Modeling of Domain-Specific ECA Policies

Raphael Romeikat
University of Augsburg
Institute of Computer Science
Augsburg, Germany
romeikat@ds-lab.org

Bernhard Bauer
University of Augsburg
Institute of Computer Science
Augsburg, Germany
bauer@ds-lab.org

Henning Sanneck
Nokia Siemens Networks
Research
Munich, Germany
henning.sanneck@nsn.com

***Abstract*—Policy-based management is a flexible approach for the management of complex systems as policies make context-sensitive and automated decisions. For the effective development of policies it is desired to specify policies at a high level of abstraction initially, and to refine them until they are represented in a machine-executable way. We present an approach that uses models to specify event-condition-action (ECA) policies at different levels of abstraction. A relational algebra is used to specify and validate the models in a formal way. Finally, executable policy code is generated from the low-level models. The approach is generic as it can be applied to any domain and supports a flexible number of abstraction layers. It is applied to the network management domain and demonstrated with the modeling and refinement of policies for coverage optimization in a mobile network.**

***Keywords*-domain; policy; ECA; modeling; refinement; MDE; network management**

I. Introduction

Increasing complexity of enterprise computing systems complicates their development and management. This evolution calls for changes in the way enterprise computing systems are built. The aspect of development is addressed by the Model-Driven Engineering (MDE) approach, which moves the focus from a code-centric to a model-centric point of view [1]. Models raise the level of abstraction and reduce complexity by separating different concerns of a system from each other. Models in MDE are no more used for discussion and documentation purposes only, but they are used as primary artifacts from which implementations are generated [2]. Apart from development, increasing complexity also affects the management of systems at runtime. This issue is addressed by the idea of self-organizing systems. One example is the Autonomic Computing Initiative by IBM, which proposes self-manageable systems in order to reduce human intervention necessary for performing administrative tasks [3].

Policies represent a promising technique for realizing autonomic capabilities within managed objects as they allow for a high level of automation and abstraction. Policy-based management has gained attention in research and industry as a management paradigm as it allows administrators to adapt the behavior of a complex system without changing source code or considering technical details. A system can continuously be adjusted to externally imposed constraints by changing the determining policies [4]. A well-known application area is network management, where policies are widely used for performing configuration processes. The usage of policy-based systems for management of mobile networks was recently considered in [5]–[10].

The event-condition-action (ECA) model is a common way to specify policies. ECA policies represent reaction rules that specify the reactive behavior of a system. An ECA policy correlates a set of events, a set of conditions, and a set of actions to specify the reaction to a certain situation. The conditions are evaluated on the occurrence of an event and determine whether the policy is applicable or not in that particular situation. The actions are only executed if the conditions are met. Multiple policy frameworks share this model as for example Ponder2 [11].

Policy-based management is a layered approach where policies exist at different levels of abstraction. For simple systems it might be sufficient to have one or two abstraction levels only, one with a business view and another one with a technical view. For larger systems or systems in a complex domain it is reasonable to introduce additional levels between the business and the technical level in order to allow for domain and policy representation at intermediate abstraction levels. Strassner defines a flexible number of abstraction layers as the Policy Continuum [5]. The idea is to define and manage policies at each level in a domain-specific terminology, and to refine them from a business level down to a technical level.

This paper is structured as follows. Section II describes how domain-specific policies are modeled at different levels of abstraction before executable code is generated. Section III provides an example from a case study. Related work is discussed in section IV. The paper concludes with a summary and future work in section V.

II. Modeling

Different types of models are used at different abstraction layers in order to specify domain-specific policies as illustrated in figure 1. The domain model allows domain experts to specify domain-specific concepts that are available in a system. The policy model allows policy experts to specify

policies that are used to manage a system. The linking model allows to link the policy model to the domain model in order to use the domain-specific concepts within the policies. For each of those models a metamodel exists that defines the structure of the model. Two layers i and j are shown exemplarily in figure 1 with layer i providing a higher level and layer j providing a lower level of abstraction. Actually, the approach supports a flexible number of abstraction layers.

