Specification of Policies Using UML Sequence Diagrams

BJØRNAR SOLHAUG, TOR HJALMAR JOHANNESSEN

Bjørnar Solhaug is working on his PhD at the University of Bergen and at SINTEF Information and Communication Technology

Tor Hjalmar Johannessen is Senior Adviser in Telenor R&I

Policy based management is an approach to administer and control distributed systems in a dynamic and flexible way. An important feature of policies is that they allow systems to be dynamically changed in order to meet new or different requirements without changing the underlying implementation of the system. A policy rather specifies the conditions under which predefined operations can be invoked. This paper presents Deontic STAIRS, a customized notation for specifying policies using UML sequence diagrams. The notation is not tailored for a specific type of policies, thus supporting the specification of policies for various purposes, such as security, access control, services and trust. The paper exemplifies the capturing and formalization of a policy for access control based on electronic authentication in a telecommunication scenario.

1 Introduction

Policy based management, see e.g. [1] for a survey, has the last decade or so emerged as a means to implement an adaptive and flexible approach to administer and control distributed systems with respect to issues such as services, networks, security and trust. An important motivation for the use of policies for systems management is that they allow systems to be dynamically changed in order to meet new or different requirements, without stopping the system and without changing the underlying implementation. This is captured in our definition of a policy, adopted from [2], namely that a policy is a set of rules governing the choices in the behavior of a system.

At the same time, the Unified Modeling Language (UML) [3] has been established as the *de facto* industrial standard for the modeling and specification of information systems. However, the UML offers little specialized support for the specification and analysis of policies. In this paper we present Deontic STAIRS, a notation customized for specifying policies using UML 2.1 sequence diagrams. The notation is defined as a modest extension of the UML, and is based on the STAIRS approach to system development with UML sequence diagrams [4].

Sequence diagrams capture dynamic/behavioral aspects of information systems, and since policies express constraints on system behavior, sequence diagrams are a suitable candidate for policy specification. The development of a sequence diagram notation customized for policy specification is desirable also because it facilitates the analysis of the relation between a policy specification and a system specification, where the latter is represented with standard UML sequence diagrams. Since our extension of the sequence diagram notation is conservative with respect to the UML standard, people that are familiar with the UML should be able to understand and use the notation.

Deontic STAIRS is not tailored for a specific type of policy, thus allowing the specification of policies for access control, service level, security management, trust management, etc. For specific domains a special purpose policy language, e.g. XACML [5] for access control, will typically have tailored constructs for its domain. A general purpose language such as Deontic STAIRS is, however, advantageous as it offers techniques for policy capturing, specification, development and analysis across domains and at various abstraction levels.

In this paper we use an example demonstrating the specification of policy rules for access control based on electronic authentication (e-authentication) of remote users in a telecommunication scenario. In the next section we give a general characterization of policies by conceptualizing the various elements of a policy. In Section 3 we introduce the e-authentication example and briefly present the standard UML sequence diagram notation. In Section 4 we show the initial, informal capturing of policy rules for access control in a structured table format. These policy rules are then formalized using Deontic STAIRS in Section 5. Section 6 characterizes and explains more closely our extensions of the UML sequence diagram notation. In Section 7 we address the relation between policies and systems and explain what it means that a system satisfies a Deontic STAIRS policy specification. Finally, in Section 8, we conclude.

2 Policy Concepts

A key feature of policies is that they "define choices in behavior in terms of the conditions under which predefined operations or actions can be invoked rather than changing the functionality of the actual operations themselves" [1]. This means that the potential behavior of the system generally spans wider than what is prescribed by the policy, i.e. the

ISSN 0085-7130 © Telenor ASA 2009

system can potentially violate the policy. A policy can therefore be understood as a set of normative rules defining the ideal, acceptable or desirable system behavior.

With a deontic approach to policy capturing and specification, policy rules are represented as conditional normative rules in the form of permissions, obligations and prohibitions. This classification is based on standard deontic logic [6], and several of the existing approaches to policy specification have language constructs of such a deontic type [7], [8], [9], [10]. This categorization is furthermore used in the ISO/IEC standardized reference model for open, distributed processing [11].

Taken together, the above definition of a policy and description of policy rules yield a characterization of the ingredients or constructs that are required in order to capture the relevant elements of a policy. This characterization is depicted in the UML class diagram of Figure 1. The diagram shows the elements of a policy and how they are related.

The class diagram depicts that a policy is a nonempty set of policy rules. A policy rule is a permission, an obligation or a prohibition (which we refer to as the modality of the rule), and imposes a constraint on some behavior.

Since a policy defines the conditions under which operations or actions can be invoked, policy rules are conditional rules. This is depicted in the class diagram by the notion of trigger. The trigger specifies the conditions under which the policy rule applies, and these conditions can be certain system behavior or certain system states. A policy trigger can also be given as a combination of the two, in which case the rule applies by the occurrence of certain behavior in certain states. Furthermore, the trigger can be omitted in the specification of a policy rule. In that case the rule applies under all circumstances and is referred to as a standing rule.

3 Electronic Authentication

To illustrate language constructs and central notions we use a running example throughout the paper addressing the issue of authorizing access to an information system on the basis of e-authentication of remote users. We use a somewhat simplified e-authentication model based on the NIST guideline [12], where e-authentication is defined as "the process of establishing confidence in user identities electronically presented to an information system".

