Embedding Policy Rules for Software-Based Systems in a Requirements Context

Mark Strembeck
Department of Information Systems, New Media Lab
Vienna University of Economics and BA, Austria
mark.strembeck@wu-wien.ac.at

Abstract

Policy rules define what behavior is desired in a software-based system, they do not describe the corresponding action and event sequences that actually "produce" desired ("legal") or undesired ("illegal") behavior. Therefore, policy rules alone are not sufficient to model every (behavioral) aspect of an information system. In other words, like requirements policies only exist in context*, and a policy rule set can only be assessed and sensibly interpreted with adequate knowledge of its embedding context. Scenarios and goals are artifacts used in requirements engineering and system design to model different facets of software systems. With respect to policy rules, scenarios are well suited to define how these rules are embedded into a specific environment. A goal is an objective that the system under consideration should or must achieve. Thus, the control objectives of a system must be reflected in the policy rules that actually govern a system's behavior.*

1 Introduction and Motivation

Information systems are software-based systems that assist human-users in the execution of complex tasks and support the business processes of an organization. Business processes, in turn, are performed to reach the operational goals of the corresponding organization. *Scenarios* describe action and event sequences and make process descriptions explicit. They are used in different research areas like human-computer interaction, strategic management, and software engineering for example (see, e.g., [5]). Operational goals and scenarios are derived from long-term strategic and mid-term tactical goals and scenarios. Operational scenarios define standardized action and event sequences and therefore describe the routine business processes of an organization which are performed to meet the corresponding operational goals.

Policies are rules governing the choices in behavior of a system [10]. They exist on different levels of abstraction and provide a means to synchronize software-based systems with the operational goals and scenarios of a specific organization (as indicated by Figure 1). A *policy rule set* consists of several related or interdependent rules, and a software system may be controlled by various policy rule sets. In this paper, we see

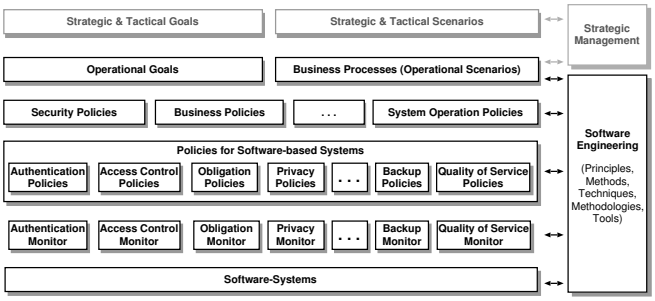

Figure 1. Policies synchronize software systems with operational goals and scenarios

requirements and policy rules as interrelated artifacts to describe *complementary perspectives* of software-based systems. In particular, we present an approach to model the embedding context of policy rules. On the one hand, scenarios and goals serve as a means to understand the objectives that should be achieved via a set of policy rules (and from a requirements perspective policy rules can be seen as a form of solution models). On the other hand, requirements engineering techniques allow for a systematic engineering of policy rules, and, to a certain degree, enable to check a set of policy rules for completeness with regard to the goals and scenarios that model the requirements of a software-based system. Both purposes rely on traceability links (see, e.g., [8]) between modeling level (requirements) artifacts and the corresponding policy rules.

2 Scenarios and Goals

Scenarios are tools for understanding (cognitive aspect), tools for elicitation and specification (engineering aspect), and tools for communication (social aspect). In the area of software engineering, scenarios are used to explore and to describe the (actual or intended) system behavior as well as to specify user needs. Scenarios can be described in many different ways. Commonly, they are specified with (structured) text descriptions and with different types of diagrams, e.g. message sequence charts, activity diagrams, or petri-nets. Scenarios may also depict system internal activities that are not (directly) visible to the user.

