Model-based Management of Embedded Service Systems – An Applied Approach

Stefan Illner, Heiko Krumm,
Ingo Lück, Andre Pohl
University of Dortmund, Germany
{illner, krumm, pohl}@ls4.cs.uni-dortmund.de
ingo.lueck@materna.de

Andreas Bobek, Hendrik Bohn,
Frank Golatowski
University of Rostock, Germany
{andreas.bobek, hendrik.bohn,
frank.golatowski}@uni-rostock.de

Abstract

The automatic integration of devices into dynamic, automatically configured networks alone does not take advantage of the entire potential of Service Oriented Architectures (SOA). Using service management, independent services can be directed to perform meta tasks in a SOA network.
In this paper we describe and evaluate the service management tool MoBaSeC*, which consists of two parts: at system runtime, management services are running in order to enforce management policies. The second part of the tool is a graphical model editor which supports the user in setting up the desired management policies easily.*
This research is part of the European SIRENA (Service Infrastructure for Real-time Embedded Networked Applications). The use of MoBaSeC *was evaluated by the project in automotive application and the results are summarized here.*

1 Introduction

The *Service Oriented Architecture (SOA)* approach provides a standardized view to software components in a network. These components – called *Services* – are described in a standardized format (*Service Description*) which is used to integrate, announce, discover and use the functionality they offer. Furthermore services can subscribe to events generated by other services to get informed about state changes. The implementation of the services is hidden from the service users, which only make use of the provided functionality using a well defined interface.

This research is part of the *SIRENA* project [1]. SIRENA was aimed at heterogeneous embedded devices interacting inside and between four different domains (automotive, telecommunication, industrial and home automation). Several SOAs were evaluated such as *Universal Plug and Play (UPnP)*, *Web services* or the *Open Services Gateway Initiative (OSGi)* for example. Due to interoperability and platform independence it was decided to use UPnP and the *Devices Profile for Web Services (DPWS)*.

UPnP is a simple, easy-to-use SOA for small networks [2]. It supports ad-hoc networking of devices and interaction of services by defining their announcement, discovery and usage. The UPnP specification defines a progamming language and platform independent technology as only protocols and interfaces are specified. It divides the device life cycle into six phases: *Addressing*, *Discovery* and *Description* specify automatic integration of devices and services, *Control*, *Eventing* and *Presentation* specify how to use them.

The *Devices Profile for Web Services* (DPWS) [3], announced in August 2004 and revised in May 2005, is a profile identifying a core set of Web services that enables dynamic discovery of, and eventint capabilities for Web services. It arranges several Web service specifications such as *WS-Addressing*, *WS-Discovery*, *WS-MetadataExchange*, and *WS-Eventing* for devices, particularly. In contrast to UPnP, it supports discovery and interoperability of Web services beyond local networks. Temporarily DPWS was considered the successor of UPnP and thus subtitled with ”A Proposal for UPnP 2.0 Device Architecture” in one of its earlier versions. However, DPWS is not compatible with UPnP.

The meaningful combination of heterogeneous services – so called *Service Orchestration* – can be used to fulfill meta tasks which stand-alone services could not achieve. In the SIRENA project we dealt with the management of distributed and loosely coupled service systems running on embedded, networked devices. The mobile nature of these devices lead to new management issues and the dynamic configuration management adapting to changing environmental conditions became one of our major goals. With respect to the limited computing capabilities of embedded devices, we developed a two-phase management approach which splits up the management task into a design-time and a runtime phase. At design-time we adhere to the concepts of the model-based management approach to create the management policies whilst at runtime a small and lightweight set of management services is used to enforce the created management policies.

The remainder of this paper is organized as follows. Section 2 gives an overview of the demonstrator which shows the feasibility of the presented approach. Section 3 describes the components of the demonstrator and their functionality in detail. The evaluation results are presented in section 4, followed by a look on related work in section 5. Finally a conclusion is drawn in section 6 including an outlook to future research.

2 Architecture of the demonstrator

The presented demonstrator is part of the automotive demonstrator of the SIRENA project showing new applications of a SOA in a car. The architecture of the presented application is shown in figure 1. Its purpose is to turn off an airbag whenever a child safety seat is placed on the front seat to increase the safety of the child in the case of an accident. A UPnP network (on top of LAN and WLAN respectively) is used to connect all devices except the child safety seat which is connected via Bluetooth. The airbag is represented by an *Airbag Device* and a *Bluetooth ID Scanner* is searching for Bluetooth IDs of children safety seats. Both devices are independent from each other and only offer their own services but do not interact with each other. A third device MoBaSeC-*S Device* embeds a *Management Service*. It is configured in such a way that it browses the network to find Bluetooth ID scanners and airbag devices. If a Bluetooth ID scanner is found the MoBaSeC-S device subscribes to the scanner using UPnP Eventing. Thereby it gets informed about all Bluetooth IDs found by the scanner. Whenever a Bluetooth ID is identified as a child safety seat the airbag device is switched off using UPnP Control.

