The Policy Continuum – A Formal Model

Steven Davy[1], Brendan Jennings[1] and John Strassner[2]

[1] Telecommunications Software & Systems Group,
Waterford Institute of Technology, Cork Road, Waterford, Ireland
{bjennings, sdavy}@tssg.org
[2] Motorola Labs, Schaumburg, IL, USA
john.strassner@motorola.com

Abstract. The policy continuum, one of the cornerstones of autonomic network management, provides a framework for the development of stratified sets of policy languages, tied together by a common information model, each targeting a different constituency of policy author. This helps ensure the consistency of the policies deployed across a system and provides the foundation for powerful policy analysis processes. Whilst the notion of a policy continuum is well known there have been few examples of its realization. One contributing factor is the lack of a formal model of a policy continuum. In this paper we provide such a model and an associated policy authoring process.

1. Introduction

Policy-based network management systems [1] are widely seen as an appropriate management paradigm to facilitate high-level, human-specified cognitive decision-making in network management. As such, they will play a key role in autonomic management systems, acting as the conduit for human guidance of the operation and goals of self-governing systems [2]. Policy-based management systems aims to allow expensive human attention focus more on defining business logic and less on low-level device configuration processes. They achieve this by separating the behavioural rules for network management from the code that realises the functionality of given network devices.

Policies themselves are formulated as event-condition-actions tuples with the semantics of "on event(s), if condition(s), do action(s)." These rules can be specified at different levels of abstraction by different constituencies, from business analysts to vendor-specific device administration experts. Policies can be deployed directly onto network devices (via configurations applied typically through command-line interfaces), or be maintained by separate "Policy Decision Points," which provide actions when notified of events by the network devices. Clearly, large communications networks are hugely complex, dynamic and heterogeneous in nature, thus large numbers of policies must be defined and continuously amended to ensure the network satisfies high-level goals. A major challenge is ensuring that these policies, which are defined by many different policy authors with many kinds of expertise, are consistent with each other and have the desired effect on the network.

One of the factors inhibiting the widespread deployment of policy-based management systems is the nature of the policy languages that have been specified heretofore. Policy languages such as Ponder [3], Rei [4] and Kaos [5] are typically designed with particular application domains in mind and are not readily usable or understandably by all of the constituencies that would need to author policies in a real network management environment. Furthermore, these languages are defined independent of system information models and ontologies, which makes access to the knowledge required for analysis tasks including policy refinement, policy translation and conflict detection more difficult. To combat these shortcomings Strassner [6] introduced the concept of a "Policy Continuum," which aims to enable multiple constituencies, having different concepts and terminologies, to co-define and co-develop policies.

As described later the policy continuum consists of a stratified set of policy languages, each employing terminology and syntax appropriate for a single constituency. These languages being tied together through the use of a common information model in which policies themselves, their events, conditions, actions, subjects and targets are modeled and related to each other. Whilst the notion of a policy continuum has gained acceptance in the network management research community we have yet to see either its realisation in existing policy systems, or its use for development of policy analysis approaches. We believe that one of the reasons for this is the relatively informal nature of the original description of the continuum. In this paper we seek to redress this, by providing a formal model of the policy continuum and an associated high-level policy authoring process. We hope that this will provide an initial framework that will scope the development of policy analysis approaches and provide a starting point for discussions leading to understanding of and agreement on the architecture of large scale policy management systems.

The paper is structured as follows. Section 2 provides a more detailed introduction to the policy continuum and outlines high-level requirements for its use in the context of policy authoring processes. Section 3 provides the formal specification of the policy continuum and outlines the associated policy authoring process. Finally, section 4 summarizes the paper and outlines future work relating to realization of the policy continuum.

