Conflict Prevention via Model-driven Policy Refinement

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

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

Abstract. This paper describes an approach for application specific conflict prevention based on model-driven refinement of policies prior to deployment. Central to the approach is an algorithm for the retrieval of application-specific data from an information model relating to the subject and targets of a given policy. This algorithm facilitates the linkage of policies loosely defined at a high level of abstraction to detailed behavioural constraints specified in the information model. Based on these constraints policies are then modified so that conflicts with other deployed policies can be readily identified using standard policy conflict detection techniques. This approach enables policy enforcement to be cognisant of application specific constraints, thereby resulting in a more trustworthy and dependable policy based management system.

1 Introduction

This paper presents an approach for refinement of newly created or modified policies so that application specific conflicts with already deployed policies can be readily prevented. We propose the use of a policy analyser that can interrogate an information model containing detailed information about the system for which policy is being defined, and use this information to refine the high level policy into a policy embodying information regarding system constraints its actions may be subject to.

This paper describes the operation of a policy analyser, and a prototype implementation demonstrating its use in a policy based management system. The paper is structured as follows. §2 discusses current work on policy conflict detection and prevention, and on methods for analysing information contained within an information model. §3 presents an architecture for policy conflict prevention and specifies the algorithms for retrieval of relevant information and policy refinement. Our prototype implementation is described in §4, whilst its operation in an experimental test bed is described in §5. Finally, §6 summaries the paper and outlines topics for future work.

2 Related Work

This section discusses published work in the domains of policy conflict analysis and information model processing.

2.1 Policy Conflict Analysis

Policy conflict detection and resolution is a necessary component of any Policy Based Management System (PBMS). A PBMS must employ a facility to verify that newly created or modified policies conform to intended system behaviour before they can be deployed. From the perspective of our approach, a policy conflict can be seen to be a potential occurrence of unintended behaviour within the PBMS. This can manifest itself in many forms. Most have been documented by Charalambides, et al. [1], who categorise conflicts as domain independent or application specific. Domain independent conflict analysis can be carried out by offline processes that indicate whether conflicts will definitely occur or may occur in a specific context. If we can detect the conditions in which a conflict can occur, then we can resolve the conflict by either modifying or removing one or more conflicting policies. The issue, of course, is if we have enough knowledge to detect all conditions in which a conflict can occur. For example, if conflicts are known at design time, then one can devise strategies to deal with them. However, in networking, one often encounters conflicts at run-time which were not envisaged during the design period. Hence, the challenge is to design a robust conflict detection approach that can deal with unforeseen situations.

Detection of application specific conflicts requires more information about the system for which policies are being defined. In [2] the authors augment the PBMS with extra information, expressed as rules relating to the managed entities. These rules are triggered when an application specific conflict is about to occur; such conflicts are resolved based on specific resolution policies associated with each of these rules. This approach depends on the policy author both being able to specify system constraints that policies must adhere to, and being able to translate these constraints into the appropriate custom rule format. In [13] Shankar and Campbell use pre-conditions and post conditions to describe the effects specific actions will have on a system, they use this axiomatised rule-actions to help in the conflict prediction process. These again have to be encoded into the policies to be effective.

2.2 Information model processing

An Information Model is a representation of managed entities, concepts and their relationships independent of platform, language, and protocol. Information models play a central role in network management and considerable efforts have been expended on the specification of standard information models. One of the more mature standards is the TM-Forum's Shared Information and Data Model, which is closely related to DEN-ng [3]. One of the main advantages of DEN-ng is its extensive use of patterns and abstractions (such as roles) to allow behaviour to be defined and

orchestrated over the associated components of the system being described. Use of an information model of a system to aid in policy based management is also described in [4], aimed at managing specifically IP networks, and more recently towards autonomic communication networks [5].

