Automatic Policy Refinement Using OWL-S and Semantic Infrastructure Information

Torsten Klie[1], Benjamin Ernst[2], and Lars Wolf[1]

[1] Technische Universität Braunschweig
Institute of Operating Systems and Computer Networks
{tklie,wolf}@ibr.cs.tu-bs.de
[2] Resco GmbH, Hamburg
benjamin.ernst@resco.de

Abstract. Network management requires a lot of manual work. Due to ongoing growth of the Internet, more self-management is needed in order to deal with the growing complexity. Autonomic communication systems address this issue. Often, they make use of policy-based management systems. Policy-based management is an approach to simplify management by the use of rules. Policy refinement, i.e. breaking down high-level policies to technical configurations is still a task that needs a significant amount of manual work. We propose SEMPR, an architecture that uses Web services and automatic Web services composition as a complementary technique to policy refinement in order to automate policy-based management. Semantic Web services, described in OWL-S, as well as policies make use of our simple network ontology NINO. We show the application of our work in a Home Area Network (HAN) environment and give an outlook on future research challenges.

1 Introduction

Current network management leads to many errors, misconfigurations, and high expenses because it requires a lot of manual work. We expect the Internet and the number of connected devices still to grow significantly. Thus, costs and complexity will rise, too. Automation in network management is needed to be able to deal with the increasing complexity. Furthermore, administrators who do not have to perform repetitive, simple (but error prone) tasks can dedicate more time to network planning and optimization.

The term "autonomic communications" or "autonomic networking" is often used to refer to networks that provide self-management features. "Autonomics" here means the application of the autonomic computing paradigm [1] to the network management domain. Several authors propose the use of policy-based network management (PBNM) to obtain the desired self-governance [2,3,4]. PBNM can be seen as a possibility to reduce the complexity of network management tasks by governing the behavior of a system by the use of rules (policies) without having to reconfigure network devices manually [5]. Policies can exist on different levels of abstraction, which are organized hierarchically [6]. Although the number of levels differ in the literature, one can generally distinguish between high-level policies (HLP) inspired by the business domain and policies

on a more technical level, low-level policies (LLP), which are related to concrete network devices and its features. Translating HLP into LLP is called policy refinement. Currently, it still requires a large amount of manual work, since a general solution for policy refinement is not available. Thus, skilled personnel with good knowledge both in the business domain and in the technical domain is required. Policy refinement is one of the biggest challenges in the policy-based management research.

Due to the shortcomings of the Simple Network Management (SNMP) Framework [7], the standard management framework of the IETF, new technologies for management have been proposed. A very popular example is Web services, a technology for software systems that are designed to support inter-operable interactions between machines using a communication network. The advantages and drawbacks of this technology for management purposes have been analyzed by several people [8,9]. Furthermore, OASIS has proposed Management Using Web Services (MUWS) [10] as part of their Web Service Distributed Management (WSDM) architecture, and the DMTF has proposed WS Management [11].

In this paper, we will show the use of automatic Web services composition as a complementary technique to policy refinement in order to further automate network management. The paper is structured as follows. First of all, we will present our policy-based architecture for autonomic communications. Then we will describe the basics about (automatic) Web service composition. In Section 4, we will present our approach called SEMPR. Then, a case study will show the application of our approach in a Home Area Network (HAN) environment. After that, we will discuss the related work on policy-refinement in Section 6. Finally, we will summarize the paper and discuss some future work in Section 7.

2 Policy-based Architecture for Autonomic Communications

In this section, we will describe our proposed architecture for autonomic communications (see Figure 1). It contains three parts: the layer with network devices, the policy engine that manages the devices autonomically, and the user interface with which the administrator can specify policies and services.

The network devices are located on the bottom layer of our architecture. Some of them will offer Web services for configuration, for monitoring, and/or for notifications. They can conform to a (proposed) standard such as WSDM, WS Management, or they can be proprietary. Gateways allow to access devices that do not support Web services based management interfaces. The services that are provided by the devices either directly or via gateways are called low-level services (LLS).

The policy engine is the core of our architecture. It contains the registry for management Web services, the repository where policies are stored and the component that makes policy decisions.

Network devices register their LLS in the registry, along with information about the structure and semantics of their LLS. A service composition engine allows for combining LLS to higher-level services (HLS) to perform aggregated tasks. For example, it is possible to compose the Web services for a certain management task (LLS) into a single HLS that performs the operation in an entire subnet. The registry can be implemented as