2 Policy Continuum Concept and Requirements

Most policy-based management systems define the notion of a policy as a single entity. From the perspective of a large-scale network management system this is much too limiting. For example, there are policies to represent business rules, policies to control customer rebates, and even policies to represent configuring a feature of a device. There is little in common with these policies, because they use different grammars to express their function, and are used by different constituencies. However, they are often in reality related to each other since they either effect the same or similar behavior in the network, or their behaviors are dependent on each other. We say that individual policies therefore embody different aspects, or views, of a set of related policies, which taken together effect a particular behavior in the

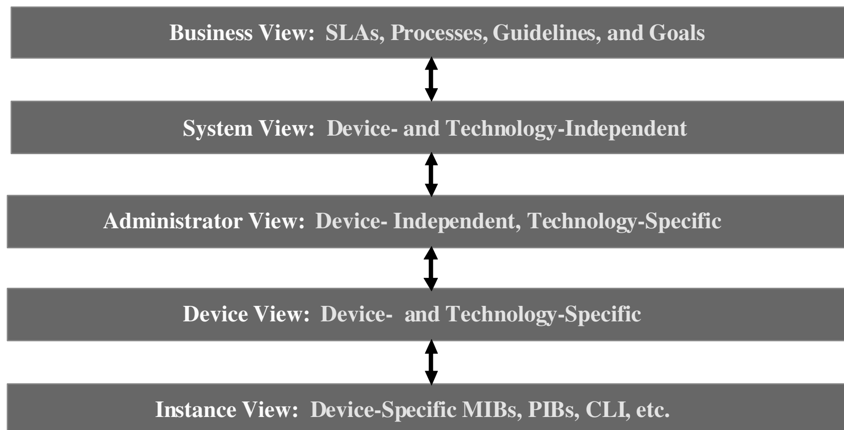

Figure 1: Policy continuum with five views, as per DEN-ng.

network. This observation is the basis for the policy continuum, which has generally been depicted via the typical realization shown in Fig. 1.

Each level of the policy continuum is optimized for a different type of constituency that needs and/or uses information of a specific nature. For example, the business user wants Service Level Agreement (SLA) information, and is not interested in the type of queuing that will be used by routers in forwarding traffic within the core network. Conversely, the administrator of the network may want to develop specific commands to program a router, and may therefore need to have a completely different representation of the policy. The policy continuum thus consists of a number of constituency specific views (five in Fig. 1.), each providing a set of terminology and a policy language appropriate to the needs of that constituency. The views being tied together by an information model that defines the scope of the views and specifies the inter-view relationships required to relate together policies defined by the different constituencies. The DEN-ng information model [7] provides explicit support for the continuum views depicted in Fig. 1. In previous work [8] we have shown how DEN-ng can be used to semi-automatically generate constituency specific policy languages and associated editors and analysis components.

The concept of a policy hierarchy was introduced by Moffett and Sloman [9], 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 [10], 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 [9] and [10], and the policy continuum as defined in [7], and in this paper 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 specifically, creating or modifying policies at low levels can affect the objectives of higher level policies. 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.

