

# Policy Interactions and Management of Traffic Engineering Services Based on Ontologies

Steven Davy<sup>1</sup>, Keara Barrett<sup>1</sup>, Martín Serrano<sup>2</sup>, John Strassner<sup>3</sup>, Brendan Jennings<sup>1</sup>, Sven van der Meer<sup>1</sup>

<sup>1</sup>TSSG, Waterford Institute of Technology, Ireland  
{sdavy, kbarrett, bjennings, vdmeer}@tssg.org

<sup>2</sup>Universitat Politècnica de Catalunya - UPC, Barcelona, Spain.  
mserrano@nmg.upc.edu

<sup>3</sup>Motorola Labs, Schaumburg, IL, USA  
john.strassner@motorola.com

**Abstract:** Trends such as converged networks and services require multiple and diverse networks and technologies to interoperate. Autonomic technologies are emerging as an important way to manage next generation services. This paper outlines an approach for lifecycle management of services and applications that require specific levels of quality of service. Our approach uses ontologies to create interoperability across different management domains using semantic reasoning, leveraging policy based management techniques to achieve autonomic behaviour. We define an architecture named PRIMO (Policy Relations and Interaction for Management using Ontologies), which provides a platform for our research. The main concern is to deliver to end users context-aware and QoS-dependant services while complying with business objectives of the service providers and network operators. We use ontologies to represent both the data and relationships between the data for all stakeholder views; this enables the interaction of various management views, especially those of the service provider and the network operator. We outline our approach for policy transformation and interaction between different management systems and show how context information, represented via ontologies, is a critical facilitator for this approach.

**Keywords:** Autonomic Systems, Autonomic Networking, Traffic Engineering, Traffic Services, Policy Management, Context Information, Next Generation Services, Ontology-Based Management.

## I. INTRODUCTION

In recent years, the business and technical aspects of systems has increased dramatically in complexity, requiring new technologies, paradigms and functionality to be introduced to cope with these challenges. The drive for more functionality has dramatically increased the complexity of the systems, so much so that it is now almost impossible for a human to manage all of the different operational scenarios that are possible in today's complex communication systems. The stovepipe systems that are currently common in OSS (Operational Support Systems) and BSS (Business Support Systems) design exemplify the necessity to incorporate the

best implementation of a given feature; unfortunately, this focuses on the needs of individual applications as opposed to overall system needs, and thus inhibits the sharing and reuse of common data. This is a specific example of the general inability of current management systems to address the increase in operational, system, and business complexity [1]. Autonomic solutions are designed to provide self-management.

Recently, several initiatives have used policy based management approaches to tackle the problem of fast and customizable delivery of QoS commitments (e.g., OPES [2] and E-Services [3]). Policy based systems for network and service management are used for governing the behaviour of a system separate from the functionality that the system provides. In other words, policy management provides the ability to govern network operations and adapt the services and resources that the network offers in response to changing user demands, business requirements, and environmental conditions.

Research into policy based management for networks and services has focused on developing languages and architectures for managing and deploying policies in distributed environments. However, policy conflict detection and resolution, along with policy refinement and policy reasoning, continue to be unsolved areas of policy based management. Unfortunately, the current state of the art in policy management dictates targeted solutions for different functions. For example, if security and network configuration are both desired to be governed using policy, the usual solution is to deploy separate policy servers. This complicates system governance, since now the interaction between policies from different contexts and abstraction levels (e.g., service and network) become more complex. Our research activity is primarily concerned with the use of policy for managing services in autonomic environments that use next generation networks [1][4].

Policy interaction is the concept of linking the affects of one policy to triggering other policies independent of the type or abstraction level of the policy. For example, there is a fundamental difference between an *obligation policy* and an *authorization policy*: the former defines what actions a subject is permitted (or not permitted) to perform on a target,

and hence is *target-based enforcement*; the latter defines the actions a subject must do to a target, and hence is *subject-based interpretation and enforcement*. As another example, policies can exist at multiple abstraction levels – this is codified as the Policy Continuum [5]. From the perspective of our research, policy interaction is the explicit triggering of policies across management boundaries and abstraction levels in order to achieve a specific goal or behaviour, where policies from all interacting management domains can affect each other. In this research, policies defined for service lifecycle management from the perspective of the service provider interact with policies defined for engineering traffic in a network for the network provider. This is different than policy refinement, where policies at one level are derived from policies at a higher level (e.g., policies controlling services are used to derive policies that control the network that provides those services). Our approach is focused on solutions that enable multiple policy-based systems that use their own, different, policy models to interoperate with each other. This is done by using ontologies and semantic integration, and is triggered by the events signaled between the involved domains and the policies at all appropriate abstraction levels.

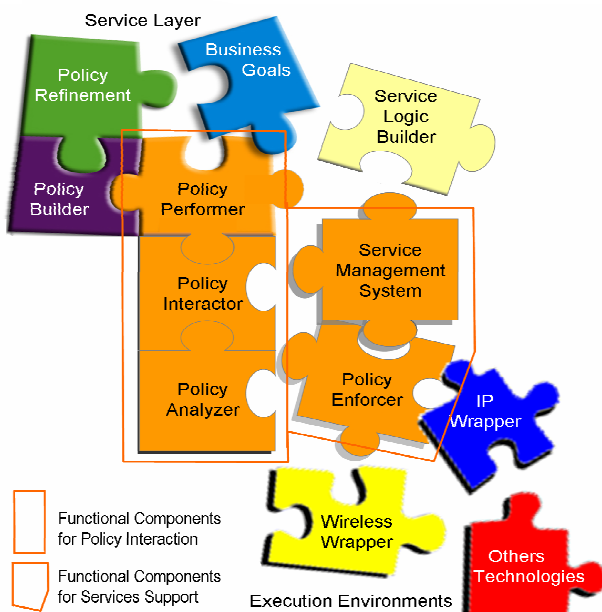


Figure 1. Functional Components for Policy Interactions and Services Support.

In previous work [6], we have proposed the combination of policies and context information in a formal and efficient way for supporting and managing services. In this paper we go one step further and present an architecture that is intended to support high-level policy service interactions to control the network traffic from many diverse stakeholders, including content providers, service providers and network operators. This is shown in figure 1, where the key elements involved in the policy interactions and described in this paper are the Policy Performer, Policy Analyzer and the Ontology-Based Policy Interactor. These three components provide the framework for supporting policy interactions with Policy-Based Service Management Systems. The core idea is to provide and intermediary system to act as mediator solution

between multiple policy-based systems and ontology-based solutions. We believe that if an architecture is able to interact with ontology-based policy models, the exchange of information and the policy-based management activity can be done transparently as result of an information interoperability process. This approach manages the full service life cycle by using ontology-based policies, which represent the desired semantics to be enforced. This enables the framework to take into account the variations in context information (contained in traffic engineering policies), and relate those variations to changes in the services operation and performance (e.g., QoS). The synergy obtained from the policy interactions, the ontology-based data representations and the policy-based paradigm represents the knowledge engineering that autonomic systems provide [7].