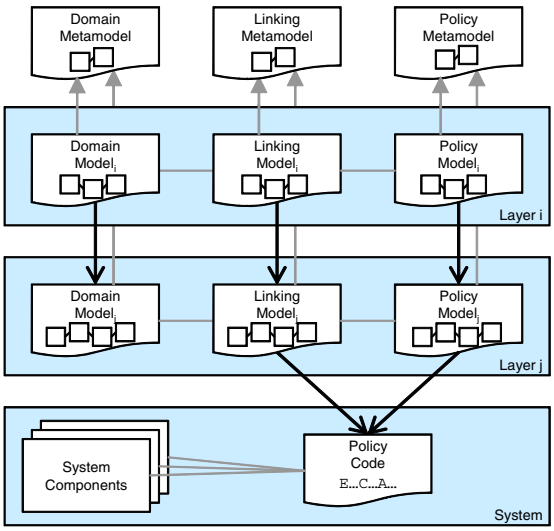

Figure 1. Methodology

A relational algebra was developed to formally define the semantics of the domain, policy, and linking metamodels and their views at the different abstraction layers. Excerpts of the algebra are presented in this paper.

A. Domain Modeling

Different expert groups are involved in the management of an enterprise computing system such as business managers or system administrators. Depending on their focus and their background, members of an expert group have a particular view on the system and they use special terminology to describe their knowledge. The *domain* represents a common understanding of those expert groups and covers the context of an enterprise computing system.

Any relevant information about the domain is covered by the *domain model*. The domain model covers the domain-specific concepts across all abstraction layers and specifies which domain-specific concepts are available. Its purpose is to specify domain knowledge independently from any policies, which will later control a system in that domain. Thus it represents the basis for building policies, which will then use domain concepts in their event, condition, and action parts. The domain model offers a particular view at any layer, which only contains the part of the domain model that is relevant at the respective layer. The domain model is an instance of the *domain metamodel*, which allows to specify domain models in a way that is more expressive than just a domain-specific vocabulary and close to the structure of an ontology. For this purpose, the metamodel represents domain-specific knowledge as shown in figure 2. It represents the abstract syntax of the domain, i.e. it defines the structure of the domain model.

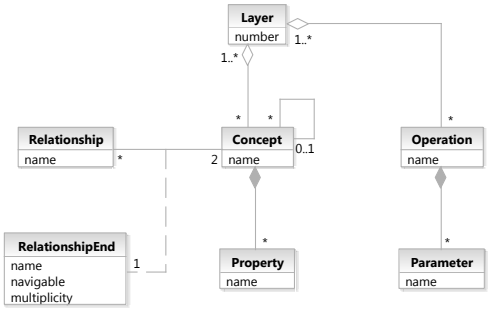

Figure 2. Domain metamodel

B. Policy Modeling

In the same way as expert groups have a particular view onto the domain, they also have a particular view onto the policies that control a system in that domain. A business expert e.g. uses different terminology to express a policy than a system administrator would do for the same policy. Also, a business policy might be represented by several technical policies at a lower abstraction layer.

Any information about the policies is covered by the *policy model*. The policy model offers a particular view at any layer, which only contains the part of the policy model that is relevant at the respective layer. The policy model is an instance of the *policy metamodel*, which contains the essential aspects required to define basic ECA policies. It is shown in figure 3 and represents the abstract syntax of policies, i.e. it defines the structure of the policy model. Various policy languages are available for different domains and application areas. A policy language usually provides some structured notation for policies and can be interpreted by an execution engine. Typically, such engines provide means to cope with priorization and conflict resolution, which reduces complexity for the developer. When developing a policy-based system, a decision for a policy language has to be made at some point. Unfortunately, no policy language is general and powerful enough to meet the requirements of any system in any domain. It is desired to specify policies independently from a particular policy language and to generate code for a particular language. For this purpose some well-known policy languages such as PonderTalk [11], KAoS [12], and Rei [13] were analyzed. Their common concepts are considered by the policy metamodel.