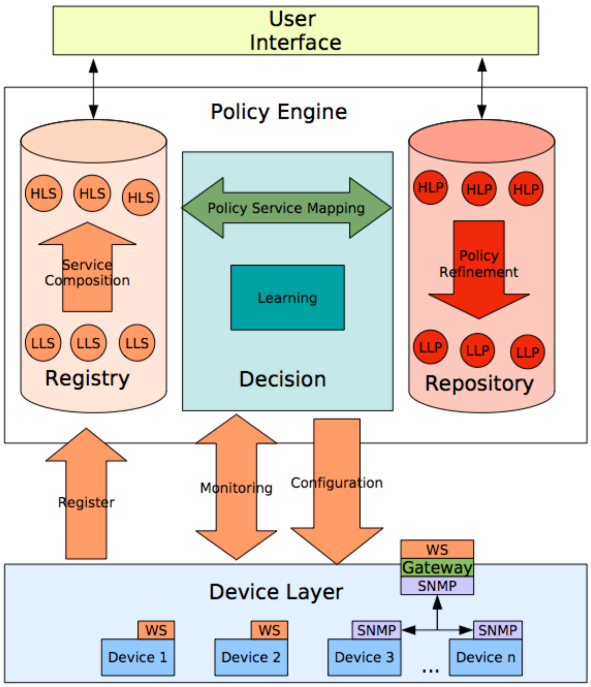

Fig. 1. A Policy-based Architecture for Autonomic Communications

a single file, a UDDI database, or peer-to-peer-based. The policy repository is the place where all the policies are located. It does not only store the policies, it also contains a policy refinement engine, that allows for breaking down HLP to LLP. In this paper, we will show how this can be achieved with Web service composition as described before.

The decision component uses the available services that are stored in the registry to monitor the devices. These monitoring information are used as input to the policy enforcement. All policies stored in the repository must be enforced. The resulting policy decisions may lead to changes in the device configurations. These changes are realized by calling the appropriate configuration Web services of the devices. A learning component not only evaluates the consequences of its own decisions, it also has to check the policies for possible conflicts. However, we do not address these two issues of the learning component in this paper and leave them for future reasearch.

With the user interface, which is located on top of the system, administrators can enter, remove, or change policies in the repository. A graphical user interface will assist them. Usually, administrators will specify HLP, which will be broken down to LLP in the repository. Moreover, in addition to the services composed within the refinement process, administrators can add and compose services manually.

3 Web Services Composition

One of the most important advantages of the machine-to-machine interaction support of Web services is the composability. Several services can be combined in order to perform an aggregated task. This composition is rather easy to implement.

The composition of Web services consists of two steps: synthesis, i.e. creating the composed service by selecting services that shall participate, and orchestration, i.e. the execution of the composed service. Composition methods can be classified by the way the synthesis is performed: manually, template-based (such as in the popular Business Process Execution Language for Web Services (BPEL4WS) [12] approach), or automatically. Of course, having the vision of creating autonomic systems in mind, the automatic approaches are the most interesting to look at. In our previous work [13], we analyzed different Web service composition approaches, and proposed the use of OWL Services (OWL-S) [14]. Other work also argues for the use of OWL-S in network management [15,16,17].

OWL-S is a commonly known ontology for semantic description of Web services and Web service compositions. The description is subdivided into three components: ServiceProfile, ServiceModel, and ServiceGrounding. The ServiceProfile describes what a service does using functional properties (IOPE: input parameters, output parameters, preconditions, and effects) and non-functional properties (e.g. `serviceName`, `textDescription`). Especially the functional properties are used to publish and find OWL-S Web services. Comparing the profiles of available services to the profile of a given task or process is called matchmaking. In the ServiceModel, a Web service is mapped to one or more processes. The IOs of processes can be defined using OWL classes or W3C XML Schema types, whereas the PEs can be modeled as logic formulas. Processes can be AtomicProcesses, CompositeProcesses or SimpleProcesses (processes that do not have a grounding and cannot be executed). All processes (except SimpleProcesses) need a ServiceGrounding that binds them to concrete specifications and messages. The grounding makes use of the Web Service Description Language (WSDL).

For the service composition using OWL-S, the IOPEs of the available Web services are described using OWL-S. Inputs and outputs are defined by a class of an OWL ontology. Using this description it is possible to combine different services in order to obtain a result not achievable by a single individual service. Figure 2 shows a simple composition based on inputs and outputs. Moreover, it is possible to use preconditions and effects to find the services that can be combined.

4 SEMPR and NINO

In this section, we will describe our proposed architecture for automatic policy refinement (SEMPR: **Sem**antic **P**olicy **R**efinement). We will first outline our apporach briefly and describe our model of the infrastructure (NINO: **N**etwork **In**frastructure **O**ntology). Then, we will present the architecture details of SEMPR. After that, we will describe the refinement approach in detail. Furthermore, we will provide some information about the implementation.

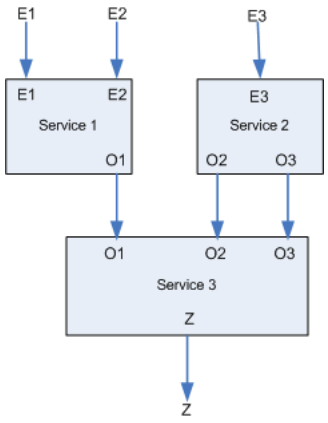

Fig. 2. Service composition with OWL-S (using input and output)

4.1 Policy Refinement with Web Service Composition

Our approach is based on semantic Web services in OWL-S. We use service composition to combine different available services in order to obtain a desired result. The idea is to create OWL-S services that can perform monitoring or configuration tasks needed by a policy. In the case of HLP, the OWL-S services will be created as a composed service, using several LLS. In order to find the LLS that have to be combined, we need a description of the infrastructure and the available Web services (usually LLS). Therefore, we intruduce NINO. The SEMPR engine, the core component of the SEMPR architecture, is responsible for identifying the needed results by extracting them from the policies, for finding relevant services, and for executing the composed services.