This paper provides an overview of our approach. In section II we briefly describe the concepts of ontology-based context and policy interaction, and how each relates to autonomic systems. Section III explains how policies interact to provide service management in autonomic environments. Section IV presents the validation and results for our policy-based management system using an autonomic scenario. Section V presents the results for our policy-based management system in reference to the related approaches. Finally, the conclusions are presented in Section VI, and are accompanied with acknowledgements and bibliography references.

## II. ONTOLOGY-BASED CONTEXT & POLICY INTERACTIONS

This section describes the policy interaction process. Context awareness and ontology usage are related to this activity to highlight their importance towards self-management in autonomic communications.

### A. Policy Interactions

Typically, a policy is defined in and of itself to manage a specific unit of behaviour associated with the managed system. The definition of policy in accordance with DEN-ng is as follows: “Policy is a set of rules that are used to manage and control the changing and/or maintaining of the state of one or more managed objects” [5].

Therefore, the task of a policy is to direct the behaviour of a set of managed objects by governing the state of the managed objects. In some circumstances, the management of the behaviour of a set of managed objects may be controlled by multiple domains. For example, managing the end-to-end QoS provided for one or more applications usually involves coordinating the behaviours of multiple managed objects in multiple administrative domains. Collaboration is required once the traffic leaves one network provider’s domain and enters another domain. One method of addressing this concern is to enable policies to interact across the domain boundaries so that the combined behaviour induced by the set of policies produces the overall intended end-to-end result. There are several issues that arise, however, when realizing this approach. The main concern is that a level of trust must be attained among the collaborating parties and the task of

triggering the correct set of relevant policies across domain boundaries.

The DEN-ng policy model can support the concept of policy interaction, where the simplest method is by exposing the fact that events can be initialized, during or on the completion of a set of actions that are associated with a policy. The effect of this is depicted in Figure 2.

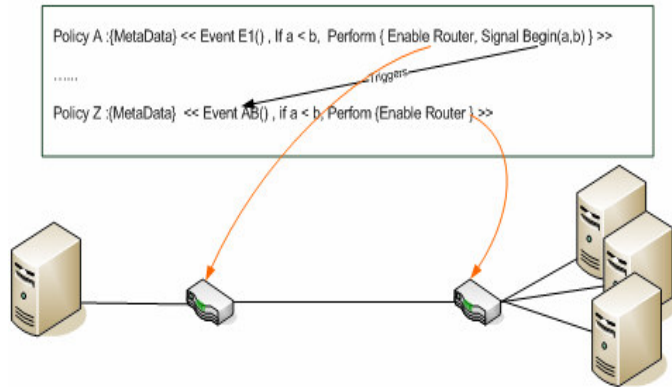


Figure 2. Policy Interaction

Figure 2 shows that an action associated with policy A is responsible for creating a signal; this signal is then used to trigger policy Z. By designing policies in such a matter, the responsibility for achieving a specific behaviour from the managed objects is distributed across a set of policies, therefore enabling a degree of collaboration between high level policies and low level policies.

### B. Policy Models for Traffic Engineering

This section describes how traffic engineering policies are used for supporting new user-oriented services. From the perspective of this work, traffic engineering policies are related to the management of network interfaces to provide specific QoS guarantees for classes of services over an IP network using Differentiated Services (DiffServ) traffic conditioning. The behaviour that is required is that of identifying and conditioning IP flows throughout the paths in a network. Traffic identification policies are responsible for identifying specific patterns in the IP Header and subsequently marking a field in the packet, usually the Type of Service field. By identifying flows, we can then define policies to condition the flows, by building appropriate traffic conditioning components, such as markers, droppers, queues, shapers and schedulers. In order to allow for the dynamic creation and deployment of traffic engineering policies, policies are applied to a set of management interfaces; particular policies can be created, removed or modified by the network administrator.

The motivation behind managing these policies is to alter the way the network behaves as various types of traffic and combinations of traffic pass through the network. This abstraction process enables the network administrators to rapidly create and deploy new traffic engineering policies to support new types of services that are being defined as products or services that are sold to (for example) subscribers. Services such as voice over IP and IPTV, can now be described by the service provider, and this description can be used to create or modify network view policies (such traffic

engineering) for the network operator in next generation network environments.

1) *Traffic Engineering Service Provider Policies:* In order to increase the business value of the management interface to the network view of the system, there must be interaction from the service view. The concepts of price, customer, and product are clearly defined at the service view of the system, and there is a requirement to reflect the creation and modification of these concepts in the underlying network. For example, when an existing customer customizes his product offering by adding or varying parameters to services in his product, these changes need to be reflected as configuration changes of the underlying network. Therefore, policy interaction using the Policy Continuum rewards us with the ability to alter the behaviour of the network as a direct affect of modifying service definitions.

The challenge here is how to adequately represent this signal of service modification to the underlying network management interfaces if the network view policies are defined using vastly different concepts, such as Ethernet interface, queue buffer size, and scheduling priority. Since the service view has no requirement to be able to represent its services using network view concepts, there needs to be a semantic bridge between the views of the system. The approach taken in this paper is to use ontology driven policy interaction to provide this semantic bridge.

In triggering policies across views and across domains, an ontology of the interfaces between the boundaries is required. An event ontology associated with domain A is integrated with an event ontology associated with domain B; this provides “hooks” to control communication between the domains, thereby providing interoperability between domains. However, the degree of interoperability that is achieved is directly related to the ability of the semantic integration process in aligning the two ontologies. In this example, the interaction between the service view and the network view is handled by first compiling an ontology of events exhibited from the service view, and then repeating the same operation for the network view. This results in a semantic integration process that can relate one or more concepts in one ontology with one or more concepts in the other ontology. Note, though, that the service view and the network view ontologies are designed independently; therefore, the semantic integration can either be shared or developed independently, as long as the same algorithms are used in the process. Once this integration has been generated (either automatically or manually), service view events that may signal a customization of service grade for a customer can be properly interpreted by the network view as one or more traffic conditioning actions.

### C. Service Management Policies

The service management policies control only the service lifecycle, not the logic of the service. In this way, service management policies are used by the policy-based system components to define the a) code distribution and code maintenance of the service as well as b) the service invocation, c) the service execution and d) service assurance processes. Coming from the general requirements associated

to the service management layer, the listed previous types of policies covering the service lifecycle can be defined, and all those policies related to the services contained in those groups. Following functional and operational aspects, the policies for managing the service lifecycle are structured hierarchically, in terms of PolicySets; this is a DEN-ng class whose two subclasses are PolicyRules and PolicyGroups. PolicyGroups can contain PolicyRules and/or other PolicyGroups, and is realised using the composite pattern shown in figure 3. A PolicySet is defined as either a PolicyGroup or a PolicyRule. The aggregation HasPolicySets means that a PolicyGroup can contain zero or more PolicySets, which in turn means that a PolicyGroup can contain either a PolicyGroup or a PolicyRule. In this way, hierarchies of PolicyGroups can be defined. A simple way to think of this is using the analogy of directory and files. A directory is a container, and can contain other containers or files. A PolicyGroup is a container, and a PolicyRule is a File.