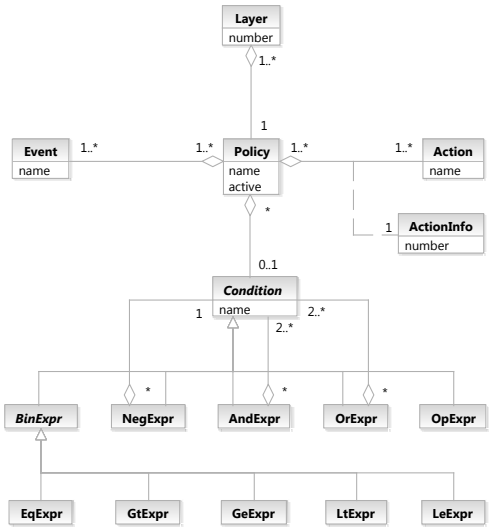

Figure 3. Policy metamodel

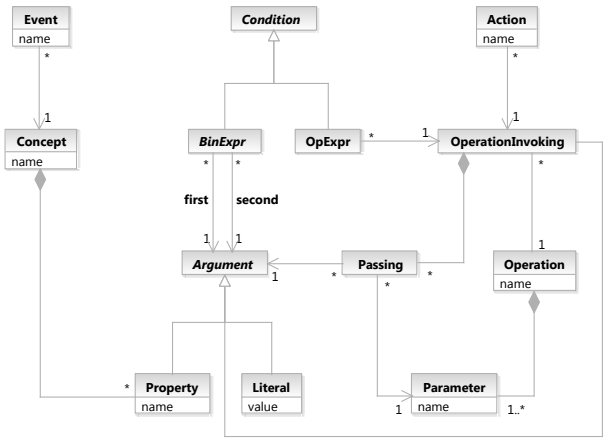

Figure 4. Linking Metamodel

C. Domain-Specific Policy Modeling

Domain and policies have been modeled independently from each other so far. The domain is specified as the domain model and policies are specified as the policy model. Now, both models must be combined in order to use the entities of the domain within the policies. For this purpose, a third type of model enables policies to refer to domain-specific information in their event, condition, and action part.

Any information about how domain-specific information is used within the policies is covered by the *linking model*. It specifies how the domain and the policy models are linked to each other. For this purpose, it allows to create links from the entities in the policy model to the entities in the domain model at the respective layers. The linking model offers a particular view at any layer, which only contains the links that are relevant at the respective layer. The linking model is an instance of the *linking metamodel*, which provides means to create links from the policy model to the domain model as shown in figure 4. It represents the abstract syntax of the links, i.e. it defines the structure of the linking model.

The formal specification defines the structure of the linking model. Additionally, some restrictions must hold on the contents of a linking model. Those restrictions are used in order to validate wheter a linking model is valid or not. One example is the usage of context information within a policy. Context information is usually passed to a policy via its events. The event properties contain information to be used within the policy and can be referenced in the policy condition and action. It is important that only properties are used within a policy that are visible for that policy. A property is visible for a policy if it belongs to a domain concept that occurs as event of that policy. This restriction is covered by (1) to (3) in the formal specification.

$$arg \in argumentsOfCondition(cd) \Rightarrow (arg \notin Properties \vee \forall po \in policiesOfCondition(cd). arg \in visibleProperties(po)) \quad (1)$$

$$arg \in argumentsOfAction(ac) \Rightarrow (arg \notin Properties \vee \forall po \in policiesOfAction(ac). arg \in visibleProperties(po)) \quad (2)$$

$$visibleProperties : Id_{Po} \rightarrow \mathcal{P}(Id_{Pr}) \quad po \mapsto \bigcup_{co \in visibleConcepts} propertiesOfConcept(co) \quad \text{with } visibleConcepts = \bigcup_{ev \in eventsOfPolicy(po)} conceptOfEvent(ev) \quad (3)$$

D. Code Generation

Once the domain, policy, and linking models have been refined to the lowest abstraction layer they are represented in a machine-executable way. The domain model represents the necessary concepts of the underlying system components and the policy and linking models represent policies that control the behavior of those components. Executable code in a policy language can now be generated from the models. This applies to any language that is able to express ECA policies as defined by the policy metamodel.

For this purpose, model transformations generate the policy code in a fully automated way. This involves model-to-model and model-to-text transformations, which first transform the policy and linking models into an intermediate representation and then into executable policy code. Transformation was implemented as proof of concept for the Ponder2 policy framework [11], which was developed at Imperial College over a number of years. Details of the transformation are presented in [14].

III. Case Study

Management of information systems is a complex task especially in the management of mobile networks. Complexity arises from the distributed architecture of the underlying cellular network with its high number of network elements (NEs) to be deployed and managed and interdependencies between their configurations.

Operation, administration, and maintenance (OAM) of a mobile network is usually based on a centralized information system which is organized in different management domains. Configuration management (CM) deals with a consistent configuration of all NEs, performance management (PM) analyzes the efficiency of the current network configuration and seeks a more efficient one, and fault management (FM) detects and resolves errors that occur in the network. Any management domain focusses on different aspects and has a special view onto the network and the configurations of the NEs.