Moreover, scenarios may also be applied to describe what should *not* happen (see, e.g., [1]), and they can easily be integrated with goals (see, e.g., [9]). Goals are, like scenarios, a familiar concept in the area of requirements engineering (see, e.g., [12]). They are well-suited to be applied in combination with scenarios to elicit and define requirements and to drive a requirements engineering process (see, e.g., [5, 9]). In general, a *goal* is an objective that the system under consideration should or must achieve. Goals can be defined on different levels of abstraction, ranging from high-level business goals to low-level technical concerns. They can be arranged in a directed graph to form goal hierarchies. Goals can model functional as well as non-functional aspects (e.g. performance). In turn, scenarios can be applied to describe alternative ways to reach a goal. Moreover, each step within a scenario is likely to be associated with a step-goal, and a step-goal then is a natural sub-goal of the (super-)goal linked to the corresponding scenario. An *obstacle* is an undesired condition which obstructs the fulfillment of one or more goals. Thus, obstacles can be seen as the opposite of goals. In the area of requirements engineering, obstacles are a valuable means to define more complete and realistic requirements (see, e.g., [13]).

Figure 2 shows a scenario/goal information model that is explicitly tailored to the purposes of this paper: the definition of the embedding context of policy rules. Approaches focusing different purposes of course apply other (tailored) information models. A *scenario* consists of one or more steps, and each step may be part of several scenarios (see Figure 2). Scenarios are performed by subjects. From the system's perspective, a *subject* is an external agent and may be either a human being or an other system or software program. Note that the *linked to* relations depicted in Figure 2 serve as placeholders for different kinds of possible trace relations (see, e.g., [8]).

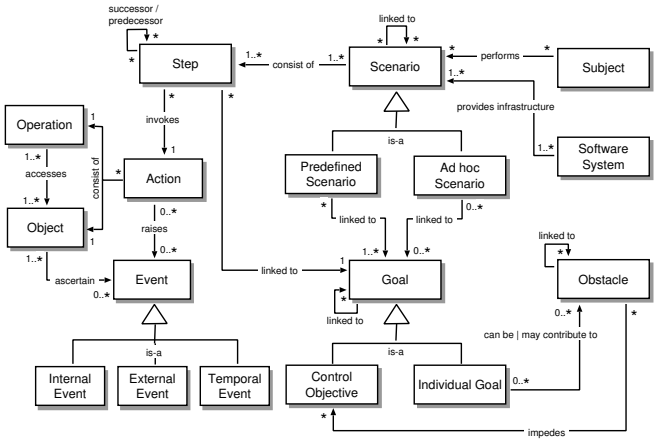

Figure 2. Scenario/Goal information model

A scenario can be modeled as a directed (possibly cyclic) graph. Each node in such a scenario graph represents a step, and steps are connected via directed edges. Each *step* invokes an action, and each *action* consists of an *operation* and a target *object* (cf. Figure 2). Actions may raise events, and an *event* is defined as an occurrence that may influence further steps or scenarios which can, or must, be executed within the respective system. Events are ascertained by objects, and the decision which events are released depends on the return value of the operation that accesses the corresponding object.

Moreover, for the purposes of this paper we distinguish two kinds of goals: control objectives and individual goals: A *control objective* is a goal specified by the authority which is responsible for the operation of a particular system. Thereby, control objectives define acceptable system behavior as intended by the system authority. In contrast to that, we define an *individual goal* as a goal representing a subject's intentions when using the system - in opposition to control objectives which reflect goals of the system authority. However, individual goals are not necessarily in conformance with the control objectives defined for a system and may even be contrary to the control objectives (see also [1]). This is especially true for malicious individual goals, like attempts to hack/crack a system, to deliberately circumvent protection measures, or to use system functions in an unintended manner. Therefore, (malicious) individual goals can be *obstacles* impeding the control objectives defined for a system (cf. Figure 2).

Control objectives can be derived from predefined as well as from ad hoc scenarios, while individual goals are (often) derived by observing ad hoc scenarios. Here, a *predefined scenario* is a scenario which was explicitly defined to model the execution of a certain intended (or unintended) system function. These scenarios are then linked to one or more control objectives (and maybe to one or more obstacles). Predefined scenarios therefore define the *context of related control objectives* and facilitate understanding of its origin. An *ad hoc scenario*, on the other hand, is a scenario that may be performed by executing a certain step sequence, but was not originally intended by the system authority. Moreover, predefined scenarios can be seen as an *instantiation of a system's decision making procedures* in case of a specific action and event sequence. Policies are rules governing the choices in behavior of a system. Therefore, scenarios are an important prerequisite for the specification and comprehension of policy rules.