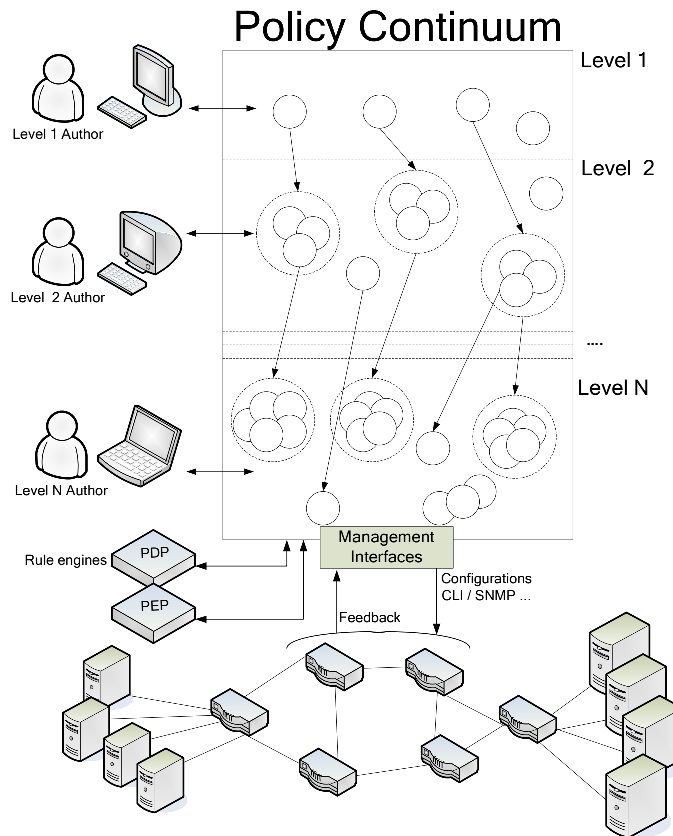

Figure 2: Conceptual Policy Continuum

It is important to note that the policy continuum is in essence a modeling tool. Firstly, it facilitates representation of policies at a high-level of abstraction, which is useful for capturing desired behavior in terms of abstract concepts such as products, services, or prices. Secondly, it provides a means of relating these high-level policies to concrete configurations (for example, router traffic policing configurations or firewall filtering rules) that should effect the desired behavior in the network. Policies at higher levels of the continuum are only deployed in the sense that they are explicitly related to the configurations that have been applied to network devices. Given this the choice of the number of views in the continuum and the scope of each view is arbitrary, although we believe the five views used in the DEN-ng realization of the policy continuum reflect natural spheres of concern in network management.

The special relationships among policies defined at different levels of the policy continuum creates dependencies among the sets of policies, 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 dependencies among the policies at the various

levels are closely aligned to the dependencies of modeled entities as defined in the information model. 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.

Fig. 2 demonstrates three specific properties of policies that exist within the policy continuum. 1) A policy may exist at any level of the continuum without the requirement of being associated to policies at other continuum levels. This property enables policies to exist solely to manage entities that are relevant only to a particular constituency of policy authors. For example, an administrator may decide to configure a routing device to provide static path through the network to support redundancy, however this policy is not directly related to supporting higher level policies. 2) A policy may reference a set of lower level policies. The lower level policies are there to support the operation of the higher level policy. 3) A policy may be associated to more than one higher level policy. Therefore a policy may be re-used to fulfill the objectives of several higher view policies, this may be the case when multiple business policies require common behaviour to be exhibited from a firewall device, or router.

The level in which a policy appears typically defines how the policy will ultimately be enforced. Policies at lower levels of the policy continuum can be enforced directly by system entities such as access control servers, firewall devices and routers. As these devices can be configured to behave in very specific ways, the policies are transformed into appropriate configuration code that in turn enables the devices to behave in the correct manner. Higher levels up the policy continuum require policy decision components and policy enforcement components to enforce the required behaviour onto the managed entities. In these circumstances the policies are not transformed into configuration, but into rule engine scripts that can react to events, evaluate conditions and enforce actions on target managed entities. Higher level policies can affect the behaviour the rule engines enforce, therefore creating a stack of behaviour, each affecting the level below.

2.1 Information Model Assumptions

As we make extensive use of the information model in describing the policy

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

Figure 3: Information Model Interface

continuum, we must outline the assumptions placed on the information model interface. We assume that external applications have available to it a range of functions for querying the information model. Fig 3 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, 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.

2.2 Policy Continuum Requirements

This sub-section focuses on the requirements of a structure to perform as 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.

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

Analysing Policies

Once a policy has been created it must be analysed for correctness, we assume there is a separate process for syntactic analysis. However the policy should also be analysed for conflicts among policies specified at the same level of the policy continuum. 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. The interfaces provided must be capable of supporting complex processes and algorithms that are contained within a policy authoring 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.