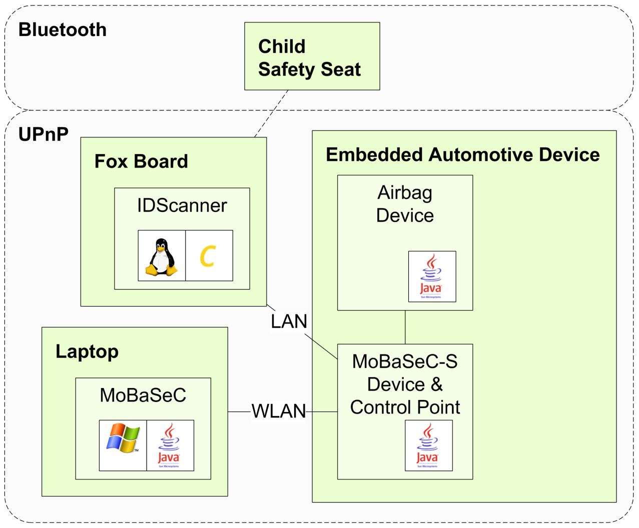


Figure 1. Architecture of the demonstrator

All components are described below in more detail.

3 Components

3.1 Bluetooth ID Scanner

When talking about loosely coupled devices and controlling them, future visions mostly end up in manual handling the devices, e.g. by remote controllers or by desktop-based applications. But there are a lot of scenarios requiring automatic invocation of services. As an example: Considering an air conditioner which should be regulated according to current temperature and time. A scheduler might be set up which defines different temperatures for different conditions. Manual regulations are not sufficient. We used the MoBaSeC tool to model such situations. Conditions may consist of state information of other devices, localization, resource, and time information, or more generally spoken of context information. To solve the localization (identification) problem in our scenario, we used a Bluetooth-enabled scanning device.

Bluetooth [4] is a well-known wireless technology, actually established as a cable-replacement for home and office computational devices and applications, mainly. Its short-range radio is operating in the ISM-band at 2.4 GHz. The Bluetooth specification also defines the *Host Controller Interface (HCI)* that acts as a uniform interface to the Bluetooth firmware. Via HCI, it is possible to search for other Bluetooth devices and services. Each Bluetooth device is equipped with a world-wide unique device ID consisting of a six byte number similar to MAC addresses of network interface cards. Device IDs along with the inquiry process make identification of devices possible. For our purpose we developed a scanning device which inquires other Bluetooth devices permanently. Since inquiry responses are only sent by devices in range, we are able discover devices near to the scanning device.

For our implementation we chose the *FOX board* available at [5]. FOX is a very small (66 x 72 mm, or 2.6 x 2.8 inches) hardware platform and can be used as an embedded system. It is equipped with a 100 MHz RISC CPU, one Ethernet and two USB ports and runs a special Linux distribution including services like a web, FTP or TELNET server. To enhance the system with UPnP and Bluetooth capabilities we connected the board with a Bluetooth USB dongle and an Ethernet cable.

The Bluetooth software component was implemented on top of *BlueZ* [6]. The C-based BlueZ API is a complete implementation of the Bluetooth stack for Linux systems. Inquiry commands are sent periodically and responses are collected. We wrapped the Bluetooth application as an UPnP service. For that purpose, an UPnP device was generated using the Intel UPnP stack [7].

3.2 Management

The two major parts of our two-phase management framework will be outlined in more detail in the following sections.

3.2.1 Model-based Management & MOBASEC

The model-based management approach uses an object-oriented model of the managed system to ease the creation and modeling of complex management policies. The graph-based modeling of such a system is supported by a graphical modeling tool. Common policy-based management approaches [8] apply low level policies to describe management demands. Systems applying the model-based approach [9] use abstract high-level policies from which concrete service configurations are automatically created.