3 Policies for Software-based Systems

Policies apply to a set of operations and objects, whereby an object invoking an operation on another object is called the *active object*, or the *subject*, of this invocation. The object an operation is aimed at is called the *passive object*, or *target*, of the respective invocation. A *policy rule set* consists of several related or interdependent rules, and a software system may be controlled by various policy rule sets. In a policy-hierarchy, higher level policies are more abstract policy descriptions which are refined on lower levels. In addition, policy definitions for a specific system may be influenced by orthogonal policies acting as meta-policies or invariants.

Figure 3 depicts an information model for policy rules. Note that this information model does not include each policy-related concept but focuses the core assets that are needed for the purposes of this paper. Here, a *Policy Rule* consists of a tuple including the following elements: a *Role* or an other suit-

able type of subject abstraction, and an *Action-Spec* (abbreviation of: action-specification). With respect to policies for software-based systems, a *Role* contains the rights and duties of a certain subject-type, and subjects may be either human users or other software-based systems. An *Action-Spec* (cf. Figure 3) consists of an operation-type (or an actual operation) and an object-type (or an individual object). An *Operation-type* groups a number of similar operations. An *Object-type* represents a number of objects with similar characteristics. Due to the administrative overhead it is often not practical to specify policy rules relating to each individual entity in real-world systems. However, if required, it is of course possible to define policy rules for individual subjects or objects.

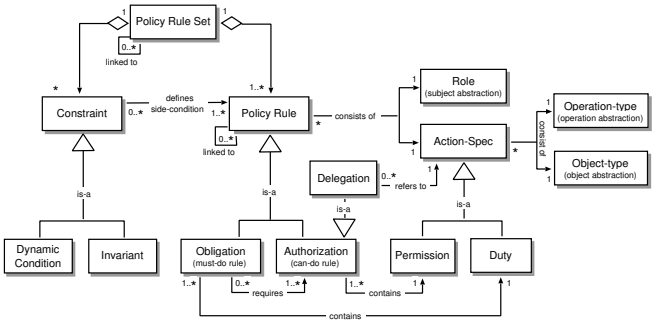

Figure 3. Core information model for policies

Here, an action-spec may be either a permission or a duty (see Figure 3). A *Permission* grants the right to perform the specified operation (or type of operation) on the specified object (or on objects of a particular type). In contrast to that, a *Duty* obliges to perform the specified operation on the specified object. A policy rule can be an *Obligation*, an *Authorization*, or a *Delegation*. A *Delegation* is a specific-type of authorization which contains the permission for a role to delegate a specific *Action-Spec* to another role. In order to discharge an obligation a subject needs sufficient authority, which means that an obligation requires at least one corresponding authorization. Therefore, an obligation *must* be linked to at least one authorization, while an authorization *can* be linked to obligations. However, note that in a system which uses obligation policies *and* strictly enforces the principle of *least privilege* each authorization *must* be linked to at least one obligation. A *Constraint* defines a function that checks one or more predicates. A constraint is either a system *Invariant* that must hold at any time or a *Dynamic Condition* that is evaluated at runtime depending on the given input parameters (see, e.g., [11]). A policy rule which is linked to one or more constraints is called a *constrained policy rule*.

4 A Complementary View

A *Policy Domain* defines the scope of a policy rule set. A subject or object that is referenced by a domain is said to be a *direct member* of this domain. Subjects and objects may enter and leave domains dynamically and may be members of several domains at the same time. Domains can be nested, and a member of a subdomain is an *indirect member* of the corresponding parent domain(s). Moreover, subdomains inherit the policies defined for their parent-domains (see, e.g., [10]).

