# Automated Refinement of Policies for Network Management

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

Bernhard Bauer
*Institute of Computer Science*
*University of Augsburg*
*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 networks as policies make context-sensitive and automated decisions. For their effective development 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 abstraction layers and that uses model transformations to refine them in an automated way. A relational algebra is used to formally validate the models and define the semantics of the refinement process. One benefit of the approach is the automated policy refinement at runtime. Changes at the high-level models are automatically reflected in their low-level implementation through refinement. This allows to manage a system at a high level of abstraction. The approach is applied to the network management domain and demonstrated with policies for physical cell identification (PCI) in a mobile network. It can also be applied to other domains and supports a flexible number of abstraction layers.**

***Keywords*-policy-based management; model-driven engineering; network management**

## I. Introduction

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 allowing administrators to adapt the behavior of a system without changing source code or considering technical details. A system can continuously be adjusted to externally imposed constraints by changing the determining policies [1]. A well-known application domain is network management, where policies are widely used for automating network management processes. The usage of policy-based systems for the management of mobile networks was recently considered in [2]–[7].

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 like for example Ponder2 [8].

Policy-based management is a layered approach where policies exist at different levels of abstraction. The number of levels depends on the system to manage. For simple systems it might be sufficient to have one or two levels only, one with a business view and another one with a technical view. For large systems in a complex domain it is reasonable to introduce additional levels between the business and the technical one in order to represent the domain and the policies at intermediate levels of abstraction. A flexible number of abstraction levels is defined as the Policy Continuum [2]. Put down to the bottom line, policies are specified at each level in a domain-specific terminology and refined from a business level down to a technical level.

The different focus of the abstraction levels causes a semantic gap as business policies are decoupled from their technical implementation. The process of providing a lower-level representation of a higher-level policy is called policy refinement. During refinement the semantic gap between the different abstraction levels must be overcome, which is a non-trivial task. Refinement gets policies closer to a machine-executable representation from which executable code can be generated automatically as proposed in [9]. Due to the semantic gap domain experts usually perform policy refinement manually by passing policies from one level down to the next one and re-writing them with the means of the lower level. We present an approach for the automated refinement of ECA policies. It is based on different models at different abstraction layers in order separate domain and policy aspects from each other. Policies are specified at the highest layer initially and iteratively refined to the lower layers until they are represented in a machine-executable way. Automated refinement at runtime allows to control the system behavior by changing the high-level models.

This paper is structured as follows. The refinement of high-level policies is described in Section II. 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. Policy Refinement

We use different models at different abstraction layers in order to specify policies as described in [10] and illustrated in Figure 1. 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 specify links between the domain and policy model in order to use the domain-specific concepts within the policies. For each of them a metamodel exists that defines the abstract syntax 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 layers. The lowest layer finally represents the models such that executable policy code can be generated from them.

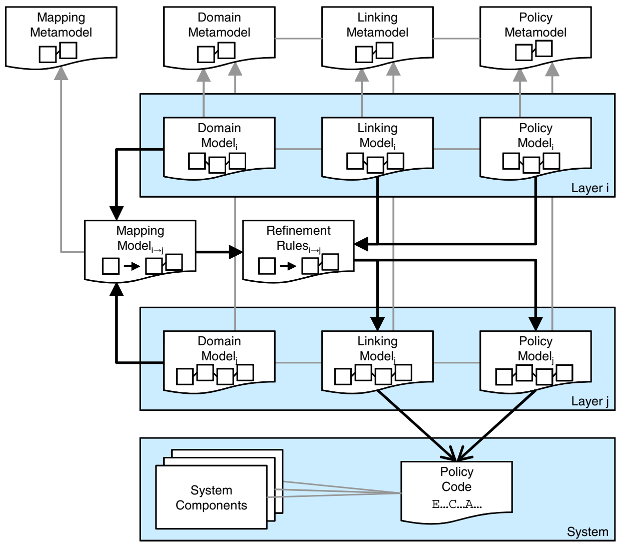

Figure 1. Policy refinement

Instead of refining high-level business policies manually, the refinement process is automated by specifying the refinement once within the domain and then applying this knowledge to the policies in that domain whereever and whenever necessary. For this purpose, domain experts specify in the *mapping model* how a higher-level representation of the domain model is refined into a lower-level one. This happens by establishing mappings between the domain-specific entities from a higher to a lower layer. Then, policy *refinement rules* are generated from the mapping model in an automated way. Finally, refined policies are generated in an automated way by applying the refinement rules to the linking and policy model. This happens in a top-down way across all layers, starting with the policies at the highest layer until policies are represented at the lowest layer with a technical point-of-view. We developed a relational algebra to formally define the models and the semantics of the refinement process. The algebra is used to validate whether model instances conform to the respective metamodel and to prove that refined models are again valid instances of the metamodels.