3 Policy Continuum Formal Model

This section presents a formal structural model for policy, and extends this model to be part of a policy continuum model. We specify base operations that are essential to enable a policy authoring process to manipulate the policy continuum.

3.1 Basic Policy Model

A policy is made up of sets of events, conditions, actions, subjects and targets. A policy therefore aggregates those events, conditions, actions, subjects and targets that it is interested in into a policy structure. The map function PolicyMap is used to associate this information to a policy (1). Therefore when this map is applied to a policy identifier, a tuple of information is returned. Following is a description of formally specified operations that can be implemented to create and modify a single policy structure. The following listed operations can be used by external processes and algorithms as an interface to create and modify policies used to manage the target system. We present three basic operations, that of *CreatePolicy*, *AddEventToPolicy* and *RemoveEventFromPolicy*. The adding and removing of conditions, action, subjects and targets repeat the pattern demonstrated in adding and removing events, and thus they are omitted.

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

CreatePolicy
When a policy is created, it is not associated with any nested sets of events, conditions, actions, subjects or targets. At this point the policy is incapable of invoking any sort of behaviour within the managed system. In the operation stated in (2), *p* is the new policy identifier and *pm* is the policy map, *PolicyMap*. Therefore the operation results in modifying *PolicyMap*, by adding a new mapping where *p* is mapped to a tuple of policy components. We observed that all of the components that comprise *p*'s tuple are empty sets, demonstrating that there are currently no events, conditions, actions, subjects or targets associated.

$$\begin{aligned} &CreatePolicy : \left(Policy \times PolicyMap\right) \rightarrow \left(PolicyMap\right) \\ &CreatePolicy\left(p, pm\right) \triangleq \left[\, p \rightarrow \left(\varnothing,\varnothing,\varnothing,\varnothing,\varnothing\right)\right] \cup pm \end{aligned} \tag{2}$$

AddEventToPolicy
An event can be added to an existing policy in the policy map; in this case the set of events that the policy is associated to is modified. This operation is useful when the policy author needs to add a specific trigger to the policy so that it will be evaluated in different circumstances. In operation (3), *se* refers to a new set of events. The union of this new set of events, with the existing set of events creates an updated set of events that this policy is associated. The operator used in this circumstance is the map

override operator which will override what the policy p is currently mapped to and subsequently maps it to a new tuple of policy components. The map index operator π^i is used to index into the tuple and retrieve information at a specific index. Therefore we use it to keep the unmodified information associated to the policy

$$AddEventToPolicy : (Policy \times \mathbb{P}Event) \rightarrow (PolicyMap \rightarrow PolicyMap)$$

$$AddEventToPolicy(p, se)\, pm \triangleq pm \dagger \left[p \rightarrow \begin{pmatrix} se \cup \pi^1 \circ pm(p), \\ \pi^2 \circ pm(p), \\ \pi^3 \circ pm(p), \\ \pi^4 \circ pm(p) \\ \pi^5 \circ pm(p) \end{pmatrix} \right] \quad (3)$$

RemoveEventFromPolicy
Similarly an event can be removed from an existing policy, by modifying only the set of events associated with that policy (4). This operation may be used if the policy author needs to reduce the circumstances for which the policy is evaluated. Here the set of events are subtracted from the existing set of events currently associated to the policy, therefore reducing the event set associated to the policy p.

$$RemoveEventFromPolicy : (Policy \times \mathbb{P}Event) \rightarrow (PolicyMap \rightarrow PolicyMap)$$

$$RemoveEventFromPolicy(p, se)\, pm \triangleq pm \dagger \left[p \rightarrow \begin{pmatrix} se \setminus \pi^1 \circ pm(p), \\ \pi^2 \circ pm(p), \\ \pi^3 \circ pm(p), \\ \pi^4 \circ pm(p), \\ \pi^5 \circ pm(p), \end{pmatrix} \right] \quad (4)$$

3.2 Policy Continuum Model

The policy continuum is modeled as a recursive tree of policy objects where the definition of a policy object is specified in the previous section. The specification detailed in equation (5) describes that a policy p is an element of the set policy.