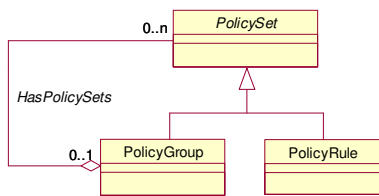


Figure 3. PolicySets

The order of execution of PolicyRules and PolicyGroups depends on the structure of the hierarchy (e.g., grouped and/or nested) and is controlled by the set of metadata attributes previously defined. For robustness and flexibility, DEN-ng enables one or more events, conditions and actions to be grouped together to form a “clause” – this nomenclature was chosen to strengthen the perception of a policy rule as a “statement”. Hence, a policy can be thought of as follows:

```

WHEN an event_clause is received that triggers a condition_clause evaluation
IF the condition_clause evaluates to TRUE, subject to the evaluation strategy
THEN execute one or more actions in the action_clause, subject to the rule
execution strategy
  
```

Thus, we can see that policies are not executed until an event that triggers their evaluation is processed. The event does not have to be actively sent to the autonomic system by a network component (e.g. as in an SNMP alarm) – passive events, such as the passing of time, can also be included. Following the receipt of such an event, it is tested for applicability to the current *context* by examining its condition clause. If it is applicable (i.e., if the condition clause evaluates to TRUE), the appropriate actions are executed.

#### D. Ontology-Based Policies Interactions using DEN-ng Principles

This section introduces how policy interaction can be supported using the DEN-ng information model along with an associated ontology. The main contribution of this paper is to show how interactions between Service Policies and Traffic Engineering Policies occur at different views, or abstraction levels. We use context information applied to appropriate levels of the Policy Continuum and ontologies applied to the engineering to achieve this interaction [8][9].

The mapping from one view of a policy to another view of the same policy is commonly called policy transformation, whether the transformation occurs at the same level of abstraction or between different levels of abstraction. The concept of a Policy Continuum [5] is an example of transforming a policy built for one constituency into a form that accommodates the needs of a different constituency. Since the different views of policies often require different concepts, entailing the need for different vocabularies and grammatical rules, semantic integration techniques are required in the policy transformation process. This is why our approach relies on ontologies. Conceptually, relating one vocabulary and associated set of grammatical rules to another vocabulary and associated set of grammatical rules is done using automated reasoning, which is an integral part of semantic interoperability [10][11]. For example, this enables relationships that have not been explicitly identified to be *inferred*, and is therefore an essential capability for policy transformation.

A couple of assumptions need to be considered at design time in order to take advantage of policy integration:

- The events generated at a specific view are viewed as signaling context information for other views. Since the context information model is shared across views, we can share events across views.
- This sharing of context enhances the value of each view, and enables the functionality being managed by each view to be efficiently integrated. This exchange of information promotes the interoperability of the information between services and systems.

1) *Traffic Engineering And Service Management Policy Interactions*: DEN-ng has defined a rich, extensible model to represent traffic conditioning mechanisms. An overview of this model is shown in Figures 4, 5 and 6.

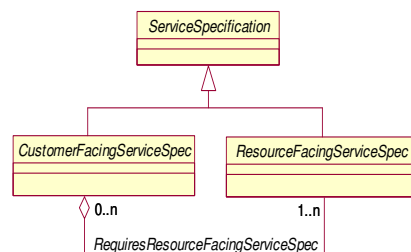


Figure 4. Service and Resource Definitions

Figure 4 shows that a Service specification is made up of customer- and resource-facing services. Customer-facing services are those that are directly visible to the customer; resource-facing services are those that aren’t visible to the customer. For example, a “VPN” is a customer-facing service, since the customer can order it and configure it. BGP and MPLS are resource-facing services, since the customer doesn’t need to know *how* the VPN works.

Figure 5 shows in how the ServicePackageSpec and ServiceBundleSpec define groups of related customer- and resource-facing services that are treated as a single coherent offering. For example, “Gold Service” can represent a package of different applications, such as VoIP, multi-media streaming, and basic Internet roaming. Each of these

applications require different traffic conditioning, which are defined by a ServiceBundle.

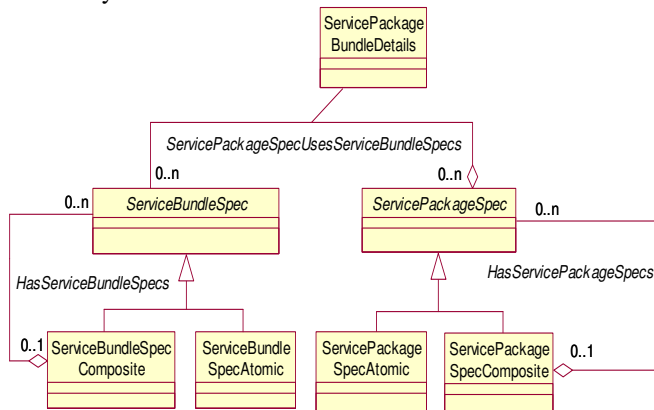


Figure 5. Defining Traffic Conditioning Requirements

A ServiceBundleSpec is a collection of services representing different sets of traffic conditioning settings. For example, a Gold ServiceBundle would provide different traffic conditioning for *each* of the applications in the ServicePackage. The advantage of this approach is that if the Customer later upgrades from Gold Service to Platinum Service, the *same design* is used – only the objects are replaced. Furthermore, any changes to the applications and/or traffic conditioning parameters are handled automatically by the autonomic system, since all that is needed is to replace one object (e.g., Gold Service) with a different object (e.g., Platinum Service).

Figure 6 shows an overview of the DEN-ng QoS model.

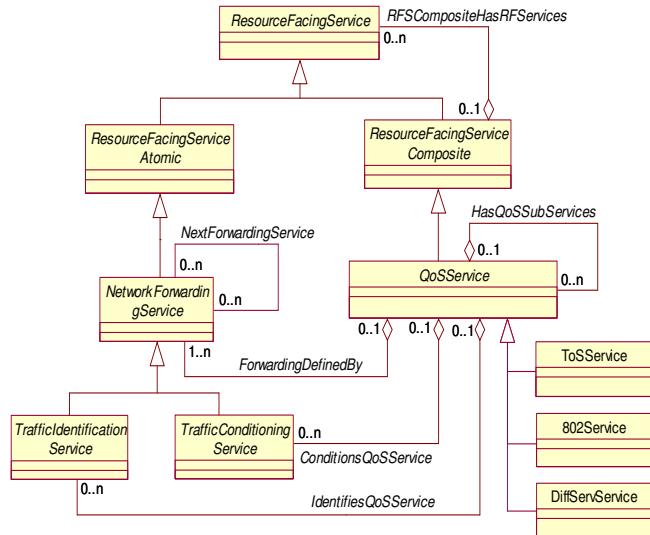


Figure 6. DEN-ng QoS High-Level Model.