A. Policy-Based Coverage Optimization

One important management task within any cellular network is coverage optimization. The cells of the network should always provide a complete coverage without areas having no coverage at all. The detection and resolution of converage holes is a complex task as it affects any of the CM, PM, and FM domains. The coverage area of each cell is determined through multiple factors.

- **Position**: Location and direction of an antenna are the main determining factors for the coverage area of a cell. These parameters are preplanned and are almost not adjustable after deployment as this would require expensive human on-site intervention.
- **Transmission power**: Later adaptations of the transmission power (TXP) have a direct impact on the size of a cell's coverage area. Adaptations can be performed remotely through the operation and maintenance system.
- **Antenna tilt**: Tilt influences coverage similarly to TXP. A rough mechanical adjustment is done when the antenna is deployed. Later adaptations of the antenna tilt are done through remote electrical tilt changes (RET) within the antenna.

In order to optimize coverage within the network, usually a sequence of power and tilt adaptations is used since they can be executed remotely. After each change the situation is re-evaluated by analysis of measurement reports. The results of this evaluation are used to determine whether additional adaptions are required or not. Due to the cellular structure of the network, power and tilt adaptations have a strong impact. Adaptions at a single cell may have impact on adjacent cells and may result in a conflict and thus undesired state of the network.

The dependencies between power and tilt adaptions require subsequent adaptions to be coordinated with each other in order to ensure that no logical errors, oscillations, or even deadlocks occur in the configuration. For this purpose a policy-based coordination mechanism was developed that takes decisions in an automated way [9], [10]. The coordination policies express the decision logic to determine if a change request should be acknowledged, rejected, or rescheduled, and if previously executed requests should be rolled back. They are represented at multiple abstraction layers. The highest layer represents a management point-of-view whereas the lowest layer represents a specific implementation for the underlying management system that uses Ponder2 [11] as policy framework.

B. Policy Modeling and Code Generation

The behavior of coverage optimization is controled by a set of ECA coordination policies. Policy modeling makes it possible to model those policies at a high level initially and to refine them afterwards. For this purpose, a domain model was specified that covers the relevant concepts for coverage optimization at different abstraction layers. The necessary coordination policies were specified as policy model at the highest layer and they were linked to the domain model with a linking model. An example is shown in the following. One coordination policy ensures that tilt change requests are rescheduled if a power change is already active. An excerpt of the respective high-level models is shown in figure 5. The refined low-level policy provides the same functionality as the high-level one but uses the technical concepts of the underlying management system. An excerpt of the respective low-level models is shown in figure 6.

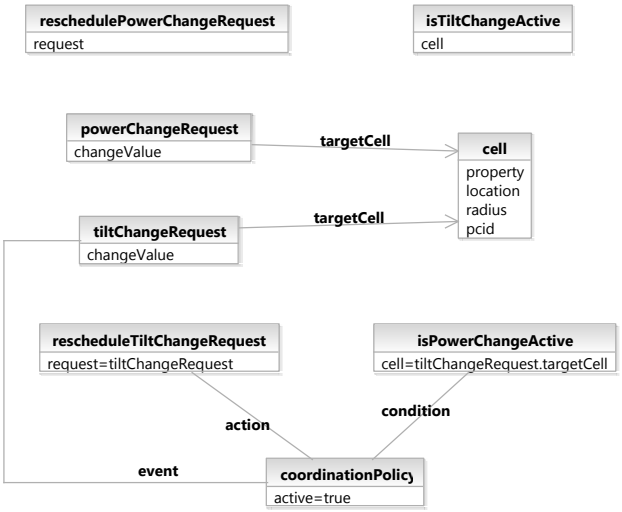

Figure 5. High-level model

During refinement, some high-level entities are replaced with low-level ones, e.g. the *tiltChangeRequest* concept with the *reconfRequest* one or the *isPowerChangeActive.targetCell* parameter with the *isReconfActive.property*

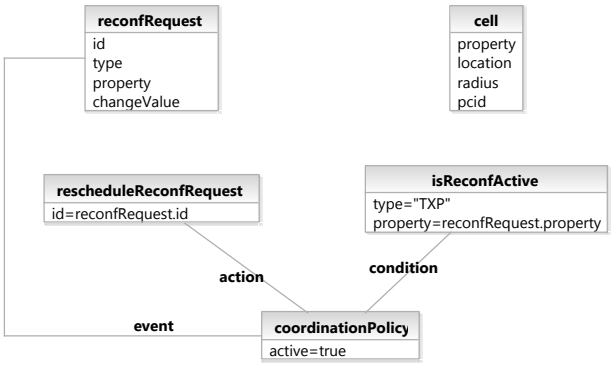

Figure 6. Low-level model

one. Furthermore, information is added by means of the additional parameter *isReconfActive.type* and the passed value *"TXP"* in the invoking of the *isReconfActive* operation.