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.

$$\begin{aligned} & p \in Policy \\ & pr \in PolicyContinuum = Policy \rightarrow (\mathbb{N} X \mathbb{P} Policy) \end{aligned} \quad (5)$$

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 then one policy at a higher level. All policies within the policy continuum must be directly accessible from the PolicyContinuum, however this does not mean that each policy is at the highest level.

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 structure to store and retrieve policies.

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 associated to policy at any other level of the policy continuum. In the operation defined in (6), *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.

$$\begin{aligned} &AddPolicy : (\mathbb{N} \times Policy) \rightarrow \left(PolicyContinuum \rightarrow PolicyContinuum\right) \\ &AddPolicy(n, p)\, pr \triangleq pr \cup \left[\, p \rightarrow \left(n, \varnothing\right) \right] \end{aligned} \tag{6}$$

RemovePolicy

A policy should be removed using this function (7) only if it does not reference lower level policies, i.e. there are no dependencies among policies. Therefore this operation will only function is 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.

$$\begin{aligned} &RemovePolicy : Policy \rightarrow \left(PolicyContinuum \rightarrow PolicyContinuum\right) \\ &RemovePolicy(p)\, pr \triangleq pr \setminus p \end{aligned} \tag{7}$$

UpdatePolicy

Updating policies (8) involves changing the associated policy set of a given policy. If the policy is updated then the goals of the higher level policy may be jeopardized, therefore care must be taken when using this function. 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.

$$\begin{aligned} &UpdatePolicy : Policy \times \mathbb{P}Policy \rightarrow \left(PolicyContinuum \rightarrow PolicyContinuum\right) \\ &UpdatePolicy(p, ps)\, pr \triangleq pr \dagger \left[\, p \rightarrow \left(\pi^{1} \circ pr(p), ps\right) \right] \end{aligned} \tag{8}$$

GetPolicyChildren
We can retrieve the child policies of a specific policy (9) 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.

$$\begin{aligned}
&GetPolicyChildren : Policy \to \left(PolicyContinuum \to \mathbb{P}Policy\right)\\
&GetPolicyChildren(\varnothing) \triangleq \varnothing\\
&GetPolicyChildren(p)\,pr \triangleq\\
&\quad \forall p_n \in \left(\left(I \to \pi^2\right) pr\right)(p) : GetPolicyChildren(p_n)\\
&\quad \cup \left(\left(I \to \pi^2\right) pr\right)(p)
\end{aligned} \tag{9}$$

GetPoliciesAtLevelN
We can retrieve all policies at a specific level of the policy continuum (10). 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.

$$\begin{aligned}
&GetPoliciesAtLevelN : \mathbb{N} \to \left(PolicyContinuum \to \mathbb{P}Policy\right)\\
&GetPoliciesAtLevelN\left(n\right) pr \triangleq dom\left(I \to\lhd \left[\pi^1 = n\right]\right)\left(pr\right)
\end{aligned} \tag{10}$$

GetAllPolicies
We can get all individual policy objects (11). Since all policies must at least be expressed in the domain of the policy continuum map.

$$\begin{aligned}
&GetAllPolicies : PolicyContinuum \to \mathbb{P}Policy\\
&GetAllPolicies(pr) \triangleq dom(pr)
\end{aligned} \tag{11}$$

GetPolicyParents
We can get all policies that reference a specific policy (12), i.e. those policies at a higher level. 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.

$$GetPolicyParents : Policy \rightarrow \left(PolicyContinuum \rightarrow \mathbb{P}Policy \right)$$
$$GetPolicyParents(p)\, pr \triangleq \left(\left(\mathrm{I} \rightarrow \pi^2 \right) GetPoliciesAtView\left(\pi^1 \circ pr(p) - 1 \right) \right)^{-1} (p) \quad (12)$$