The refinement process for policies uses the concept of a *policy hierarchy* [10] which divides the model into different layers of abstraction (cf. figure 2). On the topmost layer the model contains a very abstract description of the management objectives. On the way down to the bottom layer the model gets more and more concrete by (semi-)automatic and manual refinement of the model elements. The bottom layer finally contains a low-level policy model which, for example, can be transformed to XML-coded service configuration descriptions. The modeling of policies is supported by our MOBASEC modeling tool.

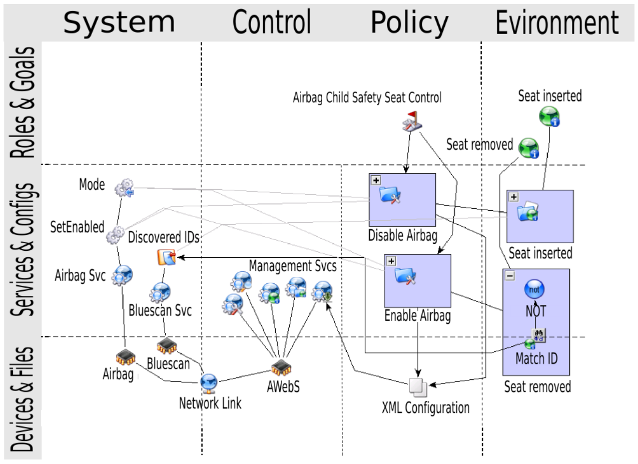


Figure 2. Model diversion overview

Besides the horizontal diversion into different abstraction layers, the model can be moreover split into four vertical parts (cf. figure 2):

System The system model contains all the real-world entities of the system to be managed. These include the devices and network connections or services with their actions and attributes, for example.

Control Services This part of the model contains a collection of services which are responsible for the runtime management of the modeled system. This includes the services for service monitoring, configuration and security management.

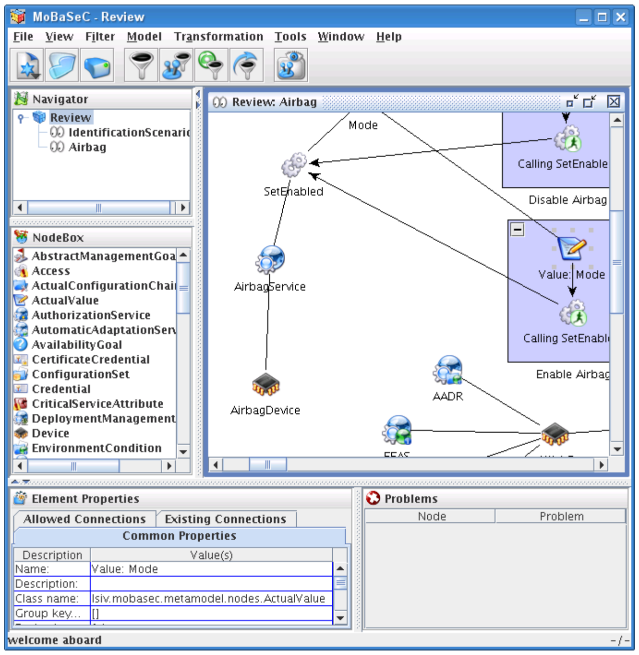
Management Policy The policy part of the model contains the description of the management actions to be applied to the system. For example, this part contains the structures for setting service attributes to specific values or calling a service's action with a particular parameter list.

Environment The environment section of the model contains the definition of the important environmental situations the system may be exposed to. These situations are directly related to particular parts of the management policy section.

For the modeling of the environment we adopted the notion of *Environment Roles* from the *Generalized Role Based Access Control (GRBAC)* [11] approach. GRBAC is an extension to the well known *Role Based Access Control* [12] model. Naturally being used for the creation of environment aware access control policies, we use environment roles as abstract representatives for environments on the topmost layer of our common management model. On the lower levels these roles are replaced by so called *environment conditions* which are logical expressions over *environment condition elements*. The elements can be used to interpret available sensor or service monitoring data. This concept enables us to model arbitrary environmental conditions based on the evaluation of simple data structures.

The information about which types of elements are available for modeling and how these can be connected to each other is encoded in the so called *meta-model*. The meta-model contains domain specific graph-node implementations, normally in form of Java classes. Each of these classes subsumes the associated behavior and properties of the real-world or policy entity represented by a model node. Moreover the meta-model contains the functions to create low-level configurations for the services that are responsible for the runtime enforcement of the modeled policies.

