Specification and Refinement of Domain-Specific ECA Policies

Raphael Romeikat and Bernhard Bauer

University of Augsburg, Institute of Computer Science, Augsburg, Germany
{romeikat,bauer}@ds-lab.org

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 for the specification and the automated refinement of domain-specific event-condition-action (ECA) policies. Domain-specific policies use domain-specific concepts within their event, condition, and action parts. 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 policies for signal quality management in a mobile network.

1 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 as it allows 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 area is network management, where policies are widely used for performing configuration 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 as for example Ponder2 [8].

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 [2]. The idea is to specify and manage policies at each level in a domain-specific terminology, and to refine them from a business level down to a technical level.

The process of providing a lower-level representation of a higher-level policy is called policy refinement. Policy refinement is a non-trivial task as different abstraction levels must be passed. Refinement is usually performed manually by passing policies from one level down to the next one and re-writing them with the means of the lower level. To address this we present a generic approach for the specification and automated refinement of domain-specific ECA policies. Automated refinement at runtime allows to control the actual system behavior by changing high-level models instead of their implementation.

This paper is structured as follows. Section 2 provides a scenario from the network management domain that is used as running example. Section 3 describes how domain-specific policies are modeled at a high level. The refinement of those policies is described in section 4. Related work is discussed in section 5. The paper concludes with a summary and future work in section 6.

2 Example Scenario

The signal quality of wireless connections in a communication system is subject to frequent fluctuations. There are various reasons for such fluctuations such as position changes of cell phones or changing weather conditions which impact the transmission. One possibility to react to fluctuating signal quality is adjusting transmission power (TXP) as power proportionally influences signal quality between an antenna and a cell phone.

The scenario now raises two objectives, amongst others. On the one hand, transmission power should be rather high in order to ensure good signal quality and to avoid connection losses. On the other hand, transmission power should be rather low in order to avoid unnecessary power consumption. As transmission power also influences the coverage area of the cells, a too high or too low setting can result in an undesired state of the network [7]. Therefore, a good tradeoff between transmission power and signal quality is desired.

In order to manage the behavior of the communication system we introduce a policy-based approach. Two ECA policies *lowQuality* and *highQuality* are responsible for adjusting the transmission power of the antenna. From a conceptual point of view the transmission power of an antenna can be increased and decreased. Changes in the signal quality are indicated by an event that contains the International Mobile Equipment Identity (IMEI) as unique identifier of the respective cell phone and the old and new value of the signal quality. The two policies are triggered whenever that event occurs and in their conditions they

check the values of signal quality enclosed in the event. If the signal quality falls below a critical value, the *lowQuality* policy increases the transmission power of the antenna. The other way round, the *highQuality* policy decrease the transmission power if the signal quality goes beyond a critical value.

The behavior of the communication system can then be adjusted at runtime via the policies. The accepted range of signal quality between the two critical values is specified in the policies conditions. Changing those critical values is possible at any time and has an immediate effect on the transmission power and signal quality.

3 Policy Specification

Different 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 policy and domain experts to link the policy model to the domain model in order to use the domain-specific concepts within the policies. Those models can also be regarded as 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.

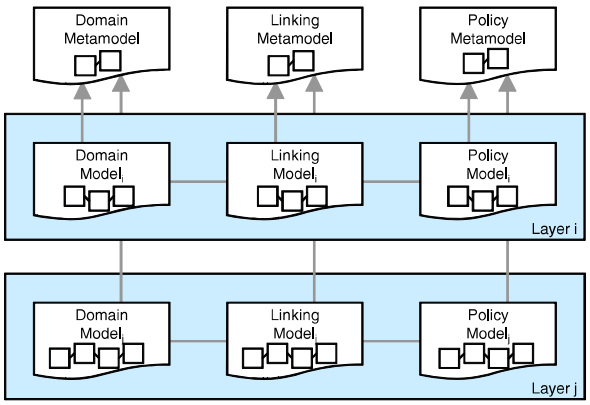

Fig. 1. Policy specification

3.1 Domain Modeling

Different expert groups are involved in the management of a 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 a 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. Figure 3 shows the domain model of the example scenario with two layers. 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.