GetCommonPolicies
We can discover those lower level policies that are in common to two given policies (13). 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.

$$GetCommonPolicies : (Policy \times Policy) \rightarrow \left(PolicyContinuum \rightarrow \mathbb{P}Policy \right)$$
$$GetCommonPolicies(p_a, p_b)\, pr \triangleq GetPolicyChildren(p_a) \cap GetPolicyChildren(p_b) \quad (13)$$

Summary of Policy Continuum
We have seen that the policy continuum is a complex structure, and we have outlined an interface to it. The interface provided enables the addition, removal and update of policies to the policy continuum; however these operations should be used under specific circumstances. We have not yet defined how to add, modify or delete a policy that is dependent on higher or lower level policies. As these more delicate operations require further analysis of the policy continuum before they can be executed, they are defined as part of an encompassing policy authoring process discussed in the next sub-section. Search operations are also provided to enable us to further examine policy relationships. The next sub-section outlines a continuum sensitive policy authoring process that makes use of the policy continuum model and interface.

3.3 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. 4 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 the 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. 5 depicts a flow diagram explaining the process from the perspective of a policy author at a particular level.

$$
\begin{aligned}
&\textbf{\textit{ModifyPolicy :}}\ (Policy \times Policy \times PolicyContinuum) \rightarrow PolicyContinuum\\
&\textbf{\textit{ModifyPolicy}}\,(p_{old}, p_{new}, pc) \triangleq\\
&\forall p_{par} \in \textbf{\textit{GetPolicyParents}}\,(p_{old})\ pc\ \textbf{:}\\
&\quad \textit{if}\ \textbf{\textit{not}}\ \ \textbf{\textit{VerifyPolicyContinuum}}\,(p_{par})\ pc\\
&\quad\ \textit{then}\\
&\qquad \textit{return pc}\\
&\textit{if}\ \textbf{\textit{DetectPolicyConflict}}\,(p_{new})\ pc\\
&\ \ \textit{then}\\
&\quad \forall p_{cnf} \in \textbf{\textit{PotentialConflictList}}\,(p_{new})\ pc\ \textbf{:}\\
&\quad\ \ \forall p_{par} \in \textbf{\textit{GetPolicyParents}}\,(p_{cnf})\ pc\ \textbf{:}\\
&\qquad\ \ \textbf{\textit{NotifyCurrentAuthor}}\,(p_{par})\\
&\quad \textit{return pc}\\
&\textit{else}\\
&\quad \forall p_{ref} \in \textbf{\textit{RefinePolicy}}\,(p_{new})\ pc\ \textbf{:}\\
&\quad\ \ \textit{if}\ p_{ref} \in NewPolicy\\
&\qquad\ \textit{then}\ \ \textbf{\textit{CreatePolicy}}\,(p_{ref}, pc)\\
&\quad\ \textit{elseif}\ \ p_{ref} \in ModifiedPolicy\\
&\qquad\ \textit{then}\ \ \textbf{\textit{ModifyPolicy}}\,(p_{ref}, pc)\\
&\textbf{\textit{CommitChange}}\,(p_{old}, p_{new}, pc)
\end{aligned}
$$