## A. Mapping Model

Any relevant information about the domain refinement is covered by the mapping model. Refinement of the domain means mapping its representation from a higher layer to a more detailled representation at lower layers in order to successively extend domain knowledge. The possible structural changes through refinement are expressed by a set of mapping patterns. These patterns define how the lower-level representation of entities is derived from their higher-level representation. The available mapping patterns are called *identity*, *replacement*, *merge*, *split*, *erasure*, and *appearance* and are illustrated in Figure 2.

- The identity pattern leaves a higher-level entity unchanged at the lower level, so $e$ at the higher layer $i$ has exactly the same representation at the lower layer $j$.
- The replacement pattern maps a higher-level entity to a lower-level one. The entity $e_1$ at layer $i$ is represented by the entity $e_2$ at layer $j$.
- The merge pattern maps multiple higher-level entities to one lower-level entity. The entities $e_{1_1}$ to $e_{1_n}$ at layer $i$ are represented by the entity $e_2$ at layer $j$.
- The split pattern maps a higher-level entity to one out of some lower-level entities. The entity $e_1$ at layer $i$ is represented by one out of the entities $e_{2_1}$ to $e_{2_n}$ at layer $j$. This pattern represents a choice between different mapping options.
- It may be the case that a higher-level entity is no more relevant at a lower level. For this purpose the erasure pattern maps an entity $e_1$ at layer $i$ to no entity at layer $j$, which means that entity does not have a lower-level representation.
- The other way round it may be the case that an entity $e_2$ at layer $j$ does not have a representation at layer $i$ and does not depend on other entities. This case is represented by the appearance pattern.

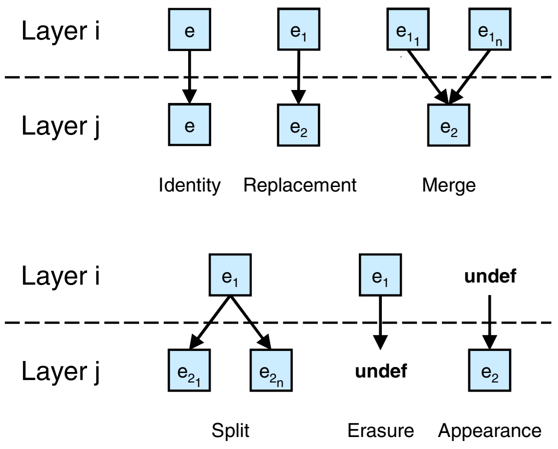

Figure 2. Mapping patterns

A concrete refinement of the domain from a higher to a lower layer is established by instantiating the refinement patterns with the entities of the domain model. An instantiated pattern is simply called *mapping*. The mapping model specifies the mappings to refine the domain model from a higher layer to a lower layer and represents the basis for the refinement of policies in that domain.

## B. Generation of Refinement Rules

The next step is the generation of refinement rules from the mapping model. This step is performed once at design time after the set of mappings has been specified or modified. Due to the different semantics of events, conditions, and actions, the impact of a mapping on a policy depends from whether the entities of that mapping appear in the event, conditon, or action part of the policy. Therefore, a mapping results in different refinement rules for the refinement of the event, condition, and action parts. Refinement rules generated from the appearance pattern are very useful as they automatically integrate information into the policies at a lower layer that was not specified at a higher layer. An example are additional parameters that appear in a refined operation. The appearance pattern allows to pass paricular values to these parameters depending on the context of the higher-level operation.