4.2 NINO

In order to describe the infrastructure and the Web services that are used for management, we needed an information model in form of an OWL ontology. There are several information models available, such as DEN-ng [18] and CIM [19]. However, these information models are very large and would need to be converted into OWL. Such a conversion (as proposed in [20]) would be a research project on its own. We wanted to concentrate on the use of the ontology for automatic Web services composition. Therefore, we created our own simple ontology called NINO.

NINO contains two ontologies that are used as repositories: a network repository ontology (`NetworkRepository.owl`) with information about the network infrastructure and a policy repository ontology (`PolicyRepository.owl`) with the policies. These repositories do not contail any definitions, but they contain individuals of the NINO classes. The definitions of the NINO classes are given in the base ontologies, a collection of different ontologies that is extensible. Currently, there is a network ontology (see Figure 3), a policy ontology, and an ontology containing the information needed for the home control services needed for our case study (see Section 5).

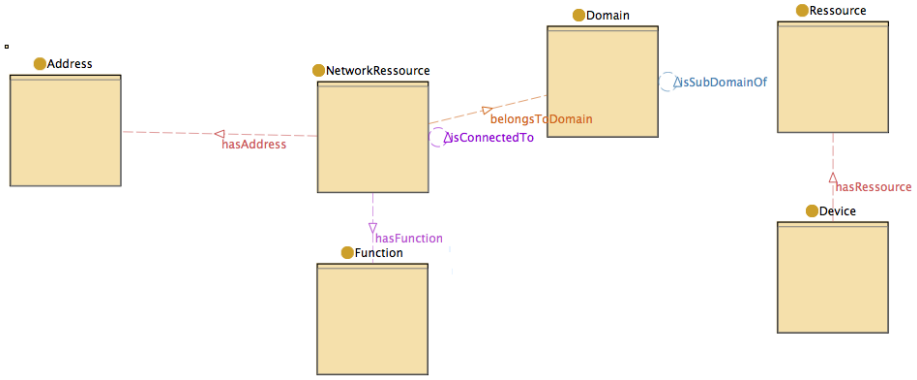

Fig. 3. Excerpt of the NINO network ontology

The network ontology provides basic classes for modelling the network infrastructure. The base class is a `Device` in the network which consists of one ore more `Ressources`[3]. A spezialization of `Ressource` is `NetworkRessource`, which represents e.g. a network interface. It can have `Addresses` (IP, MAC), belong to `Domains`, and have different `Functions`. The `Function` concept is comparable to the concept of Roles in DEN-ng. Different functionality such as routing or packet filtering is modelled as a subclass of `Function`. The policy ontology provides the classes needed to specify policies (such as `Action`, `Condition`, etc.).

4.3 SEMPR

The SEMPR engine is the core component of the SEMPR architecture (see Figure 4). It performs automatic policy refinement by composing and executing Web services matching the needs of the policies that are being refined. In the following, we will describe the different parts of the engine.

- Refinement Engine: gets the policies using the NINO API, performs the refinement for each policy with the matchmaker client, and tells the OWL-S engine to execute the composite service.
- NINO API: the link between SEMPR engine and NINO ontologies. It provides functions to read information from NINO (policies and network device descriptions).
- Matchmaker Client: supports the refinement engine by providing OWL-S services matching the given IOPEs.
- OWL-S Engine: executes the composite service by executing each atomic Web service sequentially.

[3] The distinction between `Device` and `Ressource` is necessary to model cases in that a device contains for example several network interfaces.

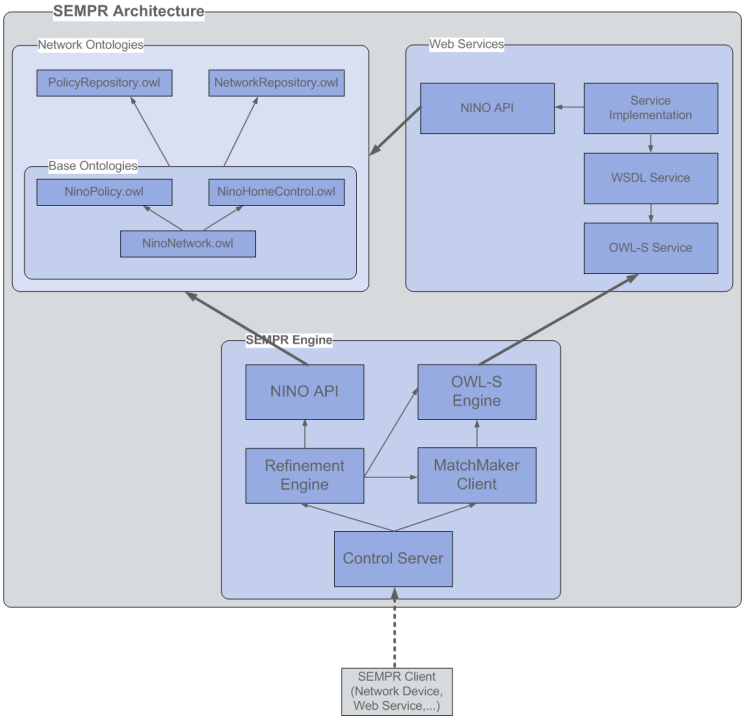

Fig. 4. SEMPR architecture

– Control Server: a little server that controls the SEMPR engine. It initiates a refinement process, adds new devices and policies to NINO, or registers or removes Web services at the matchmaker.