Figure 4 : Policy Authoring Process

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

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

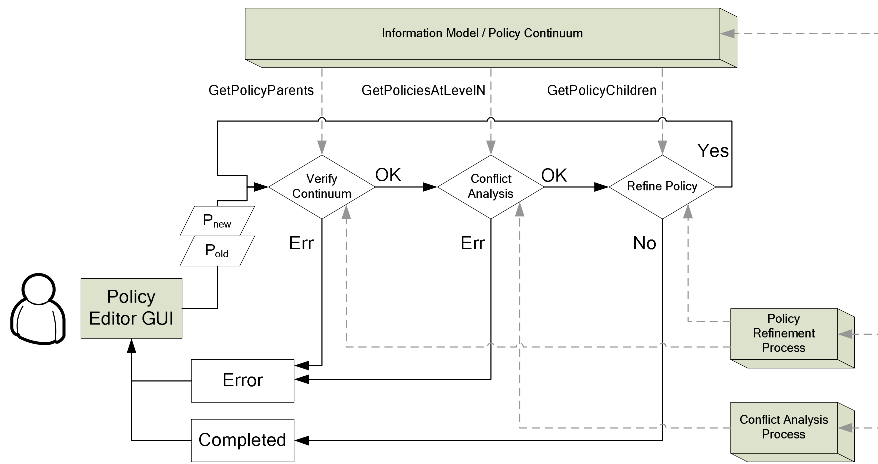

Figure 5 : Policy Authoring Process

effective solution. Specifically the algorithms are *DetectPolicyConflict* and *RefinePolicy*. We do not specify these algotithms in this paper, however the interfaces defined to the information model and indeed the policy continuum provide such algorithms with the ability to examine the dependencies among policies and to modify these dependencies as they see fit.

The above specified policy authoring process is described from the perspective of adding or modifying a policy, when removing a policy the same process is applied, where we need to investigate if the removal of a specific policy jeopardizes the objectives of dependent policies.

4 Summary and Future Work

We present a formal model of the policy continuum, with accompanying policy authoring process. Operational semantics for basic policy and 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. The policy model and continuum are tightly coupled to a system information model. This enables application specific concerns to be decoupled from the policy authoring process, thus making our approach applicable to arbitrary application domains.

In future work we investigate the processes involved in policy conflict analysis that are independent of application domain and continuum level; also we investigate processes for policy refinement for the policy continuum. We intend on incorporating ontologies to augment the information model, so that we can represent relationships that are not otherwise possible, leveraging these relationships to improve policy conflict analysis and refinement.

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/I404C).

References

1. Sloman, M., Policy driven management for distributed systems, J. Netw. Syst. Manag., vol. 2, no. 4, Dec., pp. 333-360. (1994)
2. Jennings, B. et al., Towards Autonomic Management of Communications Networks and Services, to appear, IEEE Commun. Mag., Oct. (2007)
3. Damianou, N., Dulay, N., Lupu, E., and Sloman, M.,: The ponder policy specification language, Proc. of the Int'l Workshop on Policies for Distributed Systems and Networks, pp. 18–38. , (1995)
4. Kagal, L., Finin, T., and Joshi, A.: A policy language for a pervasive computing environment, Proc. 4th Int'l Workshop on Policies for Distributed Systems and Networks, pp. 63–74. (2003)
5. Uszok, A. et al.,: Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement, Proc. of 4th Int'l Workshop on Policies for Distributed Systems and Networks , (2003.)
6. Strassner, J.:Policy-Based Network Management, Morgan Kaufmann, ISBN 1-55860-859-1, Sept. (2003.)
7. Strassner, J.:DEN-ng: achieving business driven network management, Proc. 8th IEEE/IFIP Network Operations and Management Symp. (NOMS 2002), IEEE, pp. 753-766. , (2002)
8. Barrett, K., Davy, S., Strassner, J., Jennings, B., van der Meer ,S. and Donnelly,W.: A Model Based Approach for Policy Tool Generation and Policy Analysis," Proc. 1st IEEE Int'l. Global Information Infrastructure Symp. (GIIS 2007), IEEE, 2007, pp. 99-106. (2007)
9. Moffett, J. D. and Sloman, M. S.:Policy hierarchies for distributed systems management, IEEE Journal on Selected Areas in Communications, vol. no. 9, pp. 1404–1414, (1993)
10. Wies, R. :Using a classification of management policies for policy specification and policy transformation, Integrated Network Management IV, vol. 4, pp. 44–56,(1995)