Refinement rules describe a model-to-model (M2M) transformation in a formal and language-independent way. A refinement rule is the smallest unit of transformation and transforms links between the domain and policy model from a higher layer into a lower layer. It represents a declarative rewrite rule whose left hand side (LHS) represents the input and right hand side (RHS) represents the output of the transformation. The transformation extends the linking model with the refined links. The refined links refer to refined policies, which do not yet exist in the policy model, thus the extension of the linking model also implies an extension of the policy model. The extension of the policy model needs not be specified explicitly as it follows implicitly from the formal specification of the models.

## C. Generation of Refined Policies

Finally, refined policies are generated in an automated way by applying the refinement rules to the linking and policy model. This step is performed at design time and runtime whenever a policy is added, modified, or deleted. The execution semantics of the refinement process in an operational point-of-view is illustrated by Algorithm 1.

**Algorithm 1** Generation of refined policies

```
procedure REFINEMODEL
  for $layer \leftarrow minLayer + 1$ to $maxLayer$ do
    DELETELAYER($layer$)
  end for
  for $layer \leftarrow minLayer$ to $maxLayer - 1$ do
    REFINELAYER($layer$)
  end for
end procedure
procedure REFINELAYER($layer$)
  for $rule \leftarrow$ RULES($layer$), $link \leftarrow$ LINKS($layer$) do
    if MATCH($rule, link$) then
      $refinedLink \leftarrow$ REFINELINK($rule, link$)
      ADD($layer + 1, refinedLink$)
    end if
  end for
end procedure
procedure REFINELINK($rule, link$)
  $refinedLinks \leftarrow$ APPLY($rule, link$)
  if SIZE($refinedLinks$) = 1 then
    $refinedLink \leftarrow$ FIRST($refinedLinks$)
  else
    $refinedLink \leftarrow$ DECIDE($refinedLinks$)
  end if
  return $refinedLink$
end procedure
```

- **procedure** REFINEMODEL()
  In the beginning all layers of the linking and policy model are deleted except for the highest layer. The algorithm then iterates over the layers and in each iteration refines the policies of the current layer into policies at the subsequent lower layer as intermediate result. It starts with the highest layer and stops after the second lowest layer has been refined, resulting in the policies at the lowest layer.
- **procedure** REFINELAYER($layer$)
  To refine a layer into the subsequent lower one, the algorithm filters the rules and links of the current layer and iterates over all pairs of them. The sequence of iteration is arbitrary within a layer as any sequence leads to the same result. This allows for a high parallelization of the algorithm. If a match is found for a rule and a link, the refined link is generated and added to the subsequent lower layer.
- **procedure** REFINELINK($rule, link$)
  To refine a link with a refinement rule, the algorithm applies the rule to the link according to the LHS of the rule and generates the refined link according to the RHS of the rule. Refinement rules generated from split mappings are non-deterministic and imply multiple possibilities for the refined link. They demand a decision for one out of these possibilities. This feature takes account of cases where a definite refinement is not possible initially and allows to delay the decision until the refinement is performed.

## D. Code Generation

After refinement to the lowest layer the policies are represented in a machine-executable way. The domain model then represents the concepts of the underlying system components and the linking and policy model represents policies to control 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 first involves a model-to-model transformation, which transforms the linking and policy model into an intermediate model-based representation of the target language. Then, a model-to-text transformation generates executable policy code in the target language. Code generation is realized as proof of concept for Ponder2. Details of the model transformations are presented in [9].

## III. Case Study

Operation, administration, and maintenance (OAM) of a mobile network is a complex task due to the distributed architecture of the underlying cellular network. Complexity arises from a high number of network elements (NEs) to be deployed and managed and from interdependencies between their configurations. Management tasks typically require high expertise and are performed by human operators, who use their operational experience to find optimized configurations. This requires a lot of human interaction and manual control is time-consuming, expensive, and error-prone.

### A. Physical Cell ID Assignment

The Physical Cell ID (PCI) is a fundamental parameter within Long Term Evolution (LTE) radio networks. It is used as a regionally unique identifier on the physical layer and plays an essential role for enabling radio communication and handover handling. The automated configuration of PCIs is one of the key use cases in Next Generation Mobile Networks (NGMNs) [11].

In an LTE network only 504 different PCIs are available and the PCI range is fragmented to handle different cell types, assignment at country or license borders, and temporary assignment. As there are usually a lot more than 504 cells, PCIs must be reused within the network [12]. Furthermore, there are constraints on the assignment of PCIs to neighboring cells, i.e. cells with a common coverage area:

- Collision-free: Any two neighboring cells must be assigned different PCIs.
- Confusion-free: Any neighbors of a cell must be assigned different PCIs.

