Formal specification of domain-specific ECA policy models

Raphael Romeikat, Bernhard Bauer

Angaben zur Veröffentlichung / Publication details:

Romeikat, Raphael, and Bernhard Bauer. 2011. “Formal specification of domain-specific ECA policy models.” In *Proceedings of the 2011 Fifth International Conference on Theoretical Aspects of Software Engineering, TASE 2011, 29-31 Aug. 2011, Xi’an, Shaanxi, China*, edited by Zhenhua Duan and Luke Ong, 209–12. Los Alamitos, CA: IEEE. https://doi.org/10.1109/tase.2011.29.

Formal Specification of Domain-Specific ECA Policy Models

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

***Abstract*—Policy-based management allows to adapt systems to changed requirements in a flexible and automated way. Policy development usually starts with the specification of high-level policies, which are then refined into a low-level representation. We use models to specify event-condition-action (ECA) policies at different levels of abstraction and consequently separate domain and policy aspects from each other. Domain-specific concepts are used within policies in their event, condition, and action parts. We present a formal specification of the models by means of a relational algebra. The algebra is used to validate the models at each level. Finally, executable policy code is generated from the low-level models.**

***Keywords*-policy-based management; model-driven engineering; formal specification**

I. Introduction

The idea behind policy-based management is to control and manage a complex system on a high level of automation and abstraction. Policies make context-sensitive decisions in an autonomous way and they allow to modify system behavior according to externally imposed constraints [1]. Network management is a well-known application domain where policies are widely used for performing configuration processes [2]–[7]. The event-condition-action (ECA) model is a common way to specify policies. An ECA policy describes the reactive behavior of a system to a certain situation and for this purpose correlates a set of events, a set of conditions, and a set of actions. 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 [8].

Policy-based management is a layered approach where policies exist at different levels of abstraction. Strassner defines a flexible number of abstraction layers as the Policy Continuum [2]. 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. A model-based approach for the specification of policies at different levels of abstraction that supports the refinement process was presented in [9]. Refinement starts with high-level policy models and iteratively re-writes them with the means of the lower levels. To support this we present a formal approach for the specification of those models. The models are validated at each level of abstraction based on their formal representation. Furthermore, the formal specification will help to automate the refinement process and to describe the semantics of an automated policy refinement.

This paper is structured as follows. Section II describes how policy models are formally specified at different levels of abstraction before executable code is generated. Related work is discussed in section III. The paper concludes with a summary and future work in section IV.

II. Formal Specification

We use different models at different abstraction layers in order to specify policies as illustrated in figure 1 [9]. The domain model allows domain experts to specify concepts of a domain or system. The policy model allows policy experts to specify policies that are used to manage the system. The linking model allows policy and domain experts to link the policy model to the domain model in order to use the domain-specific concepts within the policies. These models are actually parts of one large model. For each of them 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. The lowest layer represents the models such that executable policy code can be generated from them as described in [10].

We developed a relational algebra to formally specify the domain, policy, and linking metamodels, i.e. the abstract syntax of the models. The algebra is used to validate whether model instances conform to the respective metamodel. For this purpose, a concrete syntax should provide a transformation into the relational algebra. Excerpts of the algebra are presented in this paper.

A. Domain Modeling

Any relevant information about the domain is covered by the *domain model*. The domain model covers the domain-specific concepts across all layers and specifies which concepts are available. Its purpose is to specify domain

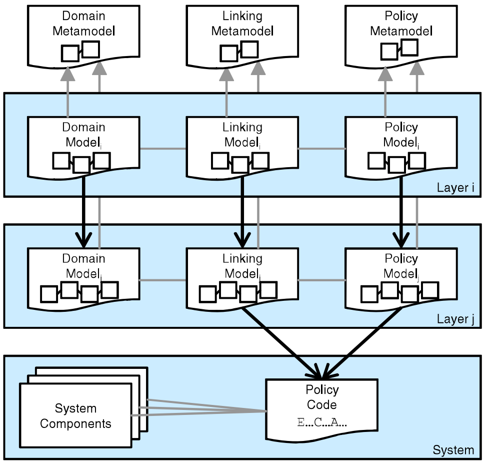

Figure 1. Overview

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-specific 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. It represents the abstract syntax of the domain, i.e. it defines the structure of the domain model.