Figure 6 shows that a QoSService can be modeled using different technologies (e.g., non-DiffServ using Type of Service (ToS), DiffServ, and even different types of 802 services). Each of these technologies is thought of as a NetworkForwardingService, which has two important subclasses: TrafficIdentificationService (e.g., classification and marking) and TrafficConditioningService (e.g., dropping, queuing, and scheduling). Each of these services are controlled via Policy, as shown in Figure 7. PartyRole is an abstraction that defines the role a set of people or an organization plays; for this discussion, two exemplar roles are Administrator and Technician. The difference is that the

former is permitted to perform more changes than the latter. Each Service and Resource has an appropriate Management Policy that is used to manage it. This Management Policy consists of a set of related Policies that handle different lifecycle aspects of the Service or Resource, such as configuration, maintenance, and retirement.

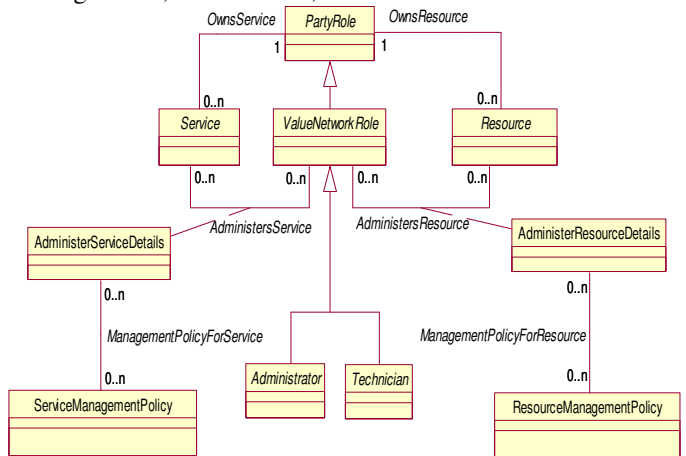


Figure 7. Policy Control of Traffic Conditioning.

### III. ONTOLOGY & POLICY-BASED APPROACH

The objective of this research is to create an automated and domain independent approach to select policy parameters for low level policies (e.g., traffic conditioning) such that the corresponding high level goals (e.g., terms of a service level agreement) are met. These low-level policies may be also used for monitoring the system for goal assessment purposes. The approach is supported by a novel combination of ontology engineering and policy-based management; the added value is the reduced reliance on technological dependencies for services support and the increased interoperability between heterogeneous policy-based service management systems.

The architecture to support Policy Relations and Interactions for Management Operations (PRIMO) is based on the use of formal ontology languages such as OWL [12]. This interaction relies on supporting ontologies using some ontology-based components, as shown in figure 8.

The purpose of the PRIMO architecture is to manage traffic engineering conditions from a high level view and, using the Policy Continuum and ontologies, enable the transition from high-level goals (as codified by service policies) to low-level network view policies. The transition process is not a refinement process itself – rather, policies are transformed in content and representation. We use context information to relate policies at different abstractions to each other, and then use events to trigger and control each set of related policies. Thus, the component will take as input a service view policy, and select a set of network view policies that can meet its goals. The selection process involves analyzing the triggering mechanisms of low level policies as expressed in an appropriate set of ontologies via automated semantic integration, and matches them to the service level policy goals. An outline of this process is depicted in Figure 8. The key enabling concept is the process of semantic

integration that can create associations and relationships among entities specified in different ontologies.

requests to the autonomic system. The framework is composed by components described briefly hereafter.

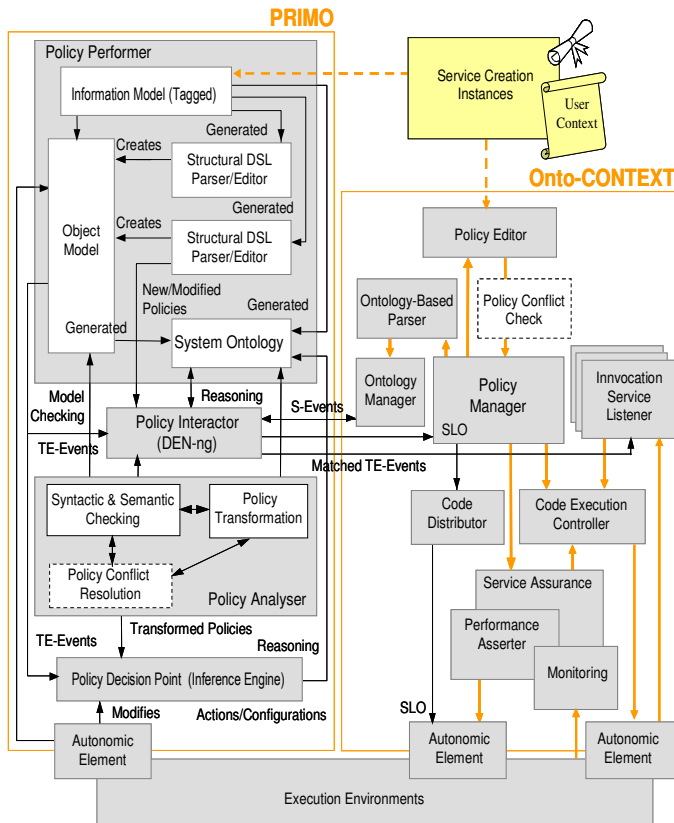


Figure 8. PRIMO architecture to support Policy Relations and Interactions for Management Operations.

In the PRIMO architecture the “Ontological Comparison” describes concepts and relationships between objects defined in the DEN-ng information model, enabling the Autonomic Manager to reason about received data and subsequently infer meaning. For example, we can find out which customers are affected by a particular SNMP alarm, even though this information is not contained in the SNMP alarm, by looking at relationships in both the information model and relevant ontologies and deducing which customers are affected by the SNMP alarm.

The autonomic system then determines whether the actual state of the managed entity is equal to its desired state. If it is, then the system keeps on monitoring the entity. If it is not, then the system defines the set of configuration changes that should be sent to the managed entity. These are translated by the Intelligent Middleware into vendor-specific commands for legacy as well as future devices.