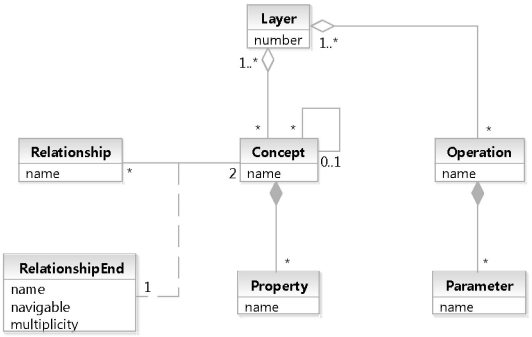

Fig. 2. Domain metamodel

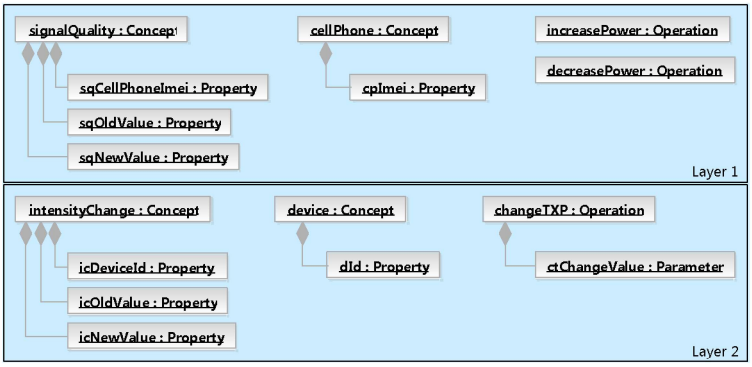

Fig. 3. Domain model

3.2 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 a different terminology to express a policy than a

system administrator does 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. Figure 5 shows an excerpt of the policy model at the first layer of the example scenario. The policy model is an instance of the *policy metamodel*, which contains the essential aspects required to specify basic ECA policies. It is shown in figure 4 and represents the abstract syntax of policies, i.e. it defines the structure of the policy model.

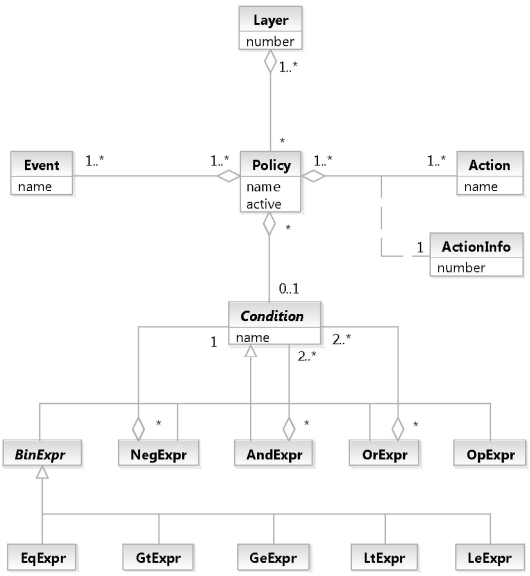

Fig. 4. Policy metamodel

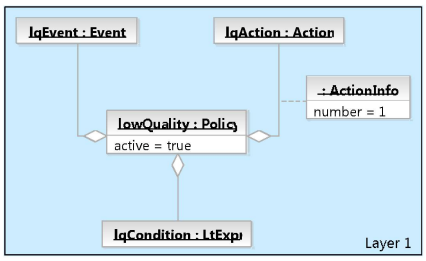

Fig. 5. Policy model (excerpt)

3.3 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 refer to the domain within the policies. For this purpose, a third 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 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. Figure 7 shows an excerpt of the linking model at the first layer of the example scenario. 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 6. It represents the abstract syntax of the links, i.e. it defines the structure of the linking model.