For model-driven policy refinement we are specifically interested in efficient retrieval of relevant information from a system model. For UML-based models like SID/DEN-ng a number of approaches for information retrieval exist. One such method described in [11] details how the UML artefacts used to build a class diagram describing an information model can be translated to an ontology where it is represented in OWL (Web Ontology Language). This ontology can then be reasoned over and queried using semantic web technologies. A benefit of this approach is the ability to use existing ontologies to expand the information model such as linking to user profiles.

Another approach would be to translate the UML into an XML format such as XMI (XML Metadata Interchange) [8] where it can be efficiently queried over using XQuery. XQuery provides an efficient method of querying repositories of XML documents within an XML database. Meier [9] describes the performance of such an XML database called eXist, where test documents of about 40 Megabytes can be efficiently queried. Information model repositories generated from UML to XMI are not expected to reach this size.

3 Description of Approach

Model-driven policy conflict prevention is the process of refining newly created or modified policies so that conflicts with already deployed policies can be readily detected using standard policy conflict detection approaches. Policy refinement in this context involves the specification of additional condition clauses within the policy, which subsequently allows the detection of conflicts with other policies that would otherwise have gone undetected by standard policy conflict detection algorithms.

More specifically, in cases where system information models describe constraints relating to the operation of managed entities, relevant policies can be augmented with conditions reflecting these constraints, so that they will not be enforced in a manner that results in these constraints being violated. System constraints in the information model are defined by the system architect who has expert knowledge in the functionality of the system being modelled. These system constraints may come in the form of action pre-conditions, invariants, or post-conditions. However the policy authors, be they business analysts or network administrators, have vastly differently views of the system for which they are defining policy. Therefore they have an incomplete view of the system as a whole. System constraints defined within the information model can help bridge this gap by supplying implicit knowledge not usually available to the policy authoring process. Our approach is to introduce an automated policy refinement process which obviates the need for policy authors to be cognisant of the detailed constraints on system operation, but which outputs policies that are sufficiently well specified that policy conflict detection processes can be

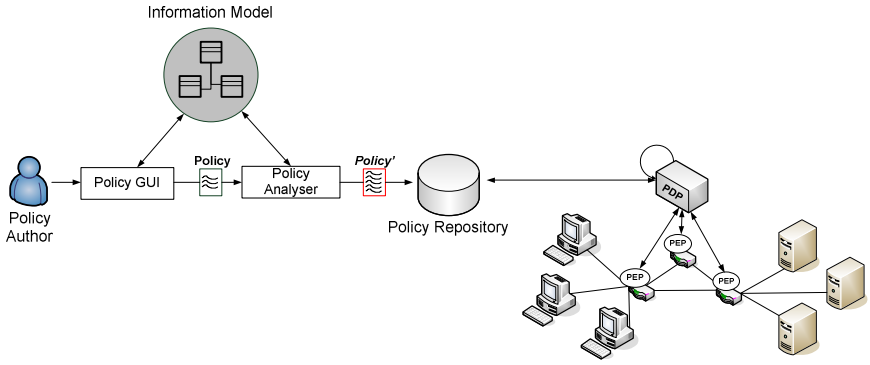

Fig. 1. PBMS Architecture incorporating model-driven conflict prevention.

effective and efficient. Our approach is primarily concerned with action pre-conditions or action constraints.

3.1 PBMS Architecture Incorporating Policy Conflict Prevention

Fig. 1. illustrates a PBMS architecture incorporating model-driven conflict prevention. We now briefly describe the role of the Policy GUI, the information model and the Policy Analyser. The Policy GUI is the interface used by policy authors who are primarily concerned with ensuring that services and resources are managed in a manner consistent with business objectives and goals. Policy authors are likely to be business analysts who define or modify policies relating to particular customers and their access to the services provided by the network. They are unlikely to have the detailed knowledge of the network required to specify policies at the level of detail required for easy detection of conflicts with other deployed policies.

The information model describes, in a platform independent manner, the characteristics and behaviour of the different managed entities comprising the managed environment, as a set of related *model elements*. Model elements include classes, attributes, relationships, constraints, and other artefacts. For example, the information model will describe which customers can use which services where and how. Constraints within the information model can be described using a constraint language like the Object Constraint Language (OCL) [7]. OCL specifies constraints using invariants, pre-conditions and post conditions associated with all attributes, associations and operations on each modelled class.