The models are created using our MOBASEC modeling tool for "**Mo**del-**Ba**sed **Se**rvice **C**onfiguration". The tool itself provides a modeling platform which includes a graphical user interface, graph-modeling and -transformation capabilities and filtering functions. Moreover it is possible to define different views on a model to enable a differentiated view on even very complex models. All other information regarding the modeling process is provided by domain specific meta-models.



Description	Value(s)
Name:	Value: Mode
Description:	
Class name:	lsiv.mobasec.metamodel.nodes.ActualValue
Group key...	[]

Figure 3. MOBASEC **screen-shot**

The graph transformation algorithms [14] that are part of the MOBASEC platform are used for automated model refinement. The idea of this is as follows: as stated above, the model is described by a graph. Therefore it is possible to define a graph transformation system consisting of single transformation rules which alter the graph in a specified way. One such rule consists of a pre-pattern L and a post-pattern R, and on detection of a L in the model, this occurrence is replaced with an instance of R, after ensuring that this step will not violate any connectivity restrictions of the affected nodes. L and R are graphs also, consisting of elements of the same meta-model as the model graph does. The rule elements can be specified using the tool in the same way the model is specified. Some nodes in L are identified with nodes in R, while others are not. The same applies to the edges in L and R. The algorithm to apply a rule r to a model graph M is roughly outlined by the following sequence:

1. Find an occurrence of L in M
2. Remove all components in M which occur in L, but do not have an identity component in R
3. Insert all components into M which occur in R, but do not have an identity component in L

The application of a rule r transforms the graph G to a Graph G', denoted by $G \Rightarrow G'$. This is called a direct derivation, and G' is directly derived from G. For a better control over the application of a rule, every rule is augmented with two additional components. A *condition* allows to specify certain constraints which must be met by the detected subgraph L' in order to match the defined pattern in the search. These constraints can refer to all attribute values of elements in L'. An *effect function* is a function which is applied to the inserted graph elements, and describes how the attribute values of the inserted elements are computed from the attributes of the elements of G.

The model created for the demonstrator to provide the (de)-activation of the airbag in the car based on the presence of the child safety seat is depicted in figure 4. On the left hand side of the model you can see the modeled *Airbag Service* and the *BlueScan Service* for scanning Bluetooth IDs. The Airbag service offers an action called *SetEnabled*, with the boolean parameter *Mode*. The BlueScan service has only one single service attribute called *DiscoveredIDs* which contains the IDs of the currently present Bluetooth devices. This attribute can be accessed via an eventing mechanism. The services are hosted by the *Airbag* and respectively the *Bluescan* device. Right next to the services, five management services are modeled, running on the device called *AWebS*. All devices are connected by a *Network Link* which has to provide IP connectivity to enable the services to communicate.

In the policy section of the model, the management goals and their resulting tasks are modeled. On the topmost layer a management goal named *Airbag-Child Safety Seat Control* is modeled. This abstract goal is refined to the management tasks modeled on the middle layer. The lower task enables the airbag by calling the *SetEnabled* action with the value for *Mode* set to *true*, while the other task disables the airbag by setting *Mode* to *false*. Management tasks are grouped together using so called *folder nodes*, which can be opened or collapsed to improve the readability of a model. On the right hand side of the model you can see the environments for our demonstrator scenario. On the upper layer the environments are represented by very abstract environment roles called *Child Seat inserted* and *Child Seat removed*. These roles are refined to appropriate environment conditions which define what sensor data should be used to determine a special environment and how this data should be interpreted. The *Seat removed* environment condition contains the definition for the situation when the child safety seat is not present. This environment is active, whenever the *DiscoveredIDs* attribute of the *BlueScan* service does not contain the Bluetooth ID of the child safety seat. The management tasks are connected to the appropriate environment condition elements.

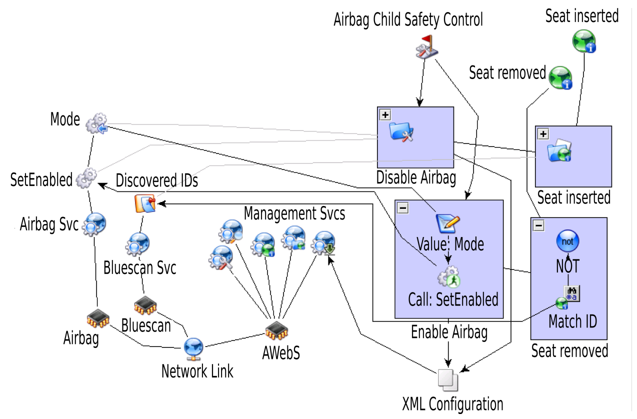


Figure 4. Airbag control model

From models like the one depicted in figure 4 the low-level XML policy descriptions can be created. For these descriptions we developed an XML Schema document that defines the structure and elements of the configuration description. The configuration description is action centric, that means all management tasks are encoded as actions of services and whenever a configuration change has to take place, this change is defined as an action call on the affected target service. This very flexible concept allows us to define the configuration of the management services and managed services the same way.