Web services provide the management functionality of network devices. They have to be created in three steps. First, the interface of the Web service has to be described with WSDL. Then, the semantic of the service has to be described with OWL-S. Finally, the Web service has to be implemented. These three steps require manual work.

SEMPR covers most of the tasks of the policy engine in described in Section 2. It performs the policy refinement by mapping policies to OWL-S service descriptions which are then composed on demand. The policy decision and the learning component are not part of SEMPR.

4.4 Policy Refinement Algorithm

An automatic policy refinement requires to break down one or more Policies into concrete device configurations without manual user interaction. SEMPR allows this by using automatic semantic Web service composition. The basic steps that are performed are described in the following.

1. Extract conditions and actions from the policy that shall be refined.
2. Extract input parameters and other specific information from the conditions and actions and store them in an input list.
3. Generate the target of the refinement from the conditions and actions. It is necessary to know the target because it defines the output that the composite Web service shall have. These information are stored in an output list.
4. The matchmaker looks for services providing the output specified in the output list.
5. If there is a service for which there are all input parameters in the input list, the service could already be executed. It will get a mark (equal to the number of the current iteration of the loop). This mark will be used later to determine the execution order of the services.
6. Output parameters for which a matching service has been found are eliminated from the output list and put into the input list. These output parameters could already be generated and used by another service as input.
7. The input parameters of the found services are added to the output list, if they are not already in the input list.
8. If there are still elements in the output list, the loop starting at step 4 will be executed again. Otherwise, the refinement terminates.

The list with the services that have to be executed and the inputs of the policies are passed to the OWL-S engine, which will invoke the Web services in the predefined order. The outputs of a Web service call are passed to the next Web services as input. An example can be found in Section 5.2. The flow chart in Figure 5 shows the steps of the refinement algorithm.

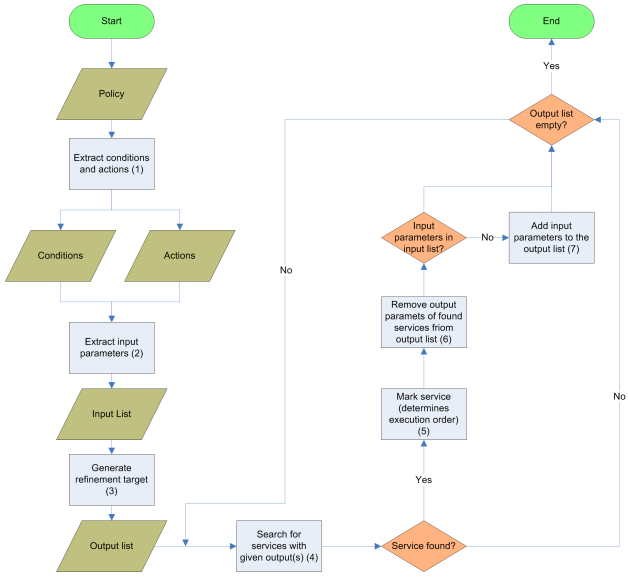

Fig. 5. Policy Refinement Algorithm (Flow Chart)

4.5 Implementation

NINO and SEMPR have been implemented in Java (Version 5). An Apache Tomcat server (Version 5.5) [21] has been used as application server for the Web services, as well as a Web server to store the OWL ontologies. The Apache Axis2 engine (RC 1.1) [22], an execution environment for Web services based on the SOAP protocol, has been installed as a servlet under Tomcat. Jena (Version 2.4) [23], an open source framework for the development of Semantic Web applications from the HP Labs Semantic Web Programme has been used to work with OWL files. After the refinement process, Web services are invoked by the OWL-S API (nightly build from November 1st, 2006) [24]. The SEMPR engine communicates with the Web services via SOAP.

5 Case Study: Home Area Networks

Home Area Networks (HANs) are networks located at the home environment of the user. They interconnect all kinds of different devices: "classic" network devices (such as computer, routers or printers), multimedia devices (such as streaming clients, VoIP telephones or set top boxes), and facilities (such as refrigerators, lights, blinds, etc.). The integration of the last category of devices is currently more a future scenario than reality. However, there are trends towards a full integration of such devices in houses, so we expect this to change in the future.

Especially in HANs, the ordinary user should not be bothered with low-level administration tasks. First of all, he/she is usually not qualified enough. Second, hiring a network administrator for these tasks would be too expensive. Finally, if the user experience is not smooth and comfortable, users will just not invest into HANs. Therefore, it would be huge advantage if HANs were manageable by HLPs.

5.1 Scenario

Our example HAN consists of four routers, to which several end devices (some with the option to connect more controlled devices) are connected (see Figure 6).

