach to policy conflict analysis is based on 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 without necessitating modification of the analysis algorithm(s); 2) analysis algorithms 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. We make the following assumptions regarding the form and structure of the information model (noting that they are satisfied by well known information models such as DEN-ng):

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 state of attributes before and after operation invocations;

The information model encodes knowledge regarding the structure of and relationships between managed entities. The degree to which the model reflects the actual managed system constrains the ability of the conflict analysis algorithm to identify potential policy conflicts;

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;

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 the conflict analysis process has 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 (due to space constraints 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, state 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.

$$objCl \in ObjectClass = Object \rightarrow Class$$
$$cld \in ClassDetails = Class \rightarrow (\mathbb{P}Operation \times \mathbb{P}Attribute \times \mathbb{P}Association \times \mathbb{P}Invariant \times Class)$$
$$opc \in OperationConstraints = Operation \rightarrow (\mathbb{P}PreCondition \times \mathbb{P}PostCondition)$$
$$inv \in Invariants \subset Constraint$$
$$prec \in PreCondition \subset Constraint$$
$$postc \in PostCondition \subset Constraint$$
$$\vdots$$

Figure 1. Information Model Querying Function Specification.

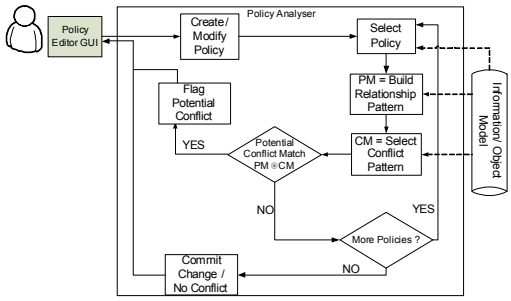

Figure 2. Policy Conflict Analysis Flowchart.

B. Conflict Analysis Algorithm Specification

Fig. 2. provides an overview of our policy conflict analysis process, depicting how it integrates with the policy authoring process. When a policy is newly created, or an already deployed policy modified, this "candidate" policy is analyzed for potential conflict on a pair-wise basis with an appropriate subset of the already deployed policies. The selection process is not the focus of this paper; for details of an appropriate selection process the reader is referred to [1]. For a given candidate and deployed policy pair, if any potential conflicts are identified the user receives appropriate feedback via the policy authoring GUI; we assume that the user then makes the decision on whether to ignore, or to attempt to resolve, the potential conflict. The two-phase conflict analysis algorithm is specified at a high level in Fig. 3; we now provide a detailed description of the two phases.

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 their different component types. If no relationship is found between the two policies then we assume no conflict exists. As depicted in Fig. 4. the initially zeroed matrix is modified by a set of operations that populate the matrix with ones depending on their result.

$$\textbf{\textit{testForConflict}} : (Policy \times Policy) \rightarrow \mathrm{B}$$
$$\textbf{\textit{testForConflict}}(p_1, p_2) \triangleq$$
$$\quad //Phase\ 1$$
$$\quad let\ m = \textbf{\textit{PolicyRelationship}}(p_1, p_2)$$
$$\quad if\ m\ !=\ ZeroMatrix$$
$$\quad\quad then$$
$$\quad\quad //Phase\ 2$$
$$\quad\quad let\ cf = \textbf{\textit{RetrieveConflictPattern}}(p_1, p_2)$$
$$\quad\quad if\ \ m \circledast cf = 1$$
$$\quad\quad\quad then\ 'Flag\ for\ Conflict'$$

Figure 3. Policy Conflict Analysis Algorithm

For example, the ***associateBySubject*** operation takes as input the two policies (denoted p_1 and p_2) and tests for a set of relationships that exist among their subjects. There are four tests to compare each policy via subject, where the tests examine the subject of the policies for subset, superset, equality or correlation (intersection). For example, if the function ***isSubjectSubset*** returns true, 1s are placed in the input matrix at the positions that signify that the subject of p_1 is a subset of the subject of p_2, and inversely that the subject of p_2 is a superset of the subject of p_1. Similarly, the same operation can be performed for testing for subject superset membership, subject equality and subject correlation. After the relationship matrix has been populated with information associated to subjects, the targets of policies can also be related in the same manner.