The case study was used as a proof of concept of the approach. The concepts of the network management system were specified as domain model and the policies of the coordination mechanism were specified as policy model and then linked to the domain model. The models were initially modeled at the highest layer and then refined into a machine-executable representation at the lowest layer. In the end the refined models were automatically transformed into the Ponder2 code shown in listing 1. This code is directly used by the underlying simulation system [15].

```
1 policy := root/factory/ecapolicy create.
2 policy event: root/event/reconfRequest;
3       condition: [ :reconfRequest.id :reconfRequest.type
            :reconfRequest.property :reconfRequest.
            changeValue | root/op isReconfActive:"TXP"
            property:reconfRequest.property ];
4       action: [ :reconfRequest.id :reconfRequest.type :
            reconfRequest.property :reconfRequest.
            changeValue | root/op rescheduleReconfRequest:
            reconfRequest.id ].
5 root/policy at:"coordinationPolicy" put:policy.
6 policy active: true.
```

Listing 1. Generated Ponder2 Code

IV. Related Work

This section summarizes approaches for the specification and refinement of policies that are comparable to the approach presented in this paper. The features are compared to the features of the other apporaches and their commonalities and differences are outlined in table I.

The authors of [16] present a model-driven approach to design policies and integrate them into the software development process. The approach is based on MDE concepts and uses a UML profile for modeling policies. GPML supports different types of policy and ECA policies are represented by the obligation policy type. The ability to define a particular vocabulary allows to adapt policies to different domains. Policies are modeled at a low level of abstraction and cannot be refined. Model transformations are used to map GPML policies via an interchange format to existing policy languages. No formal semantics is provided.

The CIM Policy Model [17] by the DMTF addresses the management of complex multi-vendor environments with a huge number of heterogeneous devices. Policies are specified in a language-independent way and abstract from hardware characteristics. A UML profile is provided for the graphical representation of policies. The CIM Policy Model is a domain-specific model with a focus on network management. Different abstraction levels and policy refinement are not supported. The approach does not address code generation for existing policy languages. Formal semantics is not provided.

DEN-ng [6] by the TMF provides an information model for system entities and policies to manage those entities. It allows entities and policies to be modeled in an implementation-independent way while omitting technical details. ECA policies can be modeled in a UML diagram. DEN-ng allows to specify a domain and resources within that domain on which policies operate. DEN-ng is based on the Policy Continuum and considers policies at different levels of abstraction. Specification of policies is addressed but policies cannot be refined automatically into a lower level of abstraction, nor transformed into an existing language. No formal semantics is provided.

[7] presents a refinement approach that focuses on policies in the autonomic networking domain. Policies represent configuration settings and are used for automated network and resource configuration. A simple policy language offers a fixed terminology to specify domain-specific policies. Event-based policies are not supported. Policies are represented at five levels of a Policy Continuum and each layer offers a sub-set of that terminology. A wizard provides a graphical user interface to specify configuration policies on the highest level. Those policies are automatically refined into concrete configuration commands on a per-device basis. For this purpose, XSLT transformations replace higher-level objects with the respective objects at the lower levels. Formal semantics and code generation for existing languages are not provided.

V. Conclusion

A model-based approach to the specification of ECA policies was presented in this paper. The usage of models allows to specify policies in a graphical way at a high level of abstraction initially. This avoids the immediate implementation of policies at a technical level. Refinement of domain-specific policies is simplified in several ways. Domain and policy aspects are consequently separated from each other into different models. The usage of different models at different layers faciliates the collaboration of different experts groups. Additionally, the respective models

	GPML [16]	CIM Policy Model [17]	DEN-ng [6]	Policy Continuum [7]	Domain-Specific Policy Modeling
ECA policies	+	o	+	-	+
Graphical modeling	+	+	+	+	+
Formal specification	-	-	-	-	+
Language-independent	+	+	+	-	+
Customizable domain	+	-	+	-	+
Abstraction levels	-	-	+	+	+
Automated refinement	-	-	-	+	-
Code generation	+	-	-	-	+

Table I
COMPARISON OF RELATED WORK

are instances of the same metamodels at any abstraction layer. A relational algebra precisely defines the semantics of the models and allows for their validation, which is a prerequisite for the transformation into executable code. The approach is novel as it is generic with respect to the domain and to the number of abstraction levels and nevertheless allows to generate executable code.