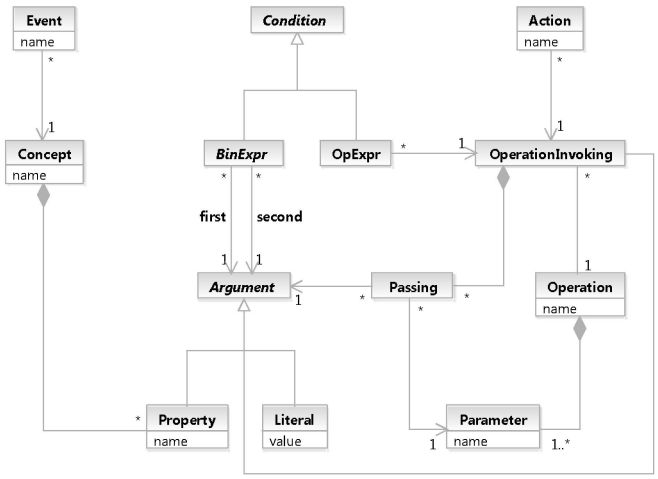

Fig. 6. Linking Metamodel

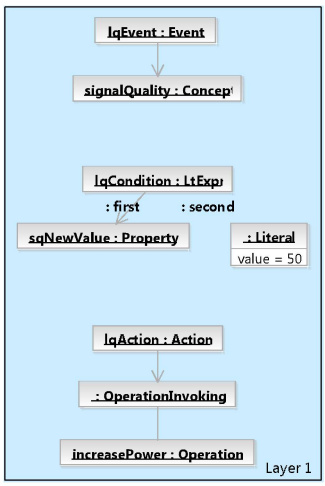

Fig. 7. Linking model (excerpt)

4 Policy Refinement

The formalization of domain-specific knowledge within the domain model allows to formalize the refinement of the domain from a higher to a lower layer as illustrated in figure 8. Refinement of the domain is the basis for the automated generation of refined policies in that domain.

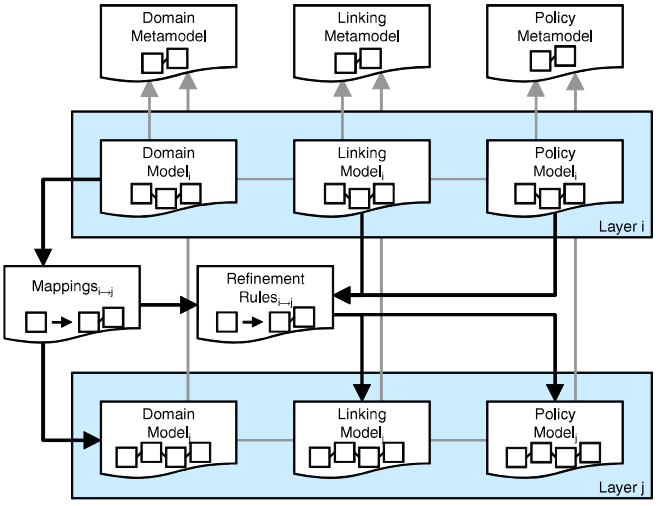

Fig. 8. Policy refinement

When going down the abstraction layers the representation of a higher-layer entity can remain the same at a lower layer or it can change completely. Refinement of the domain means mapping its representation from a higher layer to another representation at a lower layer. The possible structural changes through refinement are expressed by a set of mapping patterns. These patterns specify how the lower-layer representation of entities is derived from their higher-layer

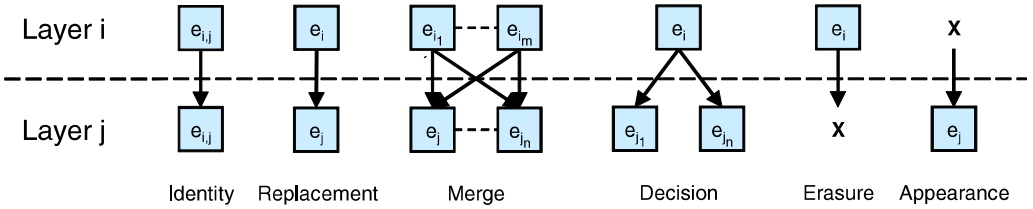

Fig. 9. Mapping patterns

representation. The available patterns are called *identity*, *replacement*, *merge*, *decision*, *erasure*, and *appearance* and are illustrated in figure 9.