The computation required to determine whether the subjects of one policy are a subset of the subjects of another policy requires specific information about the application domain. The specific information may be for example, that the subject class in the information model uses a containment relationship to indicate contained entities. The inputs to the function (which are not depicted in the figure for the sake of clarity) are an ***ObjectClass*** map and a ***ClassDetails*** map. We can make use of some well defined interfaces to the information model to simplify the computation of these functions, such as an operation that can determine if one class inherits from another. The ***isTypeOf*** operation compares two classes together that are specified from within the information model and can determine if one is equal to or inherits form the other. The ***GetMemberEntities*** function enables an entity of a particular type to be searched for all of its members by exploring containment relationships. This function is recursive, so that nested relationships are also expanded. Using these two helper functions, we can find a complete set of domain entities that represent the subjects of both input policies; a test for inclusion will determine if one set is a subset of the other.

Events, conditions and actions can be related in a manner similar to that described above for subjects and targets. When the events of one policy are computed to be a superset of the events of another policy, then we are sure that when the first policy is triggered, the second policy is also triggered. Similarly, logical inference among conditions is similar to the subset, superset, and equality relationships. Therefore, when we relate policies via the condition component, we can begin to draw conclusions as to which policies infer the condition components of other policies. Relating action components can reveal information about what actions may be repeated by other policies and therefore may be redundant. There are additional ways of relating these components. 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. 4.) is used, which consults the information model to ascertain if 1) two operations are acting on the same type of object, 2) their pre-

$$pm \in \textbf{PolicyMap} = Policy \rightarrow (\mathbb{P}Event \times \mathbb{P}Condition \times \mathbb{P}Action \times \mathbb{P}Subject \times \mathbb{P}Target)$$
$$\vdots$$
$$\textbf{SetupMatrix} : (Policy \times Policy \times PolicyMap) \rightarrow Matrix$$
$$\textbf{SetupMatrix}(p_1, p_2, pm) \triangleq$$
$$m = \textbf{ZeroMatrix}$$
$$m = \textbf{associateBySubject}(p_1, p_2, pm) \circ$$
$$\textbf{associateByTarget}(p_1, p_2, pm) \circ$$
$$\textbf{associateByEvent}(p_1, p_2, pm) \circ$$
$$\textbf{associateByCondition}(p_1, p_2, pm) \circ$$
$$\textbf{associateByAction}(p_1, p_2, pm)m$$
$$\vdots$$
$$\textbf{associateBySubject} : (Policy \times Policy \times PolicyMap \times Matrix) \rightarrow Matrix$$
$$\textbf{associateBySubject}(p_1, p_2, pm)m \triangleq$$
$$if\ \textbf{isSubjectSubset}(p_1, p_2, pm)$$
$$then$$
$$markSSB(p_1, p_2) \circ markSSP(p_2, p_1)m$$
$$elseif\ \textbf{isSubjectSuperset}(p_1, p_2, pm)$$
$$then$$
$$markSSP(p_1, p_2) \circ markSSB(p_2, p_1)m$$
$$elseif\ \textbf{isSubjectEqual}(p_1, p_2, pm)$$
$$then$$
$$markSEQ(p_1, p_2) \circ markSEQ(p_2, p_1)m$$
$$elseif\ \textbf{isSubjectCorrelated}(p_1, p_2, pm)$$
$$then$$
$$markSCOR(p_1, p_2) \circ markSCOR(p_2, p_1)m$$
$$\vdots$$
$$\textbf{associateByAction} : (Policy \times Policy \times PolicyMap \times Matrix) \rightarrow Matrix$$
$$\textbf{associateByAction}(p_1, p_2, pm, m) \triangleq$$
$$if\ \textbf{isActionSubset}(p_1, p_2, pm)$$
$$then$$
$$markASB(p_1, p_2) \circ markASP(p_2, p_1)m$$
$$elseif\ \textbf{isActionSuperset}(p_1, p_2, pm)$$
$$then$$
$$markASP(p_1, p_2) \circ markASB(p_2, p_1)m$$
$$elseif\ \textbf{isActionEqual}(p_1, p_2, pm)$$
$$then$$
$$markAEQ(p_1, p_2) \circ markAEQ(p_2, p_1)m$$
$$elseif\ \textbf{isActionCorrelated}(p_1, p_2, pm)$$
$$then$$
$$markACOR(p_1, p_2) \circ markACOR(p_2, p_1)m$$
$$if\ \textbf{isActionConflict}(p_1, p_2, pm)$$
$$then$$
$$markACLF(p_1, p_2) \circ markACLF(p_2, p_1)m$$
$$\vdots$$
$$\textbf{isSubjectSubset} : (Policy \times Policy \times PolicyMap \times ObjectClass \times ClassDetails..) \rightarrow \mathrm{B}$$
$$\textbf{isSubjectSubset}(p_1, p_2, pm, objCl, cld, parentType) \triangleq$$
$$let\quad s_1 = \bigcup \forall s_e \in \pi^4 \circ pm(p_1) :$$
$$if\ \textbf{isTypeOf}(objCl(s_e), parentType, cld)$$
$$then\ \textbf{GetMemberEntities}(s_e)$$
$$else\ s_e$$
$$let\quad s_2 = \bigcup \forall s_e \in \pi^4 \circ pm(p_2) :$$
$$if\ \textbf{isTypeOf}(objCl(s_e), parentType, cld)$$
$$then\ \textbf{GetMemberEntities}(s_e)$$
$$else\ s_e$$
$$(s_1 \subset s_2)$$
$$\vdots$$