All devices in the example are represented by an OWL individual. Furthermore, all its functionality, dependencies and connections are modeled. Figure 7 shows the OWL description of router R2. In addition to network devices and their functionality, the ontology contains other individuals with respect to the infrastructure. Rooms, for example, or other not directly network-compatible devices such as lights are included as well. Since editing OWL with an ordinary text editor is error prone, we used the knowledge modeling tool Protégé.

Policies have to be described in the same way as network resources. Each policy contains a condition and an action. A condition can be either a single condition or several conditions combined with `AND` or `OR`. Actions can be combined in the same way, but without logical connectors. A sample, rather simple policy is shown in Figure 8. This policy says that the light in `RoomHall` shall switch to the state `LightStatus6`, if the sensor in the same room (`SensorType4`) changes its state to `SensorStatus3`.

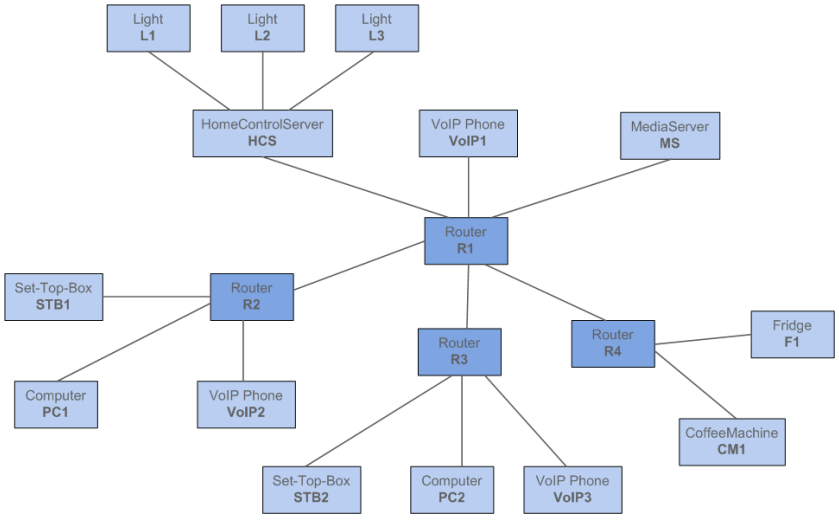

Fig. 6. The Home Area Network (HAN) example

5.2 Policy Refinement Process

In the following, we will show the refinement process step-by-step. To keep the example readable, we have chosen a very simple policy (the one presented in the previous section). The different policy conditions, actions, and Web services involved in the refinement process are described in Table 1. More detailed information about the implemented services can be found in [25].

Condition	**Inputs**	**Outputs**	**Description**
`SensorConditionByRoom`	`Room`, `SensorStatus`		fulfilled if the sensor in the given room switches to the given state
Action	**Inputs**	**Outputs**	**Description**
`LightActionByRoom`	`Room`, `LightStatus`		switches the light in the given room to the given state
Service	**Inputs**	**Outputs**	**Description**
`getLight`	`Room`	`Light`	determines a light in the given room
`getSensor`	`Room`,`SensorType`	`Sensor`	determines a sensor of the given type in the given room
`getHCS`	`Light`	`NetworkResource`	determines the network resource that controls the given light
`switchLightBySensor`	`Light`, `LStat`, `SStat`, `Sensor` `Network Resource`		creates a configuration for the given network resource that connects the given light and the given sensor

Table 1. Conditions, actions, and services used in the example

```
<p1:Device rdf:ID="Router_2">
<p1:hasRessource rdf:resource="#NetworkRessource_R2"/>
<p1:Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
Router2
</p1:Name>
</p1:Device>

<p1:NetworkRessource rdf:ID="NetworkRessource_R2">
  <p1:isConnectedTo>
    <p1:NetworkRessource rdf:ID="NetworkRessource_PC1">
       </p1:isConnectedTo>
    <p1:isConnectedTo>
    <p1:NetworkRessource rdf:ID="NetworkRessource_STB1">
  </p1:isConnectedTo>
  <p1:isConnectedTo>
    <p1:NetworkRessource rdf:ID="NetworkRessource_VoIP2">
  </p1:isConnectedTo>
  <p1:Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    NR_R2
  </p1:Name>
  <p1:isConnectedTo rdf:resource="#NetworkRessource_R1"/>
  <p1:belongsToDomain rdf:resource="#HomeNetwork_Domain"/>
  <p1:hasFunction rdf:resource="#TrafficPriorityFunction_1"/>
  <p1:hasAddress>
    <p1:IPv4Address rdf:ID="IPv4Address_23">
      <p1:AdressValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        192.168.44.20
      </p1:AdressValue>
    </p1:IPv4Address>
  </p1:hasAddress>
</p1:NetworkRessource>
```