A short example of the XML structure is outlined below:

```
<config version="v1" ...>
 <serviceConfigurations>
  <service type="ServiceType" ...>
   <action name="ActionName" ...>
    <parameter name="ParameterName" ...>
     <value>...</value>
     </parameter>
     ...
  </service>
  ...
 </serviceConfigurations>
</config>
```

3.2.2 Policy Enforcement

So far we have outlined the modeling of management policies by means of graphical modeling and model refinement using the MoBaSec tool. The automated creation of the low level policies concludes the design phase of the management process. At runtime, a set of dedicated services is used to enforce the given policies and to enable the adaptive runtime management of the system. The provided infrastructure consists of five different services, each responsible for a special subtask of our runtime enforcement process. The *Deployment Service* is responsible for the initialization of the service infrastructure. The *Monitoring Service* subsumes all functionality for the runtime monitoring of services by using polling or eventing mechanisms. The activation state of the environment elements is managed by the *Environment Element Activation Service*. The *Automatic Adaptation/Dynamic Reconfiguration Service (AADR)* detects the environmental situation a system is exposed to, and finally the *Service Control Management Service* activates the appropriate configuration for a particular environment. The service infrastructure is also depicted in figure 5.

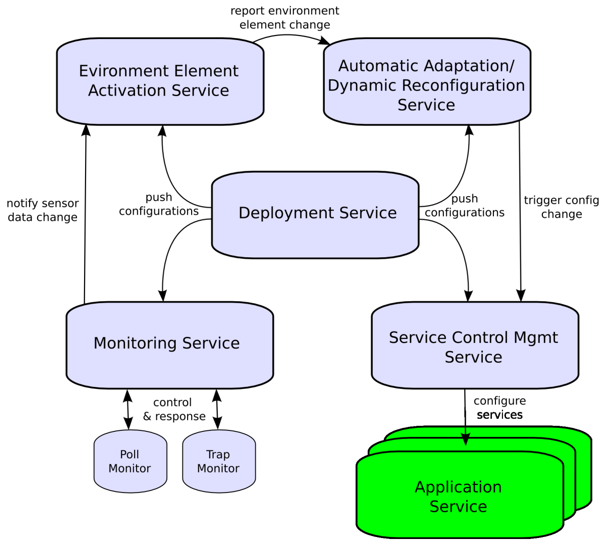

report environment element change
Evironment Element Activation Service
Automatic Adaptation/ Dynamic Reconfiguration Service
notify sensor data change
push configurations
Deployment Service
push configurations
trigger config change
Monitoring Service
Service Control Mgmt Service
control & response
configure services
Poll Monitor
Trap Monitor
Application Service

Figure 5. Runtime service infrastructure

The low-level configuration derived from the model can be directly deployed onto an existing management service infrastructure from within the MoBaSec tool. As soon as a new configuration arrives, the Deployment Service splits up the configuration data and forwards the appropriate configuration parts to the other management services. After the initialization phase the services are ready to enforce the given management policy. This process is as follows: the Monitoring Service gathers information about the current system environment by listening for incoming events or polling appropriate service actions. As soon as a change in the environment is detected, the name of the sensor element and the current value are forwarded to the Environment Element Activation Service. This service reevaluates the state of all environment condition elements that depend on the given sensor element. If this evaluation leads to a change in the set of active elements, this set is sent to the AADR. This service now starts an evaluation of the configured environment conditions which are encoded by means of logical formulas. If an environment condition is found to be active, and it was not active before, the AADR resolves the identifier of the associated configuration set and sends a message containing this identifier to the Service Control Management Service. A single configuration set contains all service configuration descriptions that are associated with a particular environment condition. Finally the Service Control Management Service loads the appropriate configuration and performs the enclosed management actions.

Service Implementation The service infrastructure introduced in the last paragraph was implemented using the Java programming language. Before implementing the services we developed an abstraction layer for service oriented architectures that hides the specific low-level implementation of UPnP or DPWS from the services implementations. Thus for porting the services to the DPWS platform we only have to implement a new communication module for the abstraction layer, while the services above the abstraction layer remain unchanged. The footprint of our implementation's compressed byte code is about 420kB with $\approx$ 80kB for the services implementations, $\approx$ 35kB for the abstraction layer, and about 300kB for the UPnP libraries.