Policies created or modified by policy authors are expressed in strict accordance with the terms used in the information model, since the policy GUI is tightly coupled to the information model, as described in [5]. Once created/modified policies are passed to the Policy Analyser, which takes their subjects and/or targets and queries the information model for relationships (and constraints on these relationships) for these subjects/targets. Using relationship and constraint information it is possible to assess more precisely those circumstances in which the policy actions should be invoked. To achieve this, the Policy Analyser employs an algorithm that retrieves the relevant relationships and constraints from the information model given an arbitrary

```
Inputs [Policy]
Outputs [Relationships and Constraints]
   List Subjects defined in Policy
   List Targets defined in Policy
   List Actions defined in Policy
   For every element of Subjects
          Subject Managed Entities = Look up the corresponding Class
          descriptions from the Information Model
   For every element of Targets
          Target Managed Entities = Look up the corresponding Class
          descriptions from the Information Model
   For every element of Target Managed Elements,
          If there is an Action requested by the Subject Managed Entity
          define within the Target Managed Entity that matches the
          Action in the Policy then add the pre-conditions of this
          action to the relationships and constraints list.
Return (Relationships and Constraints)
```

Fig.2. Policy Action Constraint Retrieval Algorithm

policy defined in accordance with that information model. Such an algorithm is specified in §3.2 below.

3.2 Policy Action Constraint Retrieval Algorithm

In specifying an algorithm for policy action constraint retrieval we firstly assume that policies specify the policy subject using the terms used in the information model (e.g. there must be a one-to-one, or one-to-many, mapping between a policy subject and a class in a UML based model). The target(s) of the policy, if included, must also be similarly specified. If the target is not specified explicitly, it must be possible to infer it from the information model by examining the relationships between the subject and the actions. Finally, policy actions must map to relationships between those model artefacts representing the policy subjects/targets.

Given these assumptions the algorithm outlined in Fig.2 provides a means of discovering the relevant policy action constraints based on model artefacts and their relationships.

3.3 Policy Refinement Algorithm

Once the associated relationships and constraints have been retrieved, the original policy needs to be refined. As there may be multiple action constraints to be added into the policy, they must first be checked against each other so that the resulting policy action constraints do not logically contradict. An example of this would be if two constraints were added to a policy specifying that the action may only be performed during daytime hours, and another constraint specifying that the action may only be performed during night time hours. This type of rule contradiction will cause the policy not be enforced at anytime, and so the policy can not be refined and

```
Inputs [(Relationships and Constraints); Policy Conditions]
Outputs [newPolicy]
   For every Relationship select PolicyAction.PreConditionsConstraint
        A Pre Condition Constraint is selected from each Relationship
        and tested against all previously selected Constraints
        For each Constraint in Constraints and Policy Conditions
             If newConstraint AND Constraint
                is a Logical Contradiction
             Then
                The Condition Clause of the Policy will never be
                satisfied and the algorithm is aborted
             Else
                Add newConstraint to the list of Constraints
Combine the resulting list of constraints to the Policy Conditions as
new conditions
Return newPolicy
```

Fig.3. Policy Refinement Algorithm

is invalid against the referenced information model. The constraints must also be checked against existing policy conditions for completeness.

4 Prototype Implementation

The prototype implementation, depicted in Fig.4., will now be described. The Policy GUI is developed in Java, and enables the policy author to create high level policy using context sensitive drop down menus. A detailed description of this GUI can be found in [5]. The options available to the policy author are limited to the entities describe in the information model, so that subject, targets and actions must be specified in the information model before they can be used to define policies. The policies output from the GUI are defined from the view the policy author has of the managed system. This allows the policy author to only be concerned with authoring policy appropriate to his level of knowledge, while enabling the policy analyser to develop more specific instances of this policy.