Variations of radio propagation properties and changes of the network topology require frequent PCI reconfigurations. As the reconfiguration of a PCI may require a restart of the respective eNodeB and thus cause a service interruption, it is important to provide an assignment that proactively reduces the number of reconfigurations. An operator can have different strategies to assign PCIs in the network. In order to carefully deal with PCIs as a limited resource, the *maxReuse* strategy can be applied. This strategy uses as few different PCIs as possible and assigns the same PCIs repeatedly. In contrast, other assignment strategies assign PCIs in a way that any two cells with the same PCI have a maximal distance from each other.

### B. Policy Refinement

In order to cope with the described requirements and to optimize the PCI assignment under the objectives of the operator, we developed an approach to apply the appropriate assignment strategy in an automated way. The decision for a strategy depends from the context, i.e. the reason for the assignment and the current network configuration. The decision logic is modeled and enforced with a set of ECA policies. There is one policy for each strategy and these policies are represented at two abstraction layers. The higher layer represents policies with a management point-of-view hiding technical details from the operator. The lower layer represents policies with a technical point-of-view and in a machine-executable way for the underlying management system.

For this purpose, a domain model is specified that covers the relevant concepts for PCI assignment. This happens at the higher layer initially. At the same layer the policies are specified as a policy model and they are linked to the domain model with a linking model. Figure 3 shows an ECA policy that is triggered whenever a generic *PCIRequest* occurs. The policy sets the assignment strategy to *maxReuse* if the target cell is located in a *sparse area* or only a *partial PCI range* is available. In both cases it makes sense to assign as few different PCIs as possible as reconfigurations are unlikely or only few PCIs are available at all. A simplified graphical notation with UML models and textual annotations is used as concrete syntax in the figure.

In order to enable a technical view, a refined domain model is specified at the lower layer. A mapping model is specified to map the domain model from the higher to the lower layer and the respective refinement rules are generated from the mapping model. (1) shows a split mapping in the relational algebra that is used to refine the high-level concept *PCIRequest* into one out of the four low-level concepts *NewCell*, *NewNeighbor*, *ChangedCellSize*, or *PCIConflict*. The respective refinement rule is shown in (2). When a policy is refined that links its event with *PCIRequest*, the operator is offered the four possibilities for the refined event link and asked for a decision.

$$\begin{aligned} &PCIRequest \xrightarrow[1\to 2]{Co} (NewCell, NewNeighbor, \\ &\quad PCIConflict, ChangedCellSize) \end{aligned} \tag{1}$$

$$\begin{aligned} &\forall ev_1.(ev_1, PCIRequest) \in EL_1 \\ &\quad \Rightarrow \exists ev_2.(ev_2, NewCell) \in EL_2 \\ &\qquad \vee (ev_2, NewNeighbor) \in EL_2 \\ &\qquad \vee (ev_2, PCIConflict) \in EL_2 \\ &\qquad \vee (ev_2, ChangedCellSize) \in EL_2 \end{aligned} \tag{2}$$

The refined policies are now generated in an automated way. For this purpose, the refinement rules are applied to the high-level linking and policy model and generate the low-level linking and policy model. Figure 4 shows the refined ECA policy. In case of refinement rule (2) the operator decided to refine the concept *PCIRequest* within the event into the concept *NewCell*. Consequently, the refinement algorithm links the refined event with the refined concept. Due to other refinement rules, the refined condition and action are linked to the respective refined operations. Also, some management terms are refined into technical ones. A *sparse area* is refined into a location with *less than five neighbors*, a *partial PCI range* into one with *less than 252 PCIs*, and the *maxReuse strategy* into a *safety margin of 2*. The safety margin ensures that the PCI of a cell is not assigned at neighboring cells of a certain degree and a safety margin of 2 is the minimal setting to avoid PCI collisions and confusions [13].