Definitions (1) to (17) represent an excerpt from the formal specification and illustrate how the domain model and its entities are defined. Let Σ be a non-empty finite set called *alphabet*.

$$Val_T := \Sigma^* \tag{1}$$
$$Val_N := \mathbb{R} \tag{2}$$
$$Val_B := \{true, false\} \tag{3}$$
$$Val := Val_T \uplus Val_N \uplus Val_B \tag{4}$$
$$Id_{La} \subseteq \mathbb{N} \tag{5}$$
$$Id_{Co}, Id_{Pr}, Id_{Op}, Id_{Pa}, Id_{Re} \subseteq Val_T \tag{6}$$

$$Concept(Layer, Name) \subseteq Id_{La} \times Id_{Co} \tag{7}$$
$$Co := \{co | \exists la.(la, co) \in Concept\} \subseteq Id_{Co} \tag{8}$$

$$Property(Concept, Name) \subseteq Co \times Id_{Pr} \tag{9}$$
$$\begin{aligned}&(co_1, pr) \in Property \wedge (co_2, pr) \in Property\\ &\quad \Rightarrow co_1 = co_2\end{aligned} \tag{10}$$
$$Pr := \{pr | \exists co.(co, pr) \in Property\} \subseteq Id_{Pr} \tag{11}$$

$$Operation(Layer, Name) \subseteq Id_{La} \times Id_{Op} \tag{12}$$
$$Op := \{op | \exists la.(la, op) \in Operation\} \subseteq Id_{Op} \tag{13}$$

$$Parameter(Operation, Name) \subseteq Op \times Id_{Pa} \tag{14}$$
$$\begin{aligned}&(op_1, pa) \in Parameter \wedge (op_2, pa) \in Parameter\\ &\quad \Rightarrow op_1 = op_2\end{aligned} \tag{15}$$
$$Pa := \{pa | \exists op.(op, pa) \in Parameter\} \subseteq Id_{Pa} \tag{16}$$

$$\begin{aligned}&DomainModel := (Concept, Hierarchy, Property,\\ &\quad Operation, Parameter, Relationship)\end{aligned} \tag{17}$$

B. Policy Modeling

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 specify ECA policies. It represents the abstract syntax of policies, i.e. it defines the structure of the policy model.

Definitions (18) to (39) represent an excerpt from the formal specification and illustrate how the policy model and its entities are defined.

$$Id_{Po}, Id_{Ev}, Id_{Cd}, Id_{Ac} \subseteq Val_T \tag{18}$$
$$Type_{Cd} := \{eq, gt, ge, lt, le, neq, and, or, op\} \tag{19}$$

$$Policy(Layer, Name, Active) \subseteq Id_{La} \times Id_{Po} \times Val_B \tag{20}$$
$$\begin{aligned}&(la_1, po, ac_1) \in Policy \wedge (la_2, po, ac_2) \in Policy\\ &\quad \Rightarrow ac_1 = ac_2\end{aligned} \tag{21}$$
$$Po := \{po | \exists la, ac.(la, po, ac) \in Policy\} \subseteq Id_{Po} \tag{22}$$

$$Event(Policy, Name) \subseteq Po \times Id_{Ev} \tag{23}$$
$$po \in Po \Rightarrow \exists ev.(po, ev) \in Event \tag{24}$$
$$Ev := \{ev | \exists po.(po, ev) \in Event\} \subseteq Id_{Ev} \tag{25}$$

$$\begin{aligned}&Condition(Policy, Name, Type, RefCond)\\ &\quad \subseteq Po \cup \{undef\} \times Id_{Cd} \times Type_{Cd} \times \mathcal{P}(Id_{Cd})\end{aligned} \tag{26}$$
$$\begin{aligned}&(po, cd_1, ty_1, rc_1) \in Condition\\ &\quad \wedge (po, cd_2, ty_2, rc_2) \in Condition\\ &\quad \Rightarrow cd_1 = cd_2 \wedge ty_1 = ty_2 \wedge rc_1 = rc_2\end{aligned} \tag{27}$$
$$\begin{aligned}&(po, cd, ty, rc) \in Condition \wedge ty \in \{eq, gt, ge, lt, le, op\}\\ &\quad \Rightarrow |rc| = 0\end{aligned} \tag{28}$$
$$\begin{aligned}&(po, cd, ty, rc) \in Condition \wedge ty \in \{neq\}\\ &\quad \Rightarrow |rc| = 1\end{aligned} \tag{29}$$
$$\begin{aligned}&(po, cd, ty, rc) \in Condition \wedge ty \in \{and, or\}\\ &\quad \Rightarrow |rc| = 2\end{aligned} \tag{30}$$
$$\begin{aligned}&(po_1, cd_1, ty_1, rc_1) \in Condition \wedge cd_2 \in rc_1\\ &\quad \Rightarrow \exists po_2, ty_2, rc_2.(po_2, cd_2, ty_2, rc_2) \in Condition\end{aligned} \tag{31}$$
$$Cd := \{cd | \exists po, ty, rc.(po, cd, ty, rc) \in Condition\} \subseteq Id_{Cd} \tag{32}$$
$$\begin{aligned}&Be := \{cd | \exists po, ty, rc.(po, cd, ty, rc) \in Condition\\ &\quad \wedge typeOfCondition(cd) \in \{eq, gt, ge, lt, le\}\} \subseteq Id_{Cd}\end{aligned} \tag{33}$$
$$\begin{aligned}&Oe := \{cd | \exists po, ty, rc.(po, cd, ty, rc) \in Condition\\ &\quad \wedge typeOfCondition(cd) \in \{op\}\} \subseteq Id_{Cd}\end{aligned} \tag{34}$$