4 Evaluation

The application and management services described in the last section where used to implement the automotive and cross-domain demonstrators from section 2. After the initial deployment and startup of all involved components, the management services where initialized with the configuration created from the above-mentioned model of the demonstrator scenarios. This initialization phase started with the *online-deployment* of the actual configuration from within the MoBaSeC tool. The initialization phase took about 30 secs. from starting the deployment in MoBaSeC until the service system was in an completely initialized state.

As the child safety seat was installed in the car, it was recognized and the management services enforced the modeled configuration change and thus set the airbags state *virtually* to *off*. After the seat had been removed, the airbag was reactivated again. The management system's response time was about 700 msecs. from monitoring a change of a sensor state to the enforcement of the new configuration at the airbag service.

The demonstrator showed that our management approach was applicable for the management of (UPnP) service systems. The airbag and bluetooth scanner services were combined together without having to implement any additional line of source code, just by creating a model of the managed system and instrumenting the management runtime service infrastructure. We also used our approach to create a personalisation service inside a car using the onboard AWebS box and a PDA.

5 Related Work

In [13] Lobo et al. describe a platform called ”Policy Management for Autonomic Computing” (PMAC). Based on an extensive policy language, management policies can be created using a so called *Policy Definition Tool*. During a policy ratification process the available policies are checked for dominance or conflict issues. The policies are used by an *Autonomic Manager* for the automated runtime management of managed resources. The main difference to the approach presented in this paper is that PMAC indeed uses a policy definition tool to support the user in creating the management policies but does not apply the model-based management approach to create management models.

6 Conclusion & Outlook

The implementation of the demonstrators showed the applicability of the management approach that was developed during the SIRENA project. Therefore our future work will concentrate on the improvement of the meta-model and refinement rules to make the policy creation process much more friendly. This includes the evaluation of so called *Management Patterns*. These patterns can be seen as a toolbox of refinement patterns that are able to deal with management issues that may arise in domains based on service oriented architectures. Moreover the implementation of a small DPWS stack for Java and its integration into the abstraction layer the management services are based on is another task we are currently working on. Finally we have to conduct more tests in complex service management scenarios to confirm our initial results.

This work was supported by the SIRENA project in the framework of the European premier cooperative R&D program ”ITEA”.

References

[1] SIRENA (Service Infrastructure for Real-time Embedded Networked Applications), `http://www.sirena-itea.org`, 2004.

[2] UPnP Forum, UPnP Device Architecture v.1.0.1, `http://www.upnp.org/resources/documents.asp`, 2003.

[3] Microsoft, Devices Profile for Web Services, `http://specs.xmlsoap.org/ws/2005/05/devprof/devicesprofile.pdf`, 2005.

[4] The Bluetooth Special Interest Group, Specification of the Bluetooth System 1.2, 2004.

[5] ACME Systems, FOX BOARD a complete Linux system on a small board, `http://www.acmesystems.it/?id=4`, 2005.

[6] BlueZ, Official Linux Bluetooth protocol stack, `http://www.bluez.org/`, 2005.

[7] Intel, Intel Software for UPnP Technology, `http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/index.htm`, 2005.

[8] M. Sloman, Policy Driven Management for Distributed Systems, *Journal of Network and Systems Management*, 2(4), 1994.

[9] Ingo Lück, Sebastian Vögel and Heiko Krumm, Model-based configuration of VPNs, *8th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Florence, Italy, 2002, 589–602.

[10] René Wies, Using a Classification of Management Policies for Policy Specification and Policy Transformation, *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, California, USA, May 1995.*

[11] Matthew J. Moyer and Mustaque Ahamad, Generalized Role-Based Access Control, *Proc. 21st Int. Conf. on Distributed Computing Systems, Mesa, USA, 2001, 391–398*.

[12] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein and Charles E. Youman, Role-Based Access Control Models, *IEEE Computer*, 29(2), 1996, 38–47.

[13] Dakshi Agrawal, Kang-Won Lee, Jorge Lobo, Policy-Based Management of Networked Computing Systems, *IEEE Communications Magazine*, vol. 43, no. 10, October 2005.

[14] M. Andries et. al.,“Graph Transformation for Specification and Programming”, *Science of Computer Programming,* Vol. 34, pp. 1–54, 1999.