Figure 4. Conflict Analysis Algorithm Snippets.

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

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

Figure 5. Matrix Comparison Operator.

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 state the object should be in before the operation can be performed and the post-condition specifies the state of the object after the operation has completed. Two post conditions are incompatible is they cannot both be satisfied at the same time. If however pre- and post- conditions are unavailable, we may revert to explicitly marking out contradictory actions which is undesirable as this process is not automated.

2) Algorithm Phase 2: Conflict Pattern Matching

The manner in which policies are analyzed for potential conflict is highly dependant on the application domain of those policies. 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. 5. The conflict matrix specifies the pre-conditions for conflict. Thus, the specification of conflict is decoupled from the detection algorithm and can accommodate different conflict definitions.

Fig. 6. 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 1 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. 6

$$\begin{bmatrix} ssb & ssp & seq & scor & 0 \\ tsb & tsp & teq & tcor & 0 \\ esb & esp & eeq & ecor & emux \\ csb & csp & ceq & ccor & cmux \\ asb & asp & aeq & acor & aclf \end{bmatrix}$$

Figure 6. Phase 2 Computation

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 depicted in Fig. 6 can be expanded upon. In this paper we consider a set of straightforward relationship types among policies; however, 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.

III. PROTOTYPE IMPLEMENTATION

For our prototype implementation we use the DEN-ng information model, as described in [2]. The instantiation of the information model, policy authoring tools and policy enforcement facilities is based on the test bed described fully in [3]. We now provide a brief overview of DEN-ng and of our test bed implementation.

A. DEN-ng Policy Model

DEN-ng is a comprehensive information model for telecommunications, 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 modeling solution. Here, we focus solely on the components of a management policy rule as expressed in the simplified model depicted in Fig. 7.

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. 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