The Policy Analyser is a Java process that is invoked on every new or modified policy. Access to the information model is performed by processing a set of XML files that represent the information model. The information model is initially described using a UML class diagram editor, and is exported to an XMI [8] format. XMI is the OMGs (Object Management Group) standard format for describing UML diagrams, however only the class diagram aspects of the standard are of interest for the moment. The information model constraints are defined in a separate OCL file. The OCL constraints are translated from managed entities action pre-conditions into policy conditions that can be understood by the policy repository and policy analyser via Kent OCL library [10]. This library provides java class implementation of OCL constraints that can be evaluated in real-time. The policy repository takes two forms; policies are stored in an XML format for query and retrieval using eXist XML database for storage and XQuery for searching; they are also stored in a JBoss rules engine in working memory, where reasoning over policies is performed. Policies

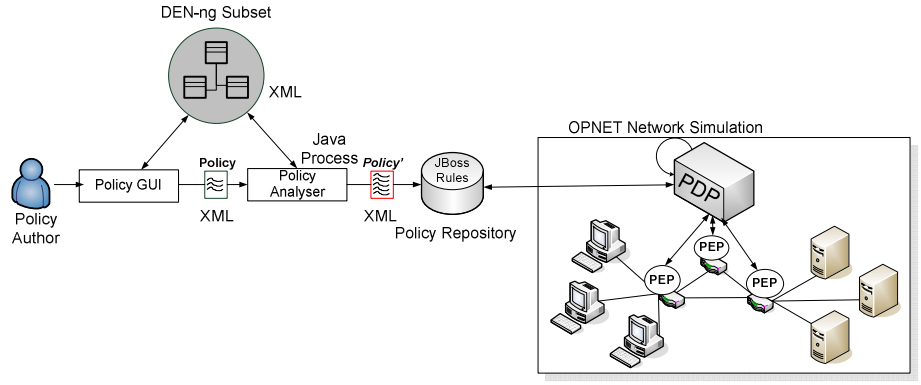

Fig. 4. Prototype Implementation

stored in the JBoss rules engine [12] are encoded as Java Bean objects, so a simple policy class hierarchy is used. The JBoss rules engine also holds a runtime representation of the data defined in the information model, such at router information and link information which is updated at regular intervals.

Some simple policy types are defined such as permit, obligation, and refrain. Policies added to the JBoss rule engine can be rapidly reasoned over to discover whether there are any domain independent conflicts, such as a conflict of modality. The rule engine can also detect if two policies referring to the same action and target will potentially cause a conflict when the conditions are satisfied.

The system being managed is simulated with OPNET, allowing for flexibility at the network level where it is easy to modify the underlying network scenario. PDPs receive updated policy and enforce it through the simulated PEPs. A more detailed description of the simulated system is provided in [5].

5 Scenario and Results

This section describes a scenario where there are two customer networks subscribed to services provided by a single Internet Service Provider (ISP). Our ISP has defined a simple information model (using a subset of DEN-ng) and policies as follows.

5.1 High Level Policies and Information Model

The policies will describe the conditions as to when a certain customer is permitted to request provision of RTP (Real Time Protocol) traffic for its usage.

There may be several similar policies defined for other customers of the system where they too are permitted to request the allocation of bandwidth. There may also be policies defined not by the business user but by the network administrator that will also require the allocation of bandwidth. When the defined policy in Fig.5 is enforced, the core network will modify the PHB (per hop behaviour) of the edge and core routers to reflect the provision of the requested service. We can see this interaction

```
<policy name="WITServicePolicy" type="permit">
     <subject type="Customer">WIT</subject>
     <event type="From">09:00</event>
     <event type="To">17:00</event>
     <event type="Trigger">RequestRTPSession</event>
     <operation>
            <target type="RouterLink"/>
            <action type="AllocateBW">
                   <param name="grade" value="1"/>
                   <param name="amount" value ="5Mbps"/>
            </action>
     </operation>
     <condition/>
</policy>
<policy name="TSSGServicePolicy" type="permit">
     <subject type="Customer">TSSG</subject>
     <event type="From">08:00</event>
     <event type="To">16:00</event>
     <event type="Trigger">RequestRTPSession</event>
     <operation>
            <target type="RouterLink"/>
            <action type="AllocateBW">
                   <param name="grade" value="1"/>
                   <param name="amount" value ="4Mbps"/>
            </action>
     </operation>
     <condition/>
 </policy>
```

Fig.5. High Level Policies

modelled in the information model in Fig. 6 below. As Fig. 6 shows, a customer can subscribe to the RTP service which uses resources such as the EdgeRouter and the CoreRouter.