The Autonomic Manager provides the novel ability to change the different control functions (such as what data to gather and how to gather it) to best suit the needs of the changing environment. An important, but future, area of research is to add learning and reasoning algorithms to the system. The former enables the system to grow and refine its knowledge base, while the latter provides the ability to generate hypotheses to more efficiently determine root cause. Finally, the Policy Manager is both an interface to the outside world (GUI and/or scripted) as well as the translation of those

### A. PRIMO Framework Components Description

Research into automatic semantic integration of ontologies is primarily concerned with the dynamic selection and invocation of web services based on some ontological description of what the web service provides. The key research challenge is to devise some methods of assurance that the service policies are actually satisfied by low level policies when the semantic integration is automated across domain boundaries. A review of most popular approaches is discussed in [13].

As a framework example, suppose a service policy exists which specifies that when a new user account is created, a signal “CreateCustomerFlow” is forwarded as a event to the reasoner of the autonomic system. The reasoner has pre-established via semantic integration that this event is equivalent to the set of network events IPAddressDetected, BeginTrafficIdentification and BeginTrafficConditioning. Each of these events can trigger related, lower level events (e.g., BeginTrafficIdentification will trigger the classification and subsequent marking of the traffic). These events are initialised and forwarded to the network Policy Decision Point (PDP) for processing, and the relevant policies are installed and executed across the network. For each policy completed, an acknowledgement is returned to the PDP. The network PDP can forward a set of signals to the service provider PDP representing that it has triggered a number of policies. A reasoner located at the service provider can intercept these events and match them against an equivalent service view event that is subsequently instantiated and forwarded to its PDP, signaling that the goal has been met.

The management of the services is done using a policy based approach that uses context information from low level policies to trigger service management operations as events. These events then trigger the appropriate high level policies, so that business goals can be enforced in the network. Table 1 shows policy examples and the interaction with service management policies. It contains a set of traffic engineering policies (in a pseudo policy language) that identifies FTP flows originating from the IP sub domain 10.10.10.0 / 24 and destined for the sub domain of 10.10.24.0 / 24. Once the flow is identified and marked as a class, there is a conditioning policy that instructs the queuing behavior to be associated.

1) *Policy Performer or Selector*: The Ontology-Based Performer is the component in charged of producing the traffic engineering policies that will be managed by external policy-based management system as a set of policies or service logic (SLO), in this way we aisle the traffic engineering policy management from service lifecycle management operations and then we can guarantee added value quality of services (QoS) dependent on the service management view that the network view can not offer directly by its interoperable lackness. The policy editor generates the policies for the network view as a set of events, related conditions and actions which are defined and put together to provide desired behaviour to be exhibited from a managed system at any view then is it up to the Ontology-

TABLE 1  
TRAFFIC ENGINEERING POLICIES

POL_1	SUBJECT QoSAdmin TARGET Routers_on_PATH_A ON IPv4PacketReceived IF IP source in 10.10.10.0/24 AND IP of type FTP AND IP destination in 10.10.24.0 / 24 THEN < MARK IP.ToS with AF41 >
POL_2	SUBJECT QoSAdmin TARGET Routers_on_PATH_A ON IPv4PacketReceived IF IP source in 10.10.10.0/24 AND IP of type FTP AND IP destination in 10.10.24.0 / 24 THEN < SHAPE IP to 512Kbps >
POL_3	SUBJECT QoSAdmin TARGET Routers_on_PATH_A ON IPv4PacketReceived IF IP.ToS equals AF41 THEN < CBWFQ weight is 5 >

based performer to build those policies that will provide the traffic engineering support from the network services.

2) *Ontology-Based Policy Interactor*: The ontology based policy interactor is the component that can semantically match and transform events at one view to events at different views between the same or different services systems. The reasoning process is performed for policy analysis and policy interactions to ensure that the policy events are syntactically and semantically compatible. Using formal semantics (ontologies), we make automated reasoning possible; ontologies facilitate the enhanced representation, exchange, integration and querying of data by annotating simple management data with formal semantics [14]. Defining an ontological model of a complex system from scratch is complicated and time consuming. The aim, therefore, is to leverage existing management models in defining ontologies; this also facilitates the definition and processing of semantic similarity between different concepts in the ontologies. We do this by generating a baseline system ontology from appropriate subsets of the DEN-ng information model.

3) *Policy Analyser*: The Policy Analyser takes the specified policies and transforms them into the PDP rule engine format as well as the configuration code format. Our baseline ontology enables this component to take advantage of automated reasoning in performing the policy transformation and policy conflict detection processes.

*Policy transformation* is the mapping between different representations of policy. Since the different representations of policies often require different concepts, semantic integration techniques are required in the policy transformation process. Automated reasoning is an integral part of semantic interoperability [15]; it is used for example to infer relationships that have not been explicitly identified.

*Policy Conflict Detection determines whether* a conflict between policies can occur. Conceptually, this can happen when two or more policies are activated simultaneously that are enforcing contradictory management operations on the system. Often, policies do not lend themselves to the detailed analysis needed to determine if their specific actions are in

conflict; hence, we use the system ontology to enable the policy analyser to reason over this information to detect occurrences of conflicting management operations. This step builds on work outlined in [11].

## B. Onto-CONTEXT Functional Description

This section provides a brief review of the Onto-CONTEXT framework with the objective of validate the use of PRIMO architecture in terms of a suitable solution for Policy-Based Context-Aware architectures. Onto-Context deals with the creation and modelling of services, including their context information, in order to provide service assurance and management functions. It does this by constructing a framework using autonomic elements [16] for the management and execution of services. The framework uses programmable networks [17][18] implemented using programmable elements to provide a suitable execution environment for the logic of the services [7].

The framework allows the use of appropriate APIs for deploying the services in mobile and IP domains. The context-policy model that we propose is flexible enough to accommodate the value of context information that needs to be evaluated, especially if the context changes [19]. The context-policy model is based on standards as much as possible and, moreover, the policy model scales with the network or applications that use it. This context-policy model is expected to be accessible from autonomic elements that can reside in heterogeneous networks and architectures. Storage and retrieval of this information is also important; this requirement is met by being defined and implemented using an object-oriented philosophy. Table 2 shows policy examples and the interaction with traffic engineering policies.

The *Policy Manager (PM)* has two main functions: 1) to govern the functionality of the system components, and 2) to check and incorporate new policies that arrive from a Policy Generation Engine. The specific fields required by these policies and its general structure are explained in the section "Policy Structure". Once a new policy or policies are received, this component parses it into Java and sends it to the Policy Conflict Check Component, which then checks for conflicts with other policies that are already loaded in the system. This provides the flexibility to a) reject the policy if it conflicts with an already deployed policy, b) replace an already deployed policy with this new policy (which of course involves locating and gracefully stopping policy execution of the previous instance), or c) note that there is an unresolved conflict and wait for a human administrator to resolve it. Once all the checks have been completed, the PM waits for the receipt of an event (or set of events) that triggers the evaluation of the condition clause of the set of policies that respond to that event.