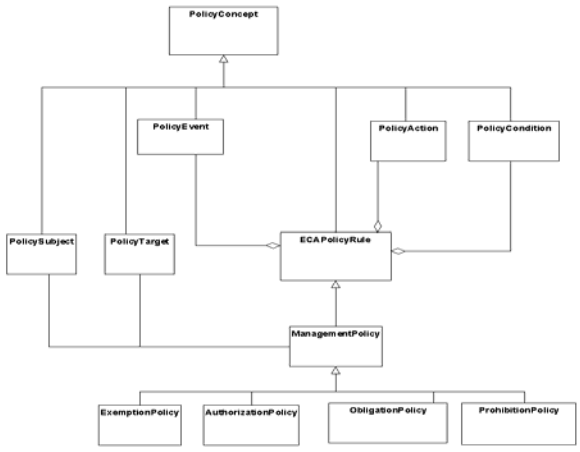

Figure 7. Simplified DEN-ng Policy Model

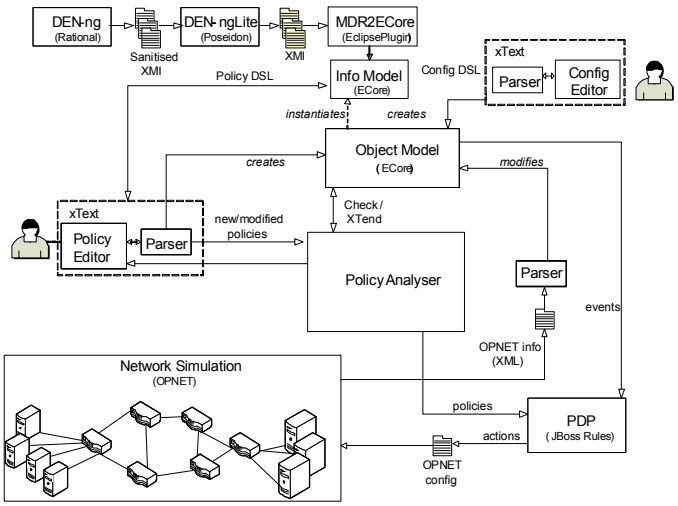

Figure 8. Test Bed Architecture.

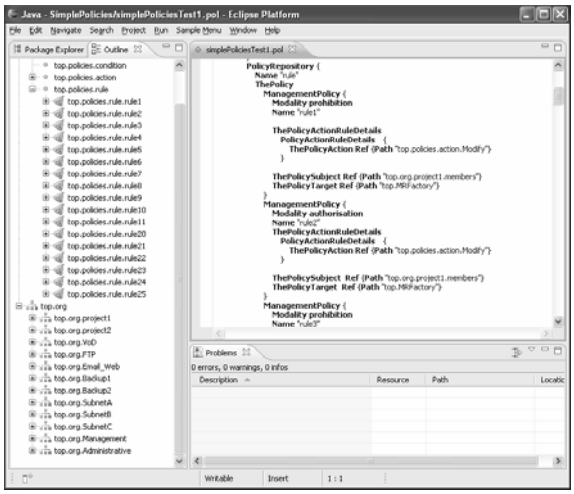

Figure 9. Policy Editor GUI.

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, where as complex entities such as customer and internet service can be represented in the information model.

B. Policy Test Bed

The test bed is an initial implementation of a policy based management system for a communications network, as described in detail in [3]. The test bed emulates management of a set of Internet services for a set of customers subscribed to an Internet Service Provider. The network 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. 8. 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), which provides the model query language xTend, that can be used to query any ECore-based information model. Using xTend we developed 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. 9. 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.

IV. CONFLICT ANALYSIS SCENARIO

In this section we illustrate the operation of the conflict analysis via a policy scenario implemented in our test bed. The scenarios collectively relate to a set of policies defined by an ISP's network administration users to effect management of network services and resources. The management domains of users and services are as depicted in Fig. 10. The first two cases described here concern access control policies, demonstrating detection of subject/target overlap and temporally overlapping conditions with contradicting actions conflicts respectively. The third case concerns filtering policies, demonstrating how different conflict patterns can be used to reflect the particular policy enforcement behavior associated with, for example, firewall rules.

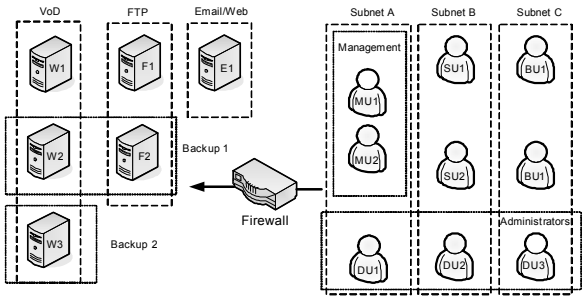

Figure 10. Conflict Analysis Scenario.

TABLE I
ACCESS CONTROL POLICIES.