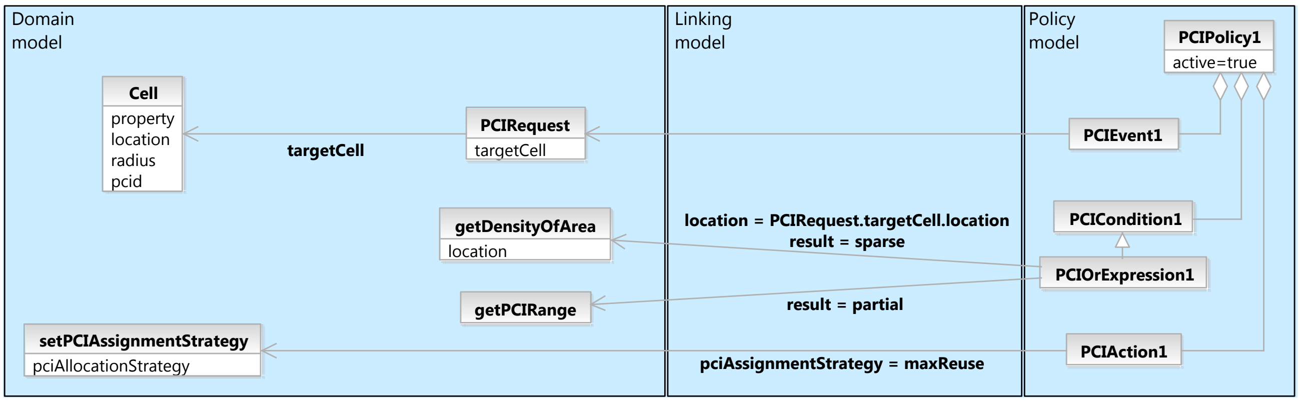

Figure 3. High-level model (excerpt)

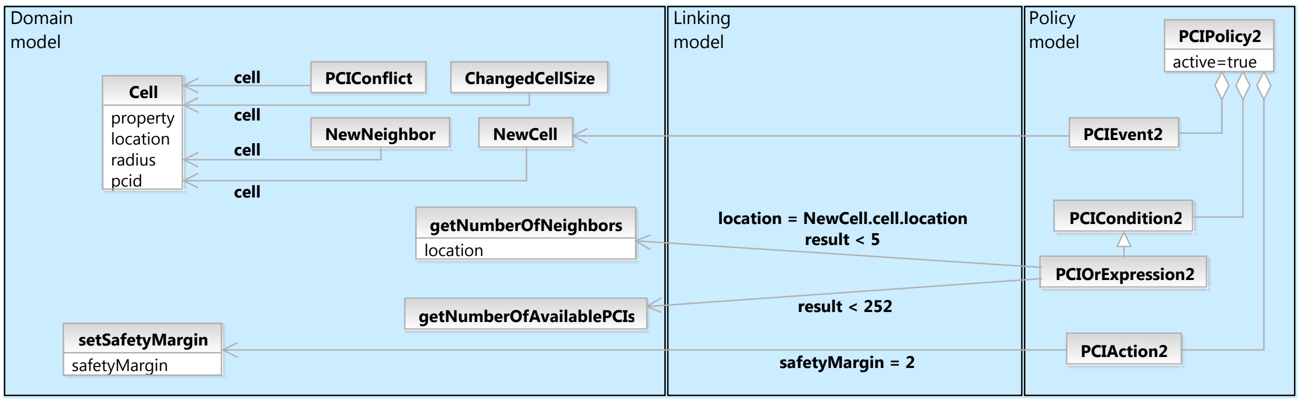

Figure 4. Low-level model (excerpt)

Finally, the refined models are automatically transformed into the executable Ponder2 code shown in Listing 1. The respective model transformations are described in [9]. The resulting code is directly used in the underlying management system [14].

```
policy := root/factory/ecapolicy create.
policy event: root/event/NewCell;
       condition: [ :NewCell.cell.location | root
           /op getNumberOfNeighbors:NewCell.cell
           .location < 5 | root/op
           getNumberOfAvailablePCIs < 252 ];
       action: [ root/op setSafetyMargin:2 ].
root/policy at:"PCIPolicy2" put:policy.
policy active: true.
```

Listing 1. Generated Ponder2 code (excerpt)

## IV. Related Work

A refinement approach that focuses on policies in the autonomic networking domain is presented in [4]. Policies represent configuration settings and are used for automated network and resource configuration. A fixed terminology is used to specify policies at five levels of a Policy Continuum and each level offers a sub-set of that terminology. Policies are automatically refined into configuration commands on a per-device basis. Formal semantics is not provided.