The *Decision Making Component* evaluates each policy, performing any run-time variable substitution required (for instance, context information changes) specified in the policy condition, and determines if the condition requirements are satisfied or not. If the condition evaluates to true, the Decision Making Component notifies the PM, so that it can determine which actions to execute for which policies. (Note that the PM will in general evaluate a new or edited policy as

TABLE 2  
SERVICE MANAGEMENT POLICIES

Service Code Distribution Policies (CDistribution)	"If (customized service B code is received) <b>then</b> (configure distribution of service B code) & (optimise Storage Point selection parameters)"
Service Code Maintenance Policies (CMaintenance)	"If (new version of customised service B code is received) <b>then</b> (remove old code version of service B from Storage Points) & (distribute new service B code)" "If (customized service B code expiration date has been reached) <b>then</b> (deactivate execution polices for service B) & (remove code of service B from Storage Points)" "If (The invocation's number for service B is very high) <b>then</b> (distribute more service B code replicas to new Storage Points)"
Service Invocation Policies (SInvocation)	"If (invocation event AF41 is received) <b>then</b> (customised service B must be downloaded to a specific IP address) "
Service Execution Policies (SExecution)	"If (invocation event AF41 is received) <b>then</b> (customized service B must be executed)"
Service Assurance Policies (SAssurance)	"If (customized service B is running) <b>then</b> (configure assurance parameters for service B) & (configure local assurance variables)" "If (level=2)&(parameterAF41>X) <b>then</b> (Action M)" "If (level=2)&(parameterSHAPE>Y) <b>then</b> (Action N)" "If (level=2)&(parameterCBWFQ<Z) <b>then</b> (level=1)&(Action K)"

soon as it is received. If the policy is not validated, then it is refused and sent back to the Policy Generation Engine for correction or storage *as unfinished policy* in a separate area.)

The *Ontology Manager* manages the entire set of context information related to a specific service. This is accomplished in a distributed manner. The Onto-Manager updates context information that describes its environment. Thus, an Onto-Manager in charge of service customization will be responsible for updating all contextual user information, and an Onto-Manager governing network resources and services is responsible for updating all network layer information. Context information then will be available for use in triggering events that may affect one or more aspects of the system (e.g., the creation, deployment, and/or customization of the service). The ontology-Manager is an ordered event-condition-action driver, as it uses the service events to map to the DEN-ng approach [5][20] used by PRIMO; when the policy event is instantiated; instances are loaded with current data that provides a picture of the system status and the policies governing the system.

The *Ontology-Based Parser* parses the Information Model objects that represent the policies and other associated managed objects [5].

The *Code Distributor* is used to download the generated code to the appropriate code repositories and/or storage points; our system also allows generated code to be downloaded directly to entities that can reconfigure themselves using this mechanism). This installation process is a precondition for a service to be delivered. This is because the autonomic system functions by dynamically adapting the services and resources that they deliver, and one way of

providing new or different functionality is through self-configuration. In particular, our approach uses a variant of model driven architecture [21]to achieve this. Code repositories will be distributed throughout the network infrastructure. Therefore, the Distributor will need to keep track of the URIs where code was installed; this is done using a system registry.

The *Invocation Service Listener* captures any of the different types of triggers (e.g., an event, such as a user logging onto the network, or the change in state of a managed entity) that may cause this service to be activated. This component also is responsible for communicating these events to the autonomic elements that provide these services.

The *Service Assurance* module continuously monitors the quality of service using the *Monitoring* function; this acts as an evaluator of the conditions or variables from networks supporting the services. The *Performance Asserter* sends instructions to the Autonomic Elements to modify the service logic to ensure that the network services do not violate their service level agreements. The usual sequence of events would start with the detection of a trigger (perhaps caused by an explicit or implicit consumer request, or the notification of a fault in the network) by the Invocation Service Listener that detects a problem with the service; the service listener binds the entities shown in Figure 6 to the appropriate autonomic elements and informs the Code Execution Controller about the problem.

The *Code Execution Controller* is the functional entity that will download the code necessary to fix the service. The Code Execution Controller composes and forwards an activation request message to the appropriate programmable nodes in the service execution environment, which in turn retrieve code from the repository and execute it, starting the service provisioning phase. As soon as the service has been started, the Service Assurance module is used to keep track of the behaviour of the service.

The *Policy Conflict Check* component is responsible for maintaining the consistency of all policies introduced into the system. For example, each dialect of the policy language has its own vocabulary and grammar rules. The system will check a submitted policy against these rules to ensure that the policy is well-formed. Decisions about when policies must be enforced are closely linked with the policy's Condition Evaluators, which are coordinated by the Decision Making Component. For example, this component checks its list of business rules that must be enforced against the list of submitted policies, and notifies the administrator of any matches. It also provides valuable information for policy conflict detection and resolution.

Using Policies to manage a communication system requires more computer resources than systems that are not policy based. Figure 9 shows the CPU usage (%) and Memory usage (in MBytes) versus the number of policies associated to a given service. We can observe that the CPU usage decreases with the replication of the service invocation rate. This means that the CPU usage decreases when the number of services (users of policy) increases, while the Memory usage increases due to the number of services (policies used to support services) being used. This occurs in

all systems (even those without a policy-based paradigm). This feature is important when we try to design a scalable system with CPU limitations. In this scenario, the advantages in using a policy-based management system outweigh the benefit of requiring less computer resources.

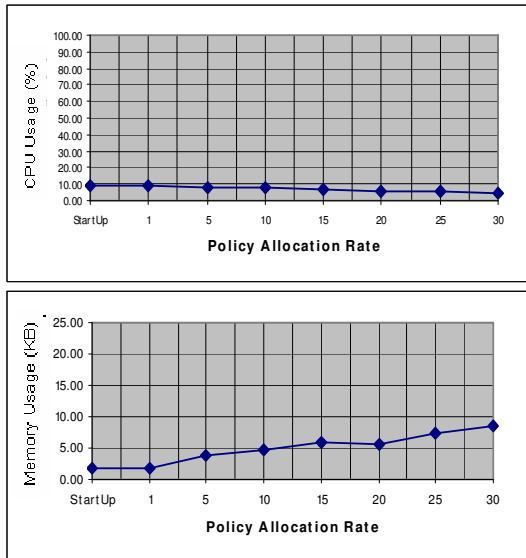


Figure 9. CPU-Memory usage vs. Policy Allocation Rate.

The performance study was made using a PC with the following characteristics: Processor: Intel Pentium 4 at 2.00 GHz; AT/At Compatible with 785 KB of RAM and supported by Microsoft Windows 2000 5.00.2195 with Service Pack 4; JRE 1.4.2\_06 from Sun. and the AppPerfect DevSuite 5.0.1 from AppPerfect Java Profiler [22]. Currently we are testing the performance without using ontologies and policies in the same system but further research work is going in this line and then have metrics for compare within both policy-driven and ontology-based policy-driven systems.