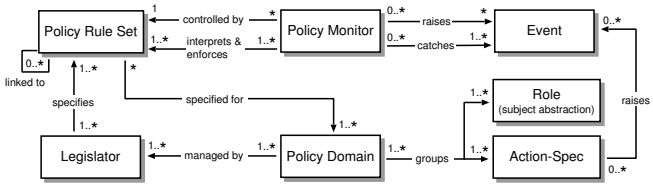

Figure 4. Policy rule set and policy domain

A *Legislator* is a person or an organizational unit which is empowered to specify policy rules that are valid within a well-defined policy domain (see also Figure 4). Each policy rule set is interpreted and enforced via a *Policy Monitor*. Policy monitors are trusted system objects which (depending on the respective implementation) are themselves (implicitly or explicitly) controlled by a policy rule set that governs the behavior of this particular policy monitor. A policy monitor catches system events that are relevant for the evaluation of certain policy rules (e.g. an access request raises an event that triggers the evaluation of corresponding authorization rules). Further on, the policy monitor raises system events to indicate the result of a rule evaluation and to trigger subsequent actions.

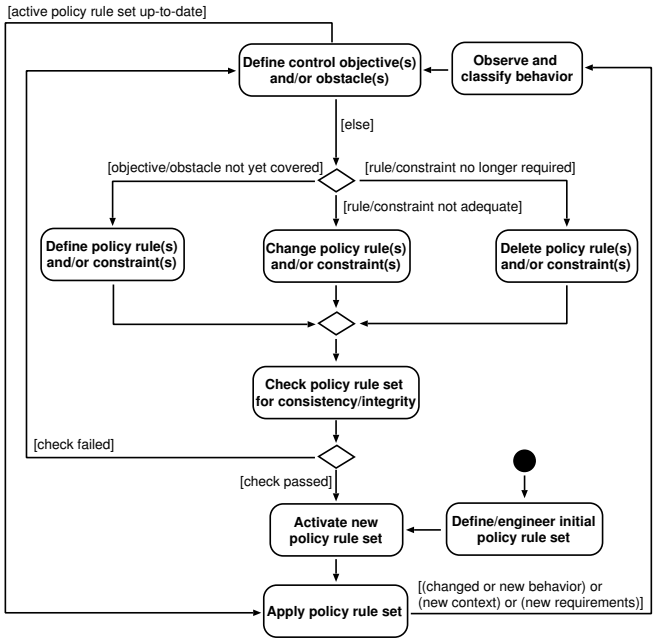

Figure 5. Maintenance of policy rule sets

Authorizations and obligations alone are not sufficient to model every behavioral aspect of an information system. In particular, suitable means are required to enable the description of interactions between different system entities. Thus, scenarios are well suited to define how policy rules are embedded into a specific environment. In other words, like requirements *policies only exist in context*, and a policy rule set can only be assessed and sensibly interpreted with adequate knowl-

edge of its embedding context. Subsequent to the definition of an initial policy rule set, the legislator observes the scenarios/processes that are performed within the respective domain. For example, changes to the policy rule set may be required if an (undesired) new behavior occurs (e.g. a subject performs unforeseen scenarios, i.e. ad hoc scenarios which are not explicitly modeled), or if the system context changes (e.g. by implementing new features in the system), or if new requirements arise (see Figure 5). In such situations, the legislator classifies the new behavior, defines the corresponding control objectives and/or obstacles (see, e.g., [7, 11]), and, if necessary, modifies the policy rule set accordingly (cf. Figure 5 and 6).