Focussing on the RouterLink managed entity; there are two operations available for this scenario – allocation and deallocation of bandwidth. We will now discuss the former, as the latter is very similar. `AllocateBW()` will instruct the nested core and edge routers to configure their PHBs to reflect the request. As there are always limited resources on the network, we cannot keep calling `AllocateBW()` and expect bandwidth to be always available to allocate. OCL is used to define the semantics of these attributes and the following OCL is attached to the `AllocateBW()` operation to constrain its use concerning how bandwidth can be allocated.

```
context RouterLink::AllocateBW(ToS:Integer, amount:Real)
pre perserveBWLimit: self.currentBW + amount <self.maxBW
```

When the original policy is run through the policy analyser it is refined with information describing more accurately when the policy should be actually enforced. The algorithm defined in Fig.2 is implemented as a set of XQuery functions where the policy document is input. For example the subjects of the policy can be discovered using the XQuery terminology, `doc("policy.xml")/policy/subject/@type`, which will return a type represented by the subject mention in the policy. Similar statements can retrieve the targets and actions of the policy. XQuery is also used to query the

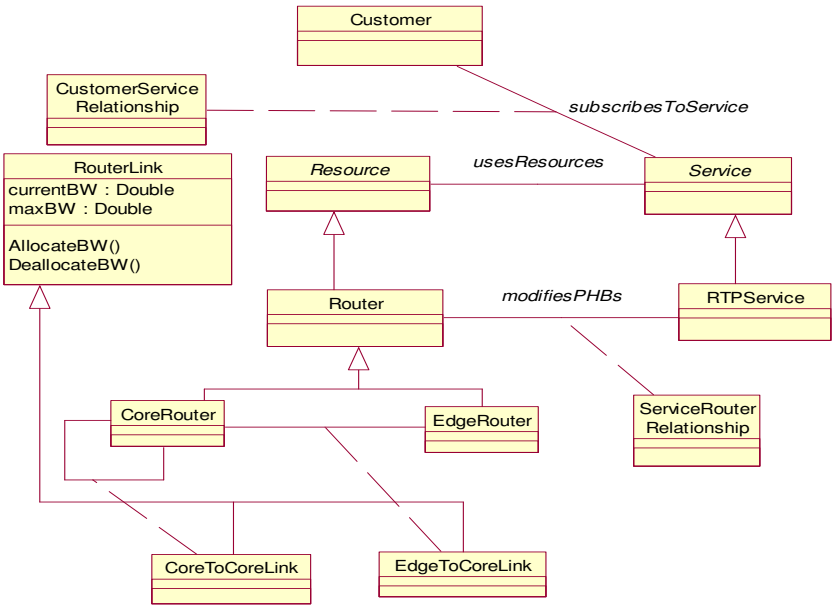

Fig.6. Den-ng Subset Information Model

XMI representing the information model. To look up a class entity's id the query below can be used.

```
for $x in doc("InfoModel.xmi")//*[@name]
     where (compare(name($x),'UML:Class') = 0) and (compare($x/@name,
     'Service') = 0)
     return string($x/@xmi.id)
```

Once we have the id of the policy entities we can then discover further associations, and relationships between other entities using the information model. The algorithm finishes with selecting the appropriate OCL from the OCL files; this is easily carried out because every OCL statement includes a context mentioning the

```
<policy name="WITServicePolicy" type="permit">
     <subject type="Customer">WIT</subject>
     <event type="From">09:00</event>
     <event type="To">17:00</event>
     <event type="Trigger">RequestRTPSession</event>
     <operation>
            <target type="RouterLink"/>
            <action type="AllocateBW">
                   <param name="grade" value="1"/>
                   <param name="amount" value ="5Mbps"/>
            </action>
     </operation>
<condition>RouterLink.currentBW + 5 < RouterLink.maxBW</condition>
</policy>
```

Fig. 7. Modified Policy

reference class and actions it is constraining. The Kent OCL library then processes the OCL and generates the extra policy conditions required to refine the associated policy. The policy in Fig.7. is a refined policy from Fig.5.