#### IV. SERVICE DEPICTED SCENARIO

As an example application of the Policy Refinement and the PRIMO architecture, we propose a set of scenarios in which the main factor is the user goal-oriented solutions and the non-intervention of specialized network managers. Once the service has been created, the main idea is that the user interacts with the system in a direct way, and this interaction generates pre-defined and/or customized services as required. The network operator now has a much simpler and reduced role – to function mainly as a simple authoring entity, and/or an entity that allows the operation. Only occasionally will the network operator have to intervene and define new policies to take direct control of the system, or make decisions when the system does not have enough information to process a decision on its own.

Next generation services need to be deployed on the fly (e.g., services that are pre-defined but that will be deployed when the user signs in to the system or after a certain period of time). Such services can be functions that execute when the user logs in, such as a video conference or the downloading of multimedia content that starts on demand. Furthermore, the services need to be assigned the appropriate

quality of service, depending on the type of user as well as user-centric context-sensitive attributes (e.g., this is a user that has just purchased a new service, or this is an established gold service user). These scenarios inherently require technological support based on policy that allows them to react in a specific manner based on the specific context.

Assume that a large quantity of users that subscribe to a wireless access service are looking for independence from Wi-Fi operators; we can use our approach to create such a service, called Sir-WALLACE (Service integrated for WiFi Access, ontoLogy-Assisted and Context without Errors). The application scenario, depicted in figure 8, requires traffic engineering algorithms, to satisfy the large demand for seamless mobility (such algorithms are composed into policies by the PRIMO architecture) and furthermore satisfy of interoperability between different technologies for the distribution of context information within the next generation services. Sir-WALLACE takes advantage of networks and user environment information and then provides a better, more advanced wireless access service to its users using traffic engineering algorithms managed by policies. The specific traffic engineering algorithms are based on user necessities (e.g., taken from their profiles) and mobility patterns (e.g., create routing algorithms based on the most frequently used access points of the user, PRIMO solve this necessity performing the policies for managing the algorithms). Sir-WALLACE is responsible for providing the QoS guarantees for a specific time period and service session; this facilitates its use among WiFi networks that use overlay networks to satisfy user demands. Sir-WALLACE upgrades the services based on the information defined in the management policies and its subsequent interpretation using the appropriate ontologies. The use of context enables the services that Sir-WALLACE provides to be better than conventional ones because it is using the context information to configure the appropriate service logic.

The user subscribes to the service (e.g., using a service setup web interface) and then, based on user profile information, the personalised services are generated and deployed among the programmable nodes covering the areas in which the user is moving. These nodes are connected to the WiFi Network nodes from the different operators; then the change of technology or access network is transparent for the user. The user information is used to know the location of the user, and provide updated context information for building an overlay network (or VPN). However, the overlay (or VPN) is not created until the context information triggers the service. Then, the overlay (or VPN) is created and stays active while the user is present in the WALLACE system.

When a user registers for this service, he enters:

- Personal information (name, address etc),
- Information about the different ways he can access the network, in order to connect to the network: MAC addresses for WLAN network cards, etc, and
- Service level, for quality of service purposes. A user can choose between service levels, which corresponds to different policies, such as: Local: related to a City; Region: related to a particular geographical area, which may

include nearby states or countries; and Global: related to many regions (i.e. Central and Eastern Europe). The system uses the context information for future deployment of the services and distributes the information to be stored in the knowledge databases.

To support management information interoperability, we propose to use components from the Context architecture [23] and extend towards Onto-Context and then provided a semantic-oriented solution using ontologies. We depicted a real next generation services scenario using the testbed of Figure 10. A precondition is that the Context platform is installed in the appropriate network edge nodes constituting a programmable overlay network, and that the user terminal is a mobile device that can access multiple wireless technologies, in particular IEEE 802.11b, IEEE 802.11a, and IEEE 802.11g. Specific events are forced in the testbed to trigger the evaluation of certain conditions and test the Invocation Service Listeners according to the corresponding variations of the context information in different views.

A variation in the traffic engineering policies or a network disconnection is always treated as a context variation. Similarly, a change of wireless access technology is viewed as a context change, which triggers a variation of the network status. These and other context changes are used to update functionality (e.g., connection speed, type of QoS, or alternative services that can be offered), and adjust appropriately the service authentication, billing, and other business issues.

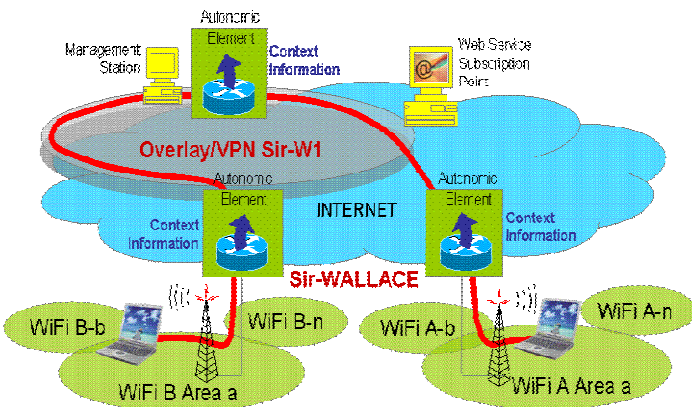


Figure 10. Sir-WALLACE Depicted Testbed Scenario.

## V. SERVICES & POLICY INTERACTION RELATED WORKS

Many applications that have or are currently being developed that require (or would benefit from) autonomic computing systems follow the premise of optimizing the support for user-oriented services using operational and/or business support systems (OSS and BSS).

The CONTEXT project [23] defines an XML Policy Model that supports the complete service life cycle. The policy model is extensible and contains parts defined as context information. This approach follows the business-oriented scope based on context information that the

networks require to operate. The service lifecycle is managed by a set of policies that contain such context information, and it is used as trigger events. However, this proposal lacks in using appropriate formalisms for sharing context information for supporting the reuse of context information contained in the policies.

Another approach dealing with Policy Based Service Management (PBSM) using representation frameworks is the TEQUILA project [24]. Its overall objective is to specify, implement and validate a set of service definition and traffic engineering tools to obtain quantitative end-to-end quality of service guarantees through planning and dynamic control of scalable and simple qualitative traffic management techniques within Internet services. This proposal works at a relatively low level; hence, its extensibility is limited.

Finally, an interesting proposal is the FOCAL architecture [25], which uses context-aware policy to control autonomic management operations and the functionality of different architectural components of the control loop. This enables the different control loop components to change the type of algorithm used, the type of function used, and even the type of data to use as a function of context. This is facilitated by enabling the detection of context changes to change the active policies that are being used at any given time. This work is focused (at least at present) on larger OSS/BSS issues; in particular, it does not discuss how semantic integration of information models with ontologies is done on a per-user, per-service basis. The advantage of the PRIMO framework is that only the local domains are required to semantically integrate management data (e.g., the Policy Interactor can deal exclusively with one or more Ontology Managers). This offers a generic policy integrator tool for ontology-based policy-driven service systems. Thus, this work can be integrated with FOCAL to provide support for multiple policy-driven, context-aware services.