$$Action(Policy, Name, No) \subseteq Po \times Id_{Ac} \times \mathbb{N} \tag{35}$$

$$po \in Po \Rightarrow \exists ac, no.(po, ac, no) \in Action \tag{36}$$

$$\begin{aligned}&(po, ac_1, no) \in Action \wedge (po, ac_2, no) \in Action\\ &\quad \Rightarrow ac_1 = ac_2\end{aligned} \tag{37}$$

$$Ac := \{ac | \exists po, no.(po, ac, no) \in Action\} \subseteq Id_{Ac} \tag{38}$$

$$PolicyModel := (Policy, Event, Condition, Action) \tag{39}$$

C. Domain-Specific Policy Modeling

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 model 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. It represents the abstract syntax of the links, i.e. it defines the structure of the linking model.

The relational algebra defines the structure of the linking model and additionally imposes some restrictions on its contents. One example is the usage of contextual information within a policy. Contextual 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 its concept is used as event of that policy. This restriction is covered by (40) to (42) in the relational algebra.

$$\begin{aligned}&arg \in argumentsOfCondition(cd) \Rightarrow\\ &\quad (arg \notin Properties \vee \forall po \in policiesOfCondition(cd).\\ &\quad arg \in visibleProperties(po))\end{aligned} \tag{40}$$

$$\begin{aligned}&arg \in argumentsOfAction(ac) \Rightarrow\\ &\quad (arg \notin Properties \vee \forall po \in policiesOfAction(ac).\\ &\quad arg \in visibleProperties(po))\end{aligned} \tag{41}$$

$$\begin{aligned}&visibleProperties : Id_{Po} \to \mathcal{P}(Id_{Pr})\\ &\quad po \mapsto \bigcup_{co \in visibleConcepts} propertiesOfConcept(co)\\ &\quad \text{with } visibleConcepts =\\ &\qquad \bigcup_{ev \in eventsOfPolicy(po)} conceptOfEvent(ev)\end{aligned} \tag{42}$$

D. Example

In a mobile network the signal quality of wireless connections between an antenna and cell phones is subject to frequent fluctuations due to position changes of cell phones and changing weather conditions. One possibility to react to fluctuating signal quality is adjusting the transmission power (TXP) at the antenna. In this example policies are used to manage the behavior of the communication system in an autonomous way. Policies e.g. adjust the transmission power of the antenna in order to ensure good signal quality and avoid unnecessary power consumption at the same time.

Figure 2 shows an example model and uses a graphical notation as concrete syntax for the model. An ECA policy *lowQuality* is triggered whenever the respective event indicates a change in the signal quality. The policy checks the details in its condition and if the signal quality falls below a critical value, increases the transmission power of the antenna with its action. An excerpt from the formal representation of the model is shown by (43) to (52).

$$Concept := \{(1, cellPhone), (1, signalQuality)\} \tag{43}$$

$$\begin{aligned}&Property := \{(cellPhone, cpImei),\\ &\quad (signalQuality, sqCellPhoneImei),\\ &\quad (signalQuality, sqOldValue),\\ &\quad (signalQuality, sqNewValue)\}\end{aligned} \tag{44}$$

$$Operation := \{(1, increasePower)\} \tag{45}$$

$$Policy := \{(1, lowQuality, true)\} \tag{46}$$

$$Event := \{(lowQuality, lqEvent)\} \tag{47}$$

$$Condition := \{(lowQuality, lqCondition, lt, \emptyset)\} \tag{48}$$

$$Action := \{(lowQuality, lqAction, 1)\} \tag{49}$$

$$EL := \{(lqEvent, signalQuality)\} \tag{50}$$

$$BEL := \{(lqCondition, sqNewValue, 50)\} \tag{51}$$

$$AL := \{(lqAction, (increasePower, \emptyset))\} \tag{52}$$

III. Related Work

Other approaches for the modeling of policies include GPML [11], the CIM Policy Model [12], and DEN-ng [3]. All of them use UML as concrete syntax to specify ECA policies in a language-independent way. Likewise, our approach includes a UML-based graphical notation as concrete syntax for the models, but additionally allows any concrete syntax in accordance with the abstract syntax of the models. Similar to our approach, GPML allows to adapt policies to different domains by defining a particular vocabulary, whereas the CIM Policy Model and DEN-ng are specifically targeted to the network management domain. DEN-ng is also based on the Policy Continuum and considers policies at different levels of abstraction, whereas GPML and the CIM Policy Model do not support different abstraction levels. GPML uses model transformations to map policies via an interchange format to existing policy languages, whereas the CIM Policy Model and DEN-ng do not address code generation. Our approach allows to generate code for any policy language that is able to express ECA policies as defined by the policy metamodel. None of the other approaches provide a formal specification of the models.