The policy defined in Fig.7 describes an extra condition of which the original policy author would not be aware. The clause is evaluated in real-time when the policies are being processed to see if they apply at the current situation. This new information will further constrain when the policy will be valid. The Kent OCL library generates a java bean that will evaluate this condition for the JBoss rule engine during analysis and at runtime.

5.2 Policy Enforcement

Suppose that the two original policies were deployed to the system, and currently the `currentBW` and `maxBW` of the related RouterLinks are 0.0Mbps and 8.0Mbps respectively. An event of type Request RTP is initiated by the customer WIT at approximately 08:15am. This event triggers the enforcement of the relevant policies allocating 5Mbps of bandwidth over the related RouterLinks (first policy enforcement in Fig. 8.). An event of type Request RTP is then initiated by the customer TSSG at approximately 9:40am (second policy enforcement in Fig. 8.). This triggers an attempt to allocate a further 4Mbps of bandwidth on the related links. However an application specific conflict occurs that was not detected previously, whereby more bandwidth is being allocated than is available. The effects of allowing this conflict to go "untreated" are unpredictable, as the situation is not catered for. From Fig.8 we see that the allowable capacity of the core link is 8 Mbps, and as the new RTP session was allowed, it can only be partially met. Also, this will adversely affect other existing sessions.

Now suppose the original policies were analysed and refined to reflect the

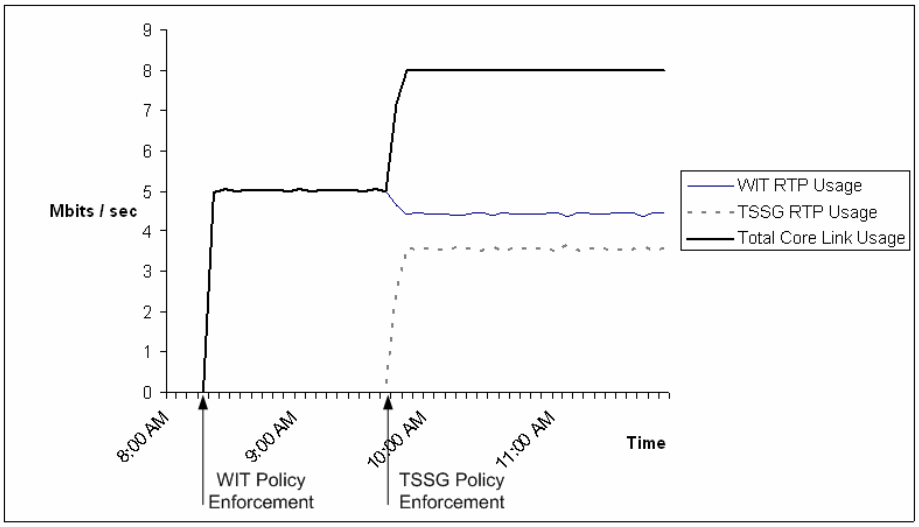

Fig. 8. Application Conflict Illustration

constraints specified within the information model. The policy information added in Fig.7 is added to both policies. In this updated scenario, the first event still succeeds, but the second event does not trigger the permit policy and is discarded, as the policy will not meet all of its conditions. Specifically, when the condition clause of the policy is checked, it is evaluated to false because the `currentBW` plus the requested bandwidth exceeds the `maxBW` of the related RouterLinks.