## VI. CONCLUSIONS

The PRIMO approach is uniquely positioned towards supporting user-oriented services, since it is *contextually aware* of changes to the system, service, environment and/or user in form of events that change the state of the systems managed by policies. The PRIMO architecture provides a means for reacting to ontology-based context changes in a predictable and scalable manner through its Ontology-based policy model. Notably, this avoids requiring the use of skilled resources for simple, manually intensive operations, and has the added benefit of automatically providing additional information that aids the operator in understanding management data. The automation that PRIMO provides to policy-based management systems is critical to support the requirements of next generation services.

The PRIMO architecture inspiration becomes from the core idea to provide a mediator solution between multiple policy-based systems and ontology-based solutions. PRIMO act as an alternate solution in the process toward autonomic systems adaptation. We believe that if exist an architecture able to interact with ontology-based policy models, the

exchange of information and the policy-based management activity can be done transparently as result of an information interoperability process. Further research work is addressing this interaction using the semantic interactions and reasoning engines to support the architecture.

The ontology-based policy interaction approach for supporting service management operations emerges as a tool to solve some of the main autonomic systems requirements in the line of self-management services, and integration, and interoperability between systems. This framework approach has been conceived and partially implemented (tested separately by now), the idea is that any policy-based service management system can interact with PRIMO (using ontology-based mechanisms) and then to provide a genuine and innovative managing tool for next generation services and interacts with autonomic systems. Further activity is being developed to make more extensive use of our semantic integration using ontologies and DEN-ng information models; in particular, we will focus on expanding the types of semantic reasoning that can be performed.

We have proposed an architecture in which the main factor is the user goal-oriented solutions and the non-intervention of specialized network managers. Once the services have been created, the user interacts with the system in a direct way; this interaction generates pre-defined and/or customized services as required. The network operator now has a much simpler and reduced role – to function mainly as a simple authoring entity, and/or an entity that allows the operation. Only occasionally will the network operator have to intervene and define new policies to take direct control of the system, or make decisions when the system does not have enough information to process a decision on its own. Future activity will be centered in the integration of both PRIMO and ontology-based context-aware Platforms such as onto-CONTEXT architecture to exchange the event's information using ontology mappings and semantic similarities.

#### ACKNOWLEDGMENTS

This research activity is co-funded by both the Ministerio de Educación y Ciencia under the project TSI2005-06413 and the Science Foundation Ireland (SFI) under the Autonomic Management of Communications Networks and Services programme (grant no. 04/IN3I404C).

#### REFERENCES

[1] J. Strassner and J. Kephart, "Autonomic Networks and Systems: Theory and Practice", Network and Operations Management Symposium-NOMS 06 Tutorial, April 2006.

[2] G. Tomlinson, R. Chen, M. Hoffman, R. Penno, "A Model for Open Pluggable Edge Services", draft-tomlinson-opes-model-00.txt, <http://www.ietf-opes.org>

[3] G. Piccinelli, C. Stefanelli, M. Morciniec. "Policy-based Management for E-Services Delivery" HP-OVUA 2001.

[4] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing", IEEE Computer, January 2003. <http://research.ibm.com/autonomic/research/papers/>

[5] J. Strassner, "Policy Based Network Management", Morgan Kaufman, ISBN 1-55860-859-1

[6] J.M. Serrano, J. Serrat, J. Strassner, R. Carroll, "Policy-Based Management and Context Modelling Contributions for Supporting Autonomic Systems". IFIP/TC6 Autonomic Networking. France Télécom, Paris, France. 2006.

[7] J. Strassner, "Seamless Mobility – A Compelling Blend of Ubiquitous Computing and Autonomic Computing", in Dagstuhl Workshop on Autonomic Networking, Jan 2006.

[8] T. R. Gruber "Toward principles for the design of ontologies used for knowledge sharing". Presented at the Padua workshop on Formal Ontology, March 1993.

[9] T. R. Gruber "A translation approach to portable ontologies". Knowledge Acquisition, 5(2):199-220, 1993.

[10] R. Neches, R. Fikes, T. Finin, R. Patil. T. Senator W.R. Swartout "Enabling technology For Knowledge Sharing". AI Magazine, Vol.12, (3), 1991.

[11] S. Davy, B. Jennings, J. Strassner, "Policy Conflict Prevention via Model-driven Policy Refinement" in Proc 17th IFIP/IEEE Distributed Systems: Operations and Management (DSOM) pp209-220 2006.

[12] J. De Bruijn, D. Fensel, R. Lara, A. Polleres, "OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web"; Nov. 2004

[13] N. Damianou, A. Bandara, M. Sloman, E.C. Lupu "A Survey of Policy Specification Approaches", Department of Computing, Imperial College of Science Technology and Medicine, London, 2002.

[14] I. Horrocks, B. Parsia, P. Patel-Schneider and J. Hendler, "Semantic web architecture: Stack or two towers?," in Proc. Principles and Practice of Semantic Web Reasoning (PPSWR 2005), pp. 37-41, Sept 2005.

[15] A. Wong, P. Ray, N. Parameswaran, J. Strassner, "Ontology mapping for the interoperability problem in network management", Journal on Selected Areas in Communications, Oct. 2005, Vol. 23, Issue 10, page(s): 2058- 2068.

[16] J. Strassner, D. Raymer, "Implementing Next generation Services Using Policy-Based Management and Autonomic Computing Principles" IEEE/IFIP Network Operations & Management Symposium 2006, 3-7 April 2006 Vancouver, Canada.

[17] S.G. Denazis, A. Galis, "Open Programmable & Active Networks: A Synthesis Study" - IEEE IN 2001 Conference, Boston, USA, 6- 9 May 2001.

[18] Raz, and Y. Shavitt, "An Active Network Approach for Efficient Network Management", IWAN'99, July 1999, Berlin, Germany, LNCS 1653, pp. 220 –231.

[19] K. Henriksen. et al. "Modelling Context Information in Pervasive Computing System". Proceedings of Pervasive 2002, LNCS 2414, pp. 167-160, 2002.

[20] TMF, "The Shared Information and Data Model – Common Business Entity Definitions: Policy", GB922 Addendum 1-POL, July 2003.

[21] Please see <http://www.omg.org/mda>

[22] AppPerfect DevSuite 5.0.1-AppPerfect Java Profiler. <http://www.appperfect.com/>

[23] IST-CONTEXT project, Active Creation, Delivery and Management of Context-Aware Services. <http://context.upc.es>

[24] IST-TEQUILA Project. Traffic Engineering for Quality of service in the Internet, at Large Scale. <http://www.ist-tequila.org>

[25] J. Strassner, E. Lehtihet, N. Agoulmine, "FOCALE – A Novel Autonomic Computing Architecture", LAACS 2006.