1. Subnet_A users ***canAccess*** {VoD,FTP,Email/Web}
2. Subnet_B users ***canNotAccess*** {VoD}
3. Subnet_B users ***canAccess*** {FTP, Email/Web}
4. *Administrator users* ***canReboot*** *{Backup1, Backup2} on ServerWarningEvent*

A. Case 1: Access Control Policies – Contradicting Actions

The administrator defines groups of policies that maintain access to specific sets of services available within the network. Table I outlines four policies defined by the network administrator. Policies are presented here using a pseudo policy language for the purpose of brevity, since the actual language used in the test bed is intentionally verbose (see [3] for full details). The following is a brief run through of the conflict analysis algorithm with policies 2 and 4 as input, where policy 4 is being added and is therefore the candidate policy. By comparing the subject sets of the two policies the algorithm can establish that policy 2 is correlated with policy 4. The targets of the policies are also correlated, and the actions of the two policies are contradicting. A conflict matrix specified to detect for subject/target correlation and action contradiction will detect a potential conflict among policies 2 and 4. As we can maintain information relating to the specific relationships among the policies that led to conflict, this information can be displayed to the policy author in order to give them more information as to why the conflict occurred, and possibly aid in the resolution of the conflict.

The calculation to discover conflict in phase two is depicted in fig. 11. The left hand side matrix represents the relationships between policies 2 and 4. The 1 in the first row and the fourth column represents a subject correlation relationship. The next row represents a target correlation relationship. The events and condition are deemed equal for the moment, whereas the actions are deemed to contradict, represented by a 1 in the fifth row. The matrix on the right hand side of the operator is the conflict pattern matrix. The first and second rows represent that any relationship among the subjects or targets can be considered as signifying potential conflict. The next two rows specify that the events must equal, the conditions must equal and the actions must contradict. After the computation a potential conflict is detected and flagged to the policy author. Feedback to the policy author about the potential conflict is depicted in the dialog box in fig. 12. Information concerning the subject, target and action relationships are displayed to the user, along with which conflict pattern detected the conflict.

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

Fig 11. Phase 2 Computation for Access Control Conflict

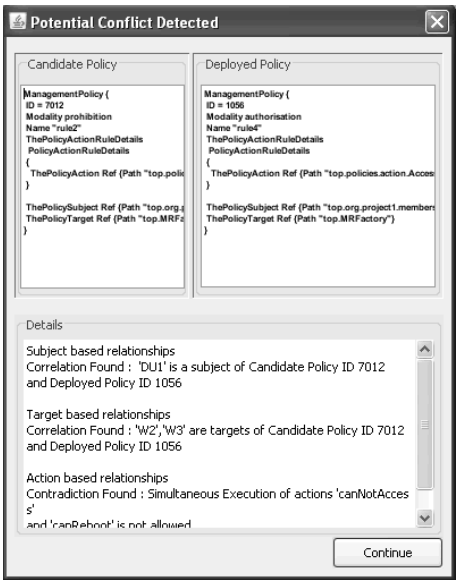

Figure 12. Conflict Analysis Feedback

B. Case 2: Access Control Policies – Temporal Conditions

This scenario examines additional relationships that can arise among policies. The condition correlation relationship is asserted if the conditions related to two policies can both be true at some point in time. Determining this fact is related to the nature of the conditions. A temporal expression is a very common policy condition and there have been many studies published examining the temporal semantics of policies. Temporal conditions can be used to specify when a policy

TABLE II
ACCESS CONTROL POLICIES WITH TEMPORAL CONDITIONS.

1. Subnet_A ***canAccess*** {VoD,FTP,Email/Web} *during interval* 7am-7pm
2. Management ***canNotAccess*** {VoD,FTP} *during interval* 6pm - 12am

should be activated during the day, or at what intervals it should be active from. The DEN-ng information model can represent various forms of time and intervals and is compliant with RFC3339. The policies outlined in Table II describe access restrictions placed on Internet services for the Subnet A group of users and the Management group of users. Policy relationship analysis shows that the subjects are related because the Management group of users is a subset of the Subnet A group of users. The targets similarly overlap; however, it is only because the conditions are correlated that a conflict can potentially occur (i.e., both conditions are satisfied between 6pm to 7pm, during which time both policies apply and a conflict can occur). To detect these types of conflict the

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

Fig 13. Phase 2 Computation for Filtering Control Conflict

TABLE III

FILTERING POLICIES.