Fig. 7. OWL description of router R2

```
<j.0:Policy rdf:ID="Policy_4">
  <j.0:Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    VacPol1
  </j.0:Name>
  <j.0:hasAction>
    <j.0:LightActionByRoom rdf:ID="LightActionByRoom_6">
      <j.0:Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        LighAction2
      </j.0:Name>
      <j.0:hasLightStatus rdf:resource="#LightStatus_6"/>
      <j.0:hasRoom rdf:resource="#Room_Hall"/>
    </j.0:LightActionByRoom>
  </j.0:hasAction>
  <j.0:hasCondition>
    <j.0:SensorConditionByRoom rdf:ID="SensorConditionByRoom_15">
      <j.0:hasRoom rdf:resource="#Room_Hall"/>
      <j.0:hasSensorType rdf:resource="#SensorType_4"/>
      <j.0:Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        SCondByRoom 1
      </j.0:Name>
      <j.0:hasSensorStatus rdf:resource="#SensorStatus_3"/>
    </j.0:SensorConditionByRoom>
  </j.0:hasCondition>
</j.0:Policy>
```

Fig. 8. A simple policy in OWL

– **Preparation:** Extracting inputs and outputs
  Inputs: `SensorStatus`, `Room`, `SensorType`, `LightStatus`
  Outputs: `SenosrConditionByRoom`, `LightActionByRoom`
  Services:

– **0th iteration:** Looking for services that can produce the desired output. The found service is `switchLightBySensorService`. Its input parameters are added to the output list if they have not already been on the input list.
  Inputs: `SensorStatus`, `Room`, `SensorType`, `LightStatus`
  Outputs: `SensorConditionByRoom`, `LightActionByRoom`, `NetworkResource`, `Light`, `Sensor`
  Services: `switchLightBySensor`

– **1st iteration:** Looking for services that can produce the desired output. Since new parameters have been added to the output list during the last iteration, more services are found. All services are checked whether they can be invoked already. `getLightService` and `getSensorService` could be invoked now, because their needed input (`Room`) is already available in the input list. Therefore, they are marked with the number of the current iteration (1) and their output is added to the input list and removed from the output list.
  Inputs: `SensorStatus`, `Room`, `SensorType`, `LightStatus`, `Sensor`, `Light`
  Outputs: `SensorConditionByRoom`, `LightActionByRoom`, `NetworkResource`
  Services: `getLightService`(1), `getHCService`, `getSensorService`(1), `switchLightBySensor`

– **2nd iteration:** No new service can be found, because no new parameters have been added to the output list. But due to the new parameters on the input list the service `getHCSService` could be invoked now. Its parameter `NetworkResource` is moved to the input list.
  Inputs: `SensorStatus`, `Room`, `SensorType`, `LightStatus`, `Sensor`, `Light`, `NetworkResource`
  Outputs: `SensorConditionByRoom`, `LightActionByRoom`
  Services: `getLightService`(1), `getHCSService`(2), `getSensorService`(1), `switchLightBySensor`

– **3rd iteration:** With the new input, the last service (`switchLightBySensor`) can also be invoked. The output list is now empty and the refinement terminates.
  Inputs: `SensorStatus`, `Room`, `SensorType`, `LightStatus`, `Sensor`, `Light`, `NetworkResource` `SensorConditionByRoom`, `LightActionByRoom`
  Outputs:
  Services: `getLightService`(1), `getHCSService`(2), `getSensorService`(1), `switchLightBySensor`(3)

– **Web service invocation:** The Web services are now called in the order determined by the marks.

5.3 Performance Evaluation

The run-time performance of the policy refinement process is not critical since it is only invoked from time to time. However, to estimate the performance in large systems and networks we looked at three steps: service registration at the matchmaker, invocation of a single OWL-S service, and a complete refinement of a single policy. The time needed to find the relevant services and determine the execution order can be neglected, since we always measured 0 ms.

In order to perform tests, different policies have been created. SEMPR has been installed on a Pentium 4 PC with 3.2 GHz CPU and 2 GiB RAM. Debian Linux (Sarge) was used as the operation system. The following measured results are based on refinements done with 35 different policies.

Registering an OWL-S Web service at the matchmaker consumes most of the time (12.985 s on average). The reason for this large amount of time is that the OWL-S descriptions contain a large number of references to OWL documents, which have to be read and are often located at different places. Service invocation takes 2.461 s on average. The differences for services with many parameters are very small. The overall time needed for the refinement depends on the number of Web services that have to be executed. In our tests, 3 - 4 Web services had to be called and the average time needed was 8.880 s.

While using a larger setting for scalability testing, we noticed a memory leak in the Axis2 engine. Thus, we had to restart our server after the invocation of 35 Web services, because there was not enough memory left to continue. This is a known bug in Axis2 that will hopefully be fixed soon.

6 Related Work on Policy Refinement

Bandara et al. [26] present a semi-automated approach for policy refinement, which is based on breaking down goals. The process works in two steps. First, high-level goals are refined into concrete goals (=system requirements). Then, system requirements are mapped to specific modules and/or operations available in the system. Goals are represented as rules in temporal logic and decomposed with elaboration patterns (with KAOS technology). Bandara et al. use UML state charts to describe the system. The relationship between the system description and the goals are called a strategy, which can either be implemented as a policy or as a system functionality. This decision cannot be automated, but Bandara et al. provide refinement patterns. To further automate the refinement process, they propose the use of Event Calculus and abductive reasoning.