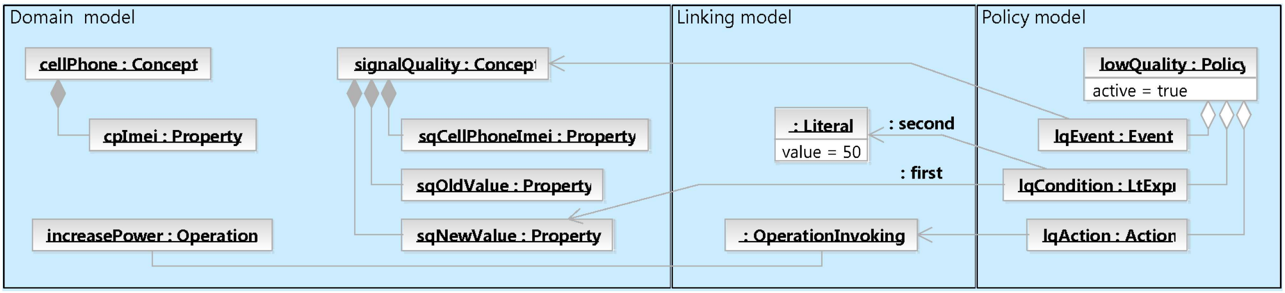

Figure 2. Model example

IV. Conclusion

A formal and model-based approach to the specification of ECA policies was presented in this paper. The usage of models allows to specify policies at a high level of abstraction initially, supports the refinement to lower levels, and thus avoids the direct implementation of policies at a technical level. A relational algebra precisely defines the abstract syntax of the models and allows for their validation.

Validation of the models is a prerequisite for their transformation into executable code. A prototype of a graphical policy editor that supports code generation for Ponder2 has already been developed [13]. The editor represents a policy model in a graphical way, so any policy can be visualized as a diagram. It offers functionality to visualize, create, and edit policy models in a comfortable way and is available under the GPL. A case study showed that a purely graphical syntax for the models might be confusing as diagrams take a lot of space in complex scenarios. An effective textual syntax is subject to future work. A representation of the models in the relational algebra for purposes of validation should be generated automatically.

In future, the refinement process should be automated with mapping patterns that replace higher-layer concepts with lower-layer ones and thus generate the refined model. The mapping patterns should be specified in the relational algebra to ensure the correctness of the refinement. The checking of properties and detection of inconsistencies within the set of policies by means of their formal specification are also subject to future work.

References

[1] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, “The Ponder Policy Specification Language,” in *2nd Workshop on Policies for Distributed Systems and Networks (POLICY)*. Springer LNCS, January 2001, pp. 18–38.

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

[3] J. Strassner, “DEN-ng: Achieving Business-Driven Network Management,” in *8th Network Operations and Management Symposium (NOMS)*. IEEE CS, April 2002, pp. 753–766.

[4] 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 (BcN)*, April 2006, pp. 1–10.

[5] T. Bandh, H. Sanneck, L.-C. Schmelz, and G. Carle, “Automated Real-time Performance Management in Mobile Networks,” in *1st WoWMoM Workshop on Autonomic Wireless AccesS (IWAS)*. IEEE CS, June 2007, pp. 1–7.

[6] 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 (IWCMC)*. ACM, June 2010, pp. 1111–1115.

[7] T. Bandh, R. Romeikat, and H. Sanneck, “Policy-based Coordination and Management of SON Functions,” in *12th International Symposium on Integrated Network Management (IM)*. IEEE ComSoc, May 2011, to be published.

[8] 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 (POLICY)*. IEEE CS, June 2008, pp. 245–246.

[9] R. Romeikat, B. Bauer, and H. Sanneck, “Modeling of Domain-Specific ECA Policies,” in *23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, July 2011, to be published.

[10] 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 (RuleML)*. Springer LNCS, November 2009, pp. 193–207.

[11] 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 (POLICY)*. IEEE CS, June 2008, pp. 101–104.

[12] Distributed Management Task Force, “CIM Policy Model White Paper,” DSP0108, June 2003.

[13] University of Augsburg, “PolicyModeler,” http://policymodeler.sf.net, August 2009.