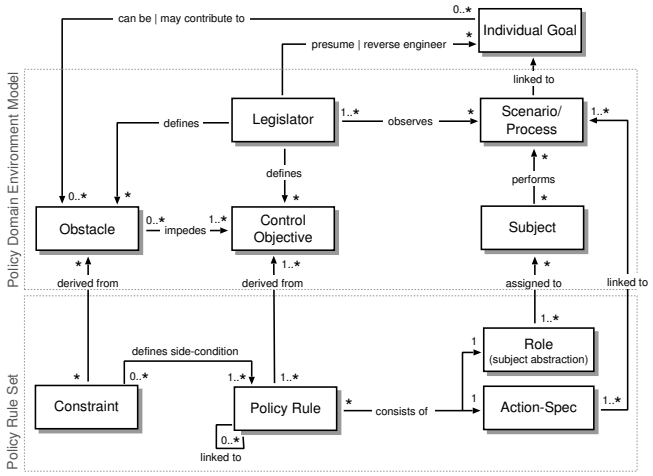

Figure 6. Embedding into a system context

The lower box in Figure 6 symbolizes a policy rule set, while the upper box covers the *Policy Domain Environment Model*. Individual goals are not (directly) included in the environment model since they represent a subject's individual intentions which may be very hard to elicit and may be contrary to the control objectives defined by the respective legislator (see also Section 2).

5 Related Work

Moffett [6] suggests that (relatively static) high-level policies can be seen as system requirements, while low-level policies can be seen as implementations of the respective high-level policies. In [6], however, the interrelations of policies and requirements are not further elaborated. In [4] Barrett motivates the need for a discussion of policy-driven system management from a human perspective, rather than a pure technical perspective. He states that, due the advantages of policy-based systems on a technical level, it is equally important to support human users in the definition and comprehension of systems that are controlled via complex policy rules. Since specifications for policy-based systems still need to be read and understood by human users, it is crucial to correctly understand when a policy applies, why it exists, and what the risks and alternatives of certain policies could be. As machine-readable policies are not likely to include such information, it is necessary to embed policies in a system context and thereby enable readers to trace policies back to their origin.

Alghathbar and Wijesekera [2] suggest to specify and analyze information flow control policies during requirements engineering. In particular, they describe the derivation of policies from UML sequence diagrams. In [3] Bandara et al. present an approach to refine high-level goals into implementable policy rules. In essence, Bandara et al. describe an engineering approach for policy rules that is based on goals and scenarios. Their approach, however, focuses on the refinement of (high-level) goals into policy rules based on Event Calculus to allow for a formal analysis of policies and to verify the correctness of a goal refinement hierarchy. Thereby it has a different perspective and a different purpose than our paper that primarily focuses on the integration of requirements engineering artifacts and policy rules on a modeling level.

References

[1] I. Alexander. Misuse Cases: Use Cases with Hostile Intent. *IEEE Software*, 20(1), January/February 2003.

[2] K. Alghathbar and D. Wijesekera. Analyzing Information Flow Control Policies in Requirements Engineering. In *Proc. of the 5th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, June 2004.

[3] A. Bandara, E. Lupu, J. Moffett, and A. Russo. A Goal-based Approach to Policy Refinement. In *Proc. of the 5th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, June 2004.

[4] R. Barrett. People and Policies: Transforming the Human-Computer Partnership. In *Proc. of the 5th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, June 2004.

[5] M. Jarke, X. Bui, and J. Carroll. Scenario Management: An Interdisciplinary Approach. *Requirements Engineering Journal*, 3(3/4), 1998.

[6] J. Moffett. Requirements and Policies. In *Proc. of the 1st International Workshop on Policies for Distributed Systems and Networks (POLICY)*, November 1999.

[7] G. Neumann and M. Strembeck. A Scenario-driven Role Engineering Process for Functional RBAC Roles. In *Proc. of 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2002.

[8] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering (TSE)*, 27(1), January 2001.

[9] C. Rolland, C. Souveyet, and C. B. Achour. Guiding Goal Modeling using Scenarios. *IEEE Transactions on Software Engineering (TSE)*, 24(12), 1998.

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

[11] M. Strembeck and G. Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments. *ACM Transactions on Information and System Security (TISSEC)*, 7(3), August 2004.

[12] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proc. of the 5th IEEE International Symposium on Requirements Engineering (RE)*, August 2001.

[13] A. van Lamsweerde and E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering (TSE)*, 26(10), October 2000.