1. IP.subnet **matches** 64.10.10.0/24 **Mark** DSCP AF11
2. IP.subnet **matches** 64.10.11.0/24 **Mark** DSCP AF12
3. IP.subnet **matches** 64.10.12.0/24 **Mark** DSCP AF13
4. IP.subnet **matches** 64.10.10.2/32 **Mark** DSCP AF11
5. IP.subnet **matches** 64.10.11.2/32 **Mark** DSCP AF11
6. IP.subnet **matches** 64.10.12.2/32 **Mark** DSCP AF11

condition correlation relationship must be marked in the conflict matrix.

C. *Case 3: Filtering Policies – Shadowing Conflicts*

Network traffic filtering policies define behavior at a low level of abstraction within the network – they are used to configure entities such as firewalls. Packet filtering policies can be described to follow the typical event-condition-action (ECA) pattern, where the event clause is the receipt of an IP Packet, the conditions are expressed as a match against the fields in IP Packet header, and the action is determined by the requirements of the filter. Therefore, packet filtering policies are targeted at constrained models of policy customized for each configuration domain. Filtering policies are typically expressed as an ordered list of policies, where the first policy matched in the list is executed, and the rest are ignored.

The policies under consideration are shown in Table III, and apply to an interface of the firewall device that gives users access to services, as shown in Fig. 10. For this scenario, subnet A users get more bandwidth than subnet B and subnet C users. Administrators are distributed across each of the domains, but should always be awarded the highest bandwidth, regardless of the subnet through which they attach to the network. If the filtering policies for subnets A, B and C are deployed before the filtering policies of the administrator, a shadow conflict occurs because the administrator's traffic will always be matched before the administrator specific rules can be applied as they are added to the bottom of the list of filtering policies. This conflict can be detected by using our algorithm by defining the conflict matrix to detect for policies that have overlapping subjects, identical targets (router interfaces), identical events (IP Packet Received events), and differing actions (mark using different code points). The condition relationship of interest is the subset relationship. The related matrix computation is outlined in fig. 13. This conflict pattern can detect shadowing conflicts only; other conflict pattern matrices can be used to detect generalization and redundancy conflict types.

V. RELATED WORK

Popular policy languages such as Ponder [4], Rei [5], KAoS [6], and XACML [7] 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 satisfied by two of the most cited networking models, CIM and DEN-ng.

An access control policy conflict as described by Lupu and Sloman [8], occurs when a set of simultaneously applicable policies result in multiple decisions being equally applicable. A pre-condition for access control policy conflict as discussed in [8] 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. Unlike our approach, this work ignores 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 together.

Bandara et al. [9], describe a formal model of policies that takes into account simple policy events and conditions. Policy conflicts are described as specific rules encoded in the logic programming language Prolog. They dictate the use of custom logic programs to describe cases for conflict, whereas our approach makes use of an interface to any compatible information model to hold application specific conflict patterns, therefore eliminating 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 apply.

Reasoning about access control policies is the primary focus of work detailed by Jajodia et.al [10, 11]. They developed a formal access control model that enabled the coexistence of multiple policies in a single system and introduces features such as rule propagation and conflict resolution policies. The access control model we assume in this paper is considerably simpler than that proposed in [10], however as our approach is independent of policy model used a richer policy model may be incorporated into the information model.

Policy ratification work by Agrawal et. al [12] investigated the problem of policy conflict from the perspective of the condition component associated to a policy. They implemented a set of algorithms to analyse policies for consistency, coverage and conflict. At the moment we do not related policies together in the method that they suggest in [12], however in future research our policy relationship matrix may be expanded to include concepts such as coverage and consistency of policy conditions.

A taxonomy of policy conflicts among firewall filtering policies [13, 14] and among network security filtering policies [15] 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 [16], describe firewall anomalies or conflicts in terms of subset, superset and correlation relationships among

the match criteria of IP packets. By their use of data mining techniques, they illustrate that rich sets of information can be drawn from analysing the relationships among policies in this way. Our approach may be extended in the future to incorporate enhanced forms of policy relationship analysis to make our policy conflict analysis algorithm more powerful.