A concrete mapping of one layer to another one is established by instantiation of the refinement patterns with the relevant entities at the respective layers. An instantiated pattern is simply called *mapping*. Figure 10 shows the mappings to refine the *lowQuality* policy of the example scenario in a textual syntax. The mappings once define refinement information within the domain and are then used for the automated refinement of policies in that domain.

$$signalQuality \mapsto intensityChange \text{ (replacement)}$$
$$sqNewValue \mapsto icNewValue \text{ (replacement)}$$
$$increasePower \mapsto changeTXP \text{ (replacement)}$$
$$undef \mapsto ctChangeValue \text{ (appearance)}$$
$$increasePower() \mapsto changeTXP(0.5) \text{ (merge)}$$

Fig. 10. Mappings (excerpt)

The generation of refined policies is divided into two parts as illustrated in figure 8. First, refinement rules are generated from the mappings. This part is only performed once after the set of mappings has been specified or modified. A refinement rule from layer i to layer j represents a model transformation that takes a policy and a linking model from layer i as input and produces a refined policy and a refined linking model at layer j as output. The left hand side (LHS) represents the input and the right hand side (RHS) represents the output of the transformation. Due to the different semantics of events, conditions, and actions, the impact of a mapping on a domain-specific policy depends from whether the entities of that mapping appear in the event, conditon, or action part of the policy. Therefore, one mapping results in three different refinement rules, one processing the event part, one the condition part, and one the action part.

Second, refined policies are generated by applying the refinement rules to the policy and the linking model. This part is be performed every time when a policy is added, modified, or deleted. In order to generate refined policies the refinement rules are applied to the policy and the linking model in a particular sequence. That sequence is determined by a pattern matching algorithm. The algorithm starts with the highest layer and generates the refined policies at one layer below. For any refinement rule that applies to those layers it matches the LHS to any

combination of entities in the policy and linking model of the highest layer. Whenever a match is found, it invokes the matching refinement rule, which then generates the refined policy according to its RHS. This process is repeated until no more refinement rule can be applied to the highest layer. The intermediate result is a refined policy and linking model at one layer below. As the algorithm works on a copy of the policies, the policies of the highest layer remain and the refined policies are added to the models. The algorithm then processes the subsequent lower layers one after another in their ordering of abstraction and finally produces a refined policy and linking model at the lowest abstraction layer.

Figures 11 and 12 show an excerpt of the refined policy and linking models of the example scenario. During refinement the policy model remaines the same as the structure of the policy is not affected by the mappings. However, the linking model is automatically changed and the policies are now linked to the domain entities of the lower layer. The event of the *lowQuality* policy is no more linked to the *signalQuality* concept, but to *intensityChange* one. The policy condition is no more linked to the *sqNewValue* property, but to the *icNewValue* one. These changes are caused by the respective replacement mappings. The policy action is no more linked to the *increasePower* operation, but to the *changeTXP* one and a literal value is passed to the *ctChangeValue* parameter of that operation. This change is caused by the respective appearance and merge mappings.

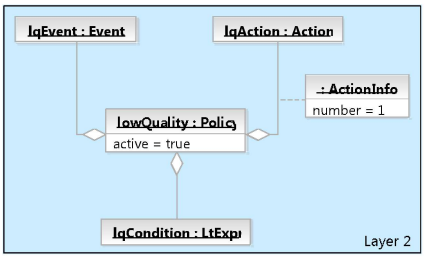

Fig. 11. Policy model (excerpt)

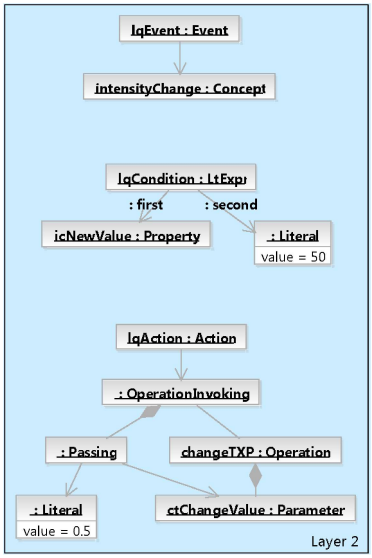

Fig. 12. Linking model (excerpt)

5 Related Work

The authors of [9] 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.