Rubio-Loyola et al. [27] present an approach that makes use of linear temporal logic and model checking as an analysis tool. The presented framework is structured according to the steps that have to be performed: goal management (goal elaboration and goal selection) and policy refinement. The refinement process consists of the following steps: requirements formulation with linear temporal logic formulas, get system behavior from system traces with SPIN model checker, apply translation primitives to generate policies, and encode deployable policies in Ponder.

According to Strassner et al. [28], a policy may exist on different abstraction levels, representing the view of different constituencies. For example, a Quality of Service

(QoS) policy representing a Service Level Agreement (SLA) looks different from the viewpoint of a sales person and a network engineer. However, both views represent the same policy. Therefore, Strassner et al. use the term "Policy Continuum" to refer to the set of different views. DEN-ng includes this set of views. It is not a single information model but offers an information model (with its own grammar and vocabulary) for each level. The policy continuum can be seen as a set of model mappings between the levels, allowing the translation of policies of different levels into one another. Ontologies that contain the necessary semantic information and knowledge of the different domains are needed to enable this translation based on reasoning.

Guerrero et al. [29] present a generic ontology-based policy refinement approach, which uses the Semantic Web Rule Language (SWRL). They propose a semantic manager that uses policies. They do not only address policy refinement, they also want to enable real interoperability which includes a backward mapping from LLPs to HLPs. Policies are represented with SWRL on each level. OWL relationships specify the relationship between high-level ontologies and low-level ontologies. Translation rules in SWRL can be used to allow the data interchange between the different levels.

Beigi et al. [30] propose transformation approach based on case base reasoning. They use a case database containing the history of the system behavior to provide an experimental basis for their transformation approach. The data in the case database is processed with mathematical analysis tools.

Casassa Mont, Baldwin, and Goh [31] present a template-based approach, which is related to a security application. They distinguish between two types of users: experts who know well the application domain and consultants, who are specialists in the business domain. The experts create policy refinement templates which the consultants can use to create and refine policies with the help of a graphical tool.

Although the number of existing approaches is quite high, they all have disadvantages that get in the way of its usage in an autonomic communication environment. They are either not fully automated ([26], [27], [31]), require a very high modelling effort ([28]), rely on historic data only ([30]), or a tailored towards a specific application such as security ([31]). SEMPR is a generic and fully automated approach. Its required modelling effort is moderate (a NINO model is needed, but no additional mappings are required). The downside is that it relies entirely on a Web serivce infrastructure in the form of Web service based management interfaces at the devices.

7 Conclusion

In this paper, we have shown, how automatic Web service composition based on OWL-S Web services can be used for refining policies. The policies and the network infrastructure, i.e. the devices and their Web services based management interfaces, are described using OWL and OWL-S, respectively. For evaluation purposes, we have presented a simple ontology called NINO. In the proposed SEMPR architecture, the conditions and actions of a policies written with NINO are extracted and passed to the matchmaking engine, which tries to find the relevant services and combines them to a composed Web services that is then executed.

Without the obstacle of the policy refinement problem, policy-based management systems will be much easier to deploy. With powerful policy-based management-systems available, autonomic communication will move from a vision closer to reality. However, autonomic communication is not only about policies. Several issues still have to be resolved. One of the most important ones is an information model and/or ontology that covers all aspects of network management.

The presented approach assumes that all devices are managed via Web services. Thus, all managed objects must provide such management interfaces. As of today, this is clearly not the case, but there are trends towards Web services in management (e.g. OASIS WSDM, WS Management) so this is not an unrealistic scenario for the future. On the other hand, for devices not supporting Web service, gateways can be used.

We are working on the integration of SEMPR in our autonomic communications architecture. Currently, we are investigating more lightweight Web service infrastructures (more lightweight than Apache Axis2) in order to use Web services on devices with rather limited resources. Moreover, we are extending the approach to take preconditions and effects of OWL-S Web services into account, and to support policies with events. Furthermore, we will address policy conflicts and evaluation of policy decisions. To implement a large-scale testbed of our system, we need also an extended ontology capable of describing all aspects of the network.

References

1. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer Magazine **36**(1) (January 2003) 41–50
2. Davy, S., Barrett, K., Balasubramaniam, S., van der Meer, S., Jennings, B., Strassner, J.: Policy-Based Architecture to Enable Autonomic Communications – A Position Paper. In: Proc. IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, USA (January 2006)
3. Agrawal, D., Lee, K.W., Lobo, J.: Policy-based Management of Networked Computing Systems. IEEE Communications Magazine **43**(10) (October 2005) 69 – 75
4. Klie, T., Wolf, L.: Autonomic Policy-based Management using Web Services. In: Proc. 2nd CoNext Conference, Lisbon, Portugal (December 2006)
5. Choudhary, A.R.: Policy-Based Network Management. Bell Labs Technical Journal **9**(1) (January 2004) 19–29
6. Moffett, J.D., Sloman, M.S.: Policy Hierarchies for Distributed Systems Management. IEEE Jounal on Selected Areas in Communications **11**(9) (December 1993) 1404–1414
7. Schönwälder, J., Pras, A., Martin-Flatin, J.P.: On the Future of Internet Management Technologies. IEEE Communications Magazine **41**(10) (October 2003)
8. Pras, A., Drevers, T., van de Meent, R., Quartel, D.: Comparing the Performance of SNMP and Web Services-Based Management. eTransactions on Network and Service Management **1**(2) (December 2004)
9. Pavlou, G., Flegkas, P., Gouveris, S., Liotta, A.: On Management Technologies and the Potential of Web Services. IEEE Communications Magazine **42**(7) (July 2004)
10. Vambenepe, W.: Web Services Distributed Management: Management Using 3 Web Services (MUWS 1.0) Part 14 . OASIS Standard wsdm-muws-part1-1.0 (March 2005)
11. McCollum, R., et al.: Web Services for Management. WS-Management (June 2005)
12. Thatte, S., et al.: Business Process Execution Language for Web Services – Version 1.1. OASIS Standard BPELv11-May052003 (May 2003)