VI. CONCLUSION

This paper presented a two phase application domain independent policy conflict analysis algorithm. Operation of the algorithm was demonstrated in a scenario with three cases, examining conflicts involving access control and firewall filtering policies. The algorithm can be applied to arbitrary domains so long as the assumptions related to the policy language are satisfied and the interface to the information model is also 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 matrix, it can be easily extended to include other policy relationship types. As the algorithm makes use of a generic interface to an information model the model can be changed depending on the domain.

The policy analysis algorithm presented in this paper assumes that all policies can be directly compared to each other. However, the relationships among policies are not as straight forward when dealing with a policy continuum. Policy analysis becomes much more complex when multiple levels of policy are considered along with multiple policy authors. In this case, the algorithm must be defined as part of an encompassing process capable of dealing with multiple constituencies' policy authors. We also intend on incorporating ontologies to augment the information model, so that we can represent relationships that are not otherwise possible to represent in the information model. Such relationships that we desire to capture are those that can relate policies together where the relationship is indirectly discovered or is inferred from relationships discovered between policies from diverse application.

ACKNOWLEDGMENT

This work has received support from the Science Foundation Ireland under the Autonomic Management of Communications Networks and Services programme (grant no. 04/IN3/I404C)

REFERENCES

[1] S. Davy, B. Jennings and J. Strassner, "Efficient Policy Conflict Analysis for Autonomic Network Management", accepted for publication in Proc. 5th IEEE Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2008), Belfast, UK, April 2008

[2] J. Strassner, J. N. de Souza, D. Raymer, S. Samudrala, S. Davy, and K. Barrett, "The design of a new policy model to support ontology-driven reasoning for autonomic networking," in Proc. 5th Latin American Network Operations and Management Symposium, LANOMS, Brazil, pp 114-125, 2007.

[3] K. Barrett, S. Davy, B. Jennings, S. van der Meer, and J. Strassner, "A model based approach for policy tool generation and policy analysis," in Proc. IEEE Global Information Infrastructure Symposium, Morocco, pp 99-106, 2007.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in Proc. of the International Workshop on Policies for Distributed Systems and Networks,Bristol, UK, pp. 18–38, 2001.

[5] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in Proc. of the 4th International Workshop on Policies for Distributed Systems and Networks, Como, Italy pp. 63–74, 2003.

[6] A. Uszok, J. M. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lot, "Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement," in Proc. of the 4th International Workshop on Policies for Distributed Systems and Networks, Como, Italy., 2003.

[7] S. Godik, T. Moses et al., "eXtensible Access Control Markup Language (XACML) Version 1.0," OASIS Standard, February, 2003.

[8] E. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," IEEE Transactions on Software Engineering, pp 852-869,1999.

[9] A. K. Bandara, E. C. Lupu, and A. Russo, "Using event calculus to formalize policy specification and analysis," in Proc. of the 4th International Workshop on Policies for Distributed Systems and Networks, Como, Italy, 2003.

[10] S. Jajodia, P. Samarati, M. Sapino, and V. Subrahmanian. Flexible support for multiple access control policies. ACM Transactions on Database Systems (TODS), 26(2) pp 214–260, 2001.

[11] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. ACM Transactions on Information and System Security, 6(2) pp286–325, 2003.

[12] D. Agrawal, J. Giles, K.-W. Lee, and J. Lobo. "Policy ratification".in Proc. of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, pages 223–232, 2005.

[13] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," Selected Areas in Communications, IEEE Journal on, vol. 23, no. 10, pp. 2069–2084, 2005.

[14] C. Lin, C. Xue, and L. Zhitang, "Analysis and classification of ipsec security policy conflicts," in Proc. FCST '06. Japan-China Joint Workshop on Frontier of Computer Science and Technology,, pp. 83–88, 2006.

[15] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies," Communications Magazine, IEEE, vol. 44, no. 3, pp. 134–141, 2006.

[16] K. Golnabi, R. Min, L. Khan, and E. Al-Shaer, "Analysis of firewall policy rules using data mining techniques," in Proc of the 10th IEEE/IFIP Network Operations and Management Symposium, (NOMS 2006). pp. 305–315, 2006