6 Conclusions and Future Work

Policy conflict situations, when not catered for, will allow the system being managed to produce unpredictable behaviour. This is an undesirable scenario for potential ISPs looking to employ policy based management to control the behaviour of their network. This paper introduces an architecture and prototype implementation that refines high level business policies with application specific information so that conflicts can be readily detected. This form of conflict prevention is made possible using an information model defined over the services and resources of the system, where the constraints of the system are defined by a domain expert. Algorithms that process a policy in order to retrieve constraint information and subsequently refine the policy are defined and implemented. A model-driven approach to refining policies towards conflict prevention frees the business user from being concerned with the behavioural details of the core network, and introduces a level of safety and dependability into the system. One potential downside is that certain business policies may not be enforced as originally described, thus provision of appropriate feedback to the policy author would be desirable.

Future work will be focused on developing our algorithm to be used with existing policy languages and policy based management systems such as Ponder [6]. We also intend on developing a richer information model along with a set of obligation, permit and refrain policies to investigate what other information can be used from the information model to aid in conflict prevention. We also intend on exploring other aspects of Information Models that define system behaviour such as flow charts and finite state machines.

Acknowledgements

The authors would like to take this opportunity to thank the anonymous reviewers for their useful comments and feedback on the paper. 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. Charalambides, M. et al., “Policy Conflict Analysis for Quality of Service Management,” in Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks POLICY'05, Stockholm, Sweden (2005) 99-108
2. Charalambides, M. et al., “Dynamic Policy Analysis and Conflict Resolution for DiffServ Quality of Service Management” in Proceedings of the IEEE/IFIP Network Operations and Management Symposium 2006, Vancouver, Canada (2006) 294-304
3. Strassner, J., “Policy-based Network Management: Solutions for the Next Generation”, Morgan-Kaufman Publishers. ISBN 1-55860-859-1 (2004)
4. Strassner, J., “Directory Enabled Networks”, Macmillan Technical Publishing, ISBN 1-57870-140-6, (1999)
5. van der Meer, S., Davy, A., Davy, S., Carroll, R., Jennings, B., Strassner, S. 2006, "Automonic Networking: Prototype Implementation of the Policy Continuum", in Proc. Workshop in Broadband Converged Networks at IEEE/IFIP Network Operations & Management Symposium, Vancouver, Canada (2006)
6. Damianou, N., Dulay, N., Lupu, E.C., Sloman, M., “The Ponder Specification Language,” presented at 2nd IEEE Workshop on Policies for Networks and Distributed Systems, Bristol, UK (2001)
7. OMG, UML 2.0 OCL Specification v2.0, Object Management Group Specification, Available at http://www.omg.org/docs/formal/06-05-01.pdf: accessed July (2006)
8. OMG, Meta Object Facility (MOF) 2.0 XMI Mapping Specification, v2.1, Object Management Group Specification, Available at http://www.omg.org/docs/formal/05-09-01.pdf accessed July (2006)
9. Meier, W., eXist: An Open Source Native XML Database, In Web, Web-Services, and Database Systems. NODe 2002 Web- and Database-Related Workshops, Erfurt, Germany (2003) 169-183
10. Akehurst, D., Patrascoiu, O., OCL 2.0 – Implementation the Standard for Multiple Metamodels”, Workshop Proceedings, 6th International Conference on the Unified Modeling Language and its Applications, UML2003, Electronic Notes in Theoretical Computer Science (2003)
11. Lehtihet, E., Strassner, J., Agoulmine, N., O Foghlu, M., Ontology-Based Knowledge Representation for Self-Governing Systems, accepted for publication in 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2006, Dublin, Ireland, October (2006)
12. JBoss Rules, JBoss, Available at http://labs.jboss.com/portal/jbossrules/ accessed August (2006)
13. Shankar, C., and Campbell, R., A Policy Based Management Framework for Pervasive Systems using Axiomatized Rule-Actions in Proceedings of the 2005 Fourth International Symposium on Network Computing and Application (NCA’05), Cambridge, Massachusetts, July (2005) 255-258