13. Klie, T., Gebhard, F., Fischer, S.: Towards Automatic Composition of Network Management Web Services. In: Proc. of 10th IFIP/IEEE International Symposium on Integrated Management (IM), Munich, Germany (May 2007)
14. Martin, D., et al.: OWL-S: Semantic Markup for Web Services. DAML White Paper Release 1.1 (November 2004)
15. Keeney, J., Carey, K., Lewis, D., O'Sullivan, D., Wade, V.: Ontology-based Semantics for Composable Autonomic Elements. In: Proc. of Workshop on AI in AutoComm at 19th Int'l Joint Conference on Artificial Intelligence, Edinburgh, Scotland (July 2005)
16. de Vergara, J.L., Villagrá, V., Berrocal, J.: Application of OWL-S to Define Management Interfaces Based on Web Services. In: Proc. 8th IFIP/IEEE Int'l Conference on Management of Multimedia Networks and Services (MMNS 2005), Barcelona, Spain (September 2005)
17. Fuentes, J.M., de Vergara, J.E.L., Castells, P.: An Ontology-Based Approach to the Description and Execution of Composite Network Management Processes for Network Monitoring. In: Proc. 17th IFIP/IEEE International Workshop on Distributed Systems, Operations and Management (DSOM), Dublin, Ireland (October 2006) 86–97
18. Strassner, J.: DEN-ng: Achieving Business-driven Network Management. In: Proc. 2002 IEEE/IFIP Network Operations and Management Symposium, Florence, Italy (April 2002)
19. DMTF: Common Information Model (CIM) Infrastructure Specification. DSP 0004, DMTF (October 2005) Version 2.3 Final.
20. Quirolgico, S., Assis, P., Westerinen, A., Baskey, M., Stokes, E.: Toward a Formal Common Information Model Ontology. In: Proc. 5th International Conference on Web Information Systems Engineering (WISE), Brisbane, Australia (November 2004) 11–21
21. Apache Software Foundation: Apache Tomcat. WWW Page (2007) http://tomcat.apache.org/.
22. Apache Software Foundation: Apache Axis2/Java – Next Generation Web Services. WWW Page (April 2007) http://ws.apache.org/axis2/index.html.
23. Hewlett-Packard Development Company: Jena - A Semantic Web Framework for Java. WWW Page (February 2007) http://jena.sourceforge.net.
24. Sirin, E.: OWL-S API. WWW Page (2004) http://www.mindswap.org/2004/owl-s/api/.
25. Ernst, B.: Automatisches Policy-Refinement mit Hilfe von semantischen Infrastruktur-Informationen. Diploma thesis, TU Braunschweig (January 2007)
26. Bandara, A.K., Lupu, E.C., Russo, A., Dulay, N., Sloman, M., Flegkas, P., Charalambides, M., Pavlou, G.: Policy Refinement for IP Differentiated Services Quality of Service Management. IEEE eTransactions on Network and Service Management **3**(2) (2006) 2–13
27. Rubio-Loyola, J., Serrat, J., Charalambides, M., Flegkas, P., Pavlou, G.: A Functional Solution for Goal-oriented Policy Refinement. In: Proc. 7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY), London, Canada (June 2006)
28. van der Meer, S., Davy, A., Davy, S., Carroll, R., Jennings, B., Strassner, J.: Autonomic Networking: Prototype Implementation of the Policy Continuum. In: Proc. of 1st IEEE Int'l Workshop on Broadband Convergence Networks (BcN), Vancouver, Canada (April 2006)
29. Guerrero, A., Villagrá, V.A., de Vergara, J.E.L., Sánchez-Macián, A., Berrocal, J.: Ontology-based Policy Refinement Using SWRL Rules for Management Information Definitions in OWL. In: Proc. 17th IFIP/IEEE International Workshop on Distributed Systems, Operations and Management (DSOM), Dublin, Ireland (October 2006) 227–232
30. Beigi, M.S., Calo, S., Verma, D.: Policy Transformation Techniques in Policy-based Systems Management. In: Proc. 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY), New York, USA (June 2004)
31. Mont, M.C., Baldwin, A., Goh, C.: POWER Prototype: Towards Integrated Policy-Based Management. In: Proc. 6th IFIP/IEEE Network Operations and Management Symposium (NOMS), Honululu, USA, Hewlett Packard Laboratories Bristol (April 2000) 789 – 802