In [15] high-level goal policies are refined into low-level policy actions to achieve them. Goals are represented at different levels of abstraction. The approach realizes a semi-automated refinement. High-level goals are first decomposed into logically equivalent sub-goals in a manual process supported by decomposition patterns. A reasoning mechanism then automatically infers system operations from the decomposed goals. The approach formalized in Event Calculus.

The authors of [16] use two abstraction levels and automatically refine high-level requirements on the behavior of a system into low-level constraint policies. The approach is based on simulation and classification techniques that relate measured data to high-level requirements and can be applied to any system that generates measurable data. The refinement determines for which combinations of system attribute values the requirements are fulfilled. Formal semantics is not provided.

In [17] high-level policies are automatically refined into low-level ones in order to better detect conflicts between the high-level ones. The refinement algorithm analyzes constraints about policy subjects, actions, and targets from an information model to assess more precisely under which circumstances the actions must or must not be invoked. These circumstances are added to the policy as condition clauses. No formal semantics is provided.

## V. Conclusion

The way policy development is performed today is contrary to the paradigms of modern software development as business policies are decoupled from their implementation, which imposes additional effort. As a solution to this issue an approach for the automated refinement of ECA policies was presented in this paper and applied to policies for PCI assignment in a mobile network.

The usage of models allows to specify policies at a high level of abstraction initially and avoids the direct implementation of policies at a technical level. Models do not only serve specification or documentation purposes, but are essential artifacts of the policy development process. Their refinement into a machine-executable representation allows to control the system behavior at runtime by changing the high-level models. Policies are refined in an automated way with refinement rules generated from mapping patterns. The degree of automation highly depends on an effective specification of the domain and the mappings. The approach also respects manual interaction, which can often not be avoided completely. However, manual steps as reduced as far as possible and whenever a refinement step cannot be performed automatically the possible solutions are proposed.

Tool support is subject to future work. A prototype of a graphical policy editor that supports code generation has already been developed [18]. The editor represents a policy model in a graphical way as a diagram and offers functionality to create and change policy models. Executable code for Ponder2 can be generated from the policy model in a fully automated way. In future, the editor will be extended to support multiple abstraction layers and trigger the refinement process after a policy was created or changed. Policies should be represented from their high-level models to their low-level implementation in one single tool. The case study showed that a graphical concrete syntax for the models is inappropriate as diagrams take a lot of space in complex scenarios. An effective textual concrete syntax is 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, 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 Press, 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, pp. 823–836.

[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, 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.

[10] 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, pp. 52–58.

[11] Next Generation Mobile Networks (NGMN) Alliance, "Use Cases related to Self Organising Network, Overall Description," Deliverable, May 2007.

[12] 3rd Generation Partnership Project (3GPP), "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall Description; Stage 2 (Release 10)," Technical Specification 36.300, June 2011.

[13] T. Bandh, G. Carle, H. Sanneck, L.-C. Schmelz, R. Romeikat, and B. Bauer, "Optimized Network Configuration Parameter Assignment Based on Graph Coloring," in *12th Network Operations and Management Symposium (NOMS)*. IEEE ComSoc, April 2010, pp. 40–47.

[14] T. Bandh, H. Sanneck, and R. Romeikat, "An Experimental System for SON Function Coordination," in *International Workshop on Self-Organizing Networks (IWSON)*. IEEE VTS, May 2011, pp. 1–2.

[15] A. K. Bandara, E. C. Lupu, J. Moffett, and A. Russo, "A Goal-based Approach to Policy Refinement," in *5th International Workshop on Policies for Distributed Systems and Networks (POLICY)*. IEEE CS, June 2004, pp. 229–239.

[16] Y. B. Udupi, A. Sahai, and S. Singhal, "A Classification-Based Approach to Policy Refinement," in *10th International Symposium on Integrated Network Management (IM)*. IEEE CS, May 2007, pp. 785–788.

[17] S. Davy, B. Jennings, and J. Strassner, "Conflict Prevention via Model-driven Policy Refinement," in *17th International Workshop on Distributed Systems: Operations and Management (DSOM)*. Springer LNCS, October 2006, pp. 209–220.

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