The possibility to generate code eliminates the dependency from a particular policy language as the same models can be used to generate code for various languages. A prototype of a graphical policy editor that supports code generation for Ponder2 has already been developed [14], [18] and is to be developed further. Automating policy refinement is also subject to future work. Currently the lower-level models are created manually based on the higher-level ones. However, it should be sufficient to specify the refinement of the domain model once and then apply that refinement to the policy and linking models in order to generate a refined representation of them whenever they are created or modified. In the end the objective is to automatically reflect changes of the high-level models in their low-level implementation.

REFERENCES

[1] J. Bézivin, "On the Unification Power of Models," *Software and Systems Modeling*, vol. 4, no. 2, pp. 171–188, May 2005.

[2] D. W. Flater, "Impact of Model-Driven Standards," in *35th Annual Hawaii International Conference on System Sciences*, vol. 9. IEEE CS, January 2002, pp. 3706–3714.

[3] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, January 2003.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language," in *2nd Workshop on Policies for Distributed Systems and Networks*, vol. 1995. Springer LNCS, 2001, pp. 18–38.

[5] J. C. Strassner, *Policy-Based Network Management: Solutions for the Next Generation*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2003.

[6] J. Strassner, "DEN-ng: Achieving Business-Driven Network Management," in *8th Network Operations and Management Symposium*. IEEE CS, April 2002, pp. 753–766.

[7] S. van der Meer, A. Davy, S. Davy, R. Carroll, B. Jennings, and J. Strassner, "Autonomic Networking: Prototype Implementation of the Policy Continuum," in *1st International Workshop on Broadband Convergence Networks*, April 2006, pp. 1–10.

[8] T. Bandh, H. Sanneck, L.-C. Schmelz, and G. Carle, "Automated Real-time Perfomance Management in Mobile Networks," in *1st WoWMoM Workshop on Autonomic Wireless AccesS*. IEEE CS, June 2007.

[9] R. Romeikat, B. Bauer, T. Bandh, G. Carle, H. Sanneck, and L.-C. Schmelz, "Policy-driven Workflows for Mobile Network Management Automation," in *6th International Wireless Communications and Mobile Computing Conference*. ACM, June 2010, pp. 1111–1115.

[10] T. Bandh, R. Romeikat, and H. Sanneck, "Policy-based Coordination and Management of SON Functions," in *12th International Symposium on Integrated Network Management*, May 2011, to be published.

[11] K. Twidle, E. Lupu, N. Dulay, and M. Sloman, "Ponder2 - A Policy Environment for Autonomous Pervasive Systems," in *9th Workshop on Policies for Distributed Systems and Networks*. IEEE CS, June 2008, pp. 245–246.

[12] A. Uszok, J. M. Bradshaw, and R. Jeffers, "KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services," *Lecture Notes in Computer Science*, vol. 2995, pp. 16–26, January 2004.

[13] L. Kagal, T. Finin, and A. Joshi, "A Policy Language for a Pervasive Computing Environment," in *4th International Workshop on Policies for Distributed Systems and Networks*, June 2003, pp. 63–74.

[14] R. Romeikat, M. Sinsel, and B. Bauer, "Transformation of Graphical ECA Policies into Executable PonderTalk Code," in *3rd International Symposium on Rule Interchange and Applications*. Springer LNCS, November 2009, pp. 193–207.

[15] T. Bandh, H. Sanneck, and R. Romeikat, "An Experimental System for SON Function Coordination," in *International Workshop on Self-Organizing Networks*, May 2011, to be published.

[16] N. Kaviani, D. Gasevic, M. Milanovic, M. Hatala, and B. Mohabbati, "Model-Driven Engineering of a General Policy Modeling Language," in *9th Workshop on Policies for Distributed Systems and Networks*. IEEE CS, 2008, pp. 101–104.

[17] Distributed Management Task Force, "CIM Policy Model White Paper," DSP0108, June 2003.

[18] University of Augsburg, "PolicyModeler," August 2009. [Online]. Available: http://policymodeler.sf.net