The CIM Policy Model [10] by the Distributed Management Task Force (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.

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 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.

6 Conclusion

A domain-specific approach for the specification and refinement of ECA policies was presented in this paper. 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. High-level policies are made executable through refinement into a machine-executable representation. This allows to control the actual system behavior at runtime by changing the high-level models. The approach is novel as it is generic with respect to the domain, to the language, and to the number of abstraction levels and is nevertheless fully automated. No working solution has been known yet that realizes policy refinement in an automated way and that is not specifically tailored to a particular problem or domain.

Models in the approach do not only serve specification or documentation purposes, but are essential artifacts of the policy development process that initially starts with non-executable business policies and results in their technical and executable representation. The separation of knowledge into different models or model parts allows for an effective collaboration of domain and policy experts. The usage of different abstraction layers faciliates the collaboration of business and technical experts. Mapping patterns allow to generate refined ECA policies in an automated way and generation of refined policies helps to consolidate development effort. The approach is generic as it can be applied to any domain and any number of abstraction levels. It is also extensible and e.g. allows to integrate additional mapping patterns in order to cover dependencies that cannot be addressed with the provided patterns.

Tool support is subject to future work. A prototype of a graphical policy editor has already been developed [11, 12] and is to be developed further. This involves a modeling the domain, specifying the mappings, and modeling policies in that domain. The editor should trigger the policy refinement process after a policy was changed and generate code for existing policy languages.

References

1. Damianou, N., Dulay, N., Lupu, E.C., Sloman, M.: The Ponder Policy Specification Language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
2. Strassner, J.: Policy-Based Network Management: Solutions for the Next Generation. Morgan Kaufmann Publishers, San Francisco (2003)
3. Strassner, J.: DEN-ng: Achieving Business-Driven Network Management. In: 8th Network Operations and Management Symposium, pp. 753–766. IEEE CS, Los Alamitos (2002)
4. van der Meer, S., Davy, A., Davy, S., Carroll, R., Jennings, B., Strassner, J.: Autonomic Networking: Prototype Implementation of the Policy Continuum. In: 1st International Workshop on Broadband Convergence Networks, pp. 1–10 (April 2006)
5. Bandh, T., Sanneck, H., Schmelz, L.C., Carle, G.: Automated Real-time Performance Management in Mobile Networks. In: 1st WoWMoM Workshop on Autonomic Wireless Access, pp. 1–7. IEEE CS, Los Alamitos (2007)
6. Romeikat, R., Bauer, B., Bandh, T., Carle, G., Sanneck, H., Schmelz, L.-C.: Policy-driven Workflows for Mobile Network Management Automation. In: 6th International Wireless Communications and Mobile Computing Conference, pp. 1111–1115. ACM, New York (2010)
7. Bandh, T., Romeikat, R., Sanneck, H.: Policy-based Coordination and Management of SON Functions. In: 12th International Symposium on Integrated Network Management (May 2011) (to be published)
8. Twidle, K., Lupu, E., Dulay, N., Sloman, M.: Ponder2 - A Policy Environment for Autonomous Pervasive Systems. In: 9th Workshop on Policies for Distributed Systems and Networks, pp. 245–246. IEEE CS, Los Alamitos (2008)
9. Kaviani, N., Gasevic, D., Milanovic, M., Hatala, M., Mohabbati, B.: Model-Driven Engineering of a General Policy Modeling Language. In: 9th Workshop on Policies for Distributed Systems and Networks, pp. 101–104. IEEE CS, Los Alamitos (2008)
10. Distributed Management Task Force: CIM Policy Model White Paper. DSP0108 (June 2003)
11. University of Augsburg: PolicyModeler (August 2009), `http://policymodeler.sf.net`
12. Romeikat, R., Sinsel, M., Bauer, B.: Transformation of Graphical ECA Policies into Executable PonderTalk Code. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 193–207. Springer, Heidelberg (2009)