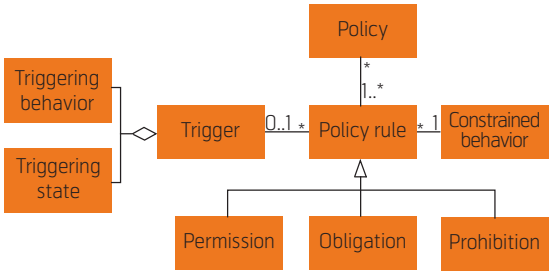

Figure 1 Policy concepts

For a user to be able to authenticate to an information system, the user must be a subscriber to a credential service provider (CSP) whose services are used by the system. The CSP is a trusted entity that issues tokens and (electronic) credentials to subscribers. A credential is a digital document (e.g. a public key certificate) that binds an identity to a token and is used in authentication. A token (e.g. a private key or a password) is possessed by the subscriber. The authentication of a subscriber to a system amounts to proving through a secure protocol the possession and control of the token by the subscriber.

Authentication begins by the user applying to a registration authority (RA) to become a subscriber to a CSP. This is illustrated in the sequence diagram *subscribe* in Figure 2. The RA receives the application to register along with documentation of the user's id. After the successful identity proofing, the RA notifies the CSP, which in turn issues a credential and a token to the user which now has become a subscriber to the SCP.

Figure 2 shows the most basic constructs of UML sequence diagrams. The keyword sd to the upper left denotes the type of diagram and is followed by a chosen name for the diagram. Each of the entities partici-

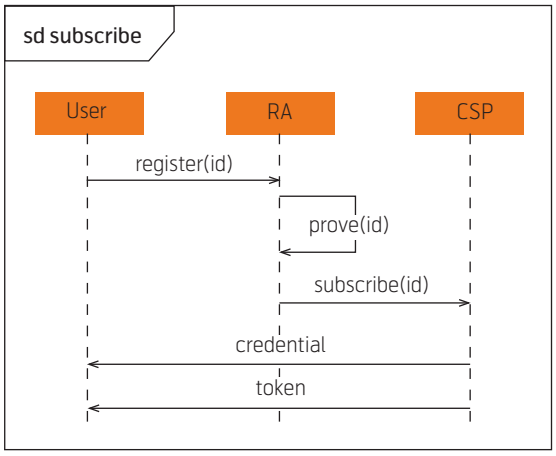

Figure 2 A registration authority (RA) identity proofs the user, after which the user becomes a subscriber to a credential service provider (CSP). Being a subscriber, the user can later authenticate to systems by means of a credential and a token issued by the CSP

ISSN 0085-7130 © Telenor ASA 2009

Rule ID	Triggering behavior	Triggering state	Modality	Constrained behavior
balance	The user securely authenticates to the banking server	The credential has not expired	Permission	The user retrieves its account balance
denial	The user fails to securely authenticate to the banking server	N/A	Prohibition	The user retrieves its account balance
logFailure	The user fails to securely authenticate to the banking server	N/A	Obligation	The server logs the transaction data and the user credential

Table 1 The table format for the initial capturing of a policy, where each row represents a rule and each column corresponds to an element of the conceptual model depicted in Figure 1

pating in the interaction is represented by a dashed, vertical line called a lifeline, where the box at the top of the line contains the name of the entity. The interaction between the entities is represented by messages which are depicted by arrows from one lifeline to another (or to itself).

In addition to what is shown in Figure 2, UML sequence diagrams are equipped with a set of composition operators for combining diagrams, most notably seq for weak sequencing, alt for specifying alternatives, par for parallel composition, and loop for several sequential compositions of one diagram with itself. The reader is referred to [4], [13] for further details about the sequence diagram notation and its formalization in STAIRS.

4 Policy Capturing Using Tables

In this section we show the initial, informal capturing of some relevant policy rules for access control based on e-authentication. Before we address the specific rules we continue the e-authentication example.

Once a user has become a subscriber to a credential service provider, he or she can use the issued credential to authenticate to an information system. This may begin by the user sending its credential to a server within the system. The server can then confirm that the credential is issued and validated by the given CSP, and that the credential has not expired. Successful authentication thereafter requires that the user proves through a secure authentication protocol that he or she controls the token binding the credential to the user's identity.

The relying party can use authenticated information to make decisions with respect to access control and authorization. In the following we give examples of policy rules that govern such decisions where we assume that the services that are provided are banking services over the internet.

Depending on the type or purpose of a policy, the initial capturing of the policy rules may be conducted on the basis of, for example, risk analyses, security requirements, service level agreements or trust relations. Before the formalization, implementation and enforcement of a policy it may be useful to capture and specify policy rules in an informal, yet structured way. Such an informal structuring of a policy can be conducted using the table format shown in Table 1. In this table each of the policy rule elements of the class diagram of Figure 1 is represented by a column. A row in the table describes a policy rule and is identified by a unique name, the rule ID. Such a table as a whole is then a representation of a policy to be formalized and implemented.

A very basic banking service to remote users is to provide access to the user's account balance. This is a service that should be available to authenticated users and is therefore captured as a permission rule. In the table format, this rule is specified with the rule ID 'balance' in Table 1.

For this permission rule to apply, i.e. for the rule to trigger, the system behavior of secure user authentication must occur. Additionally the credential held by the user cannot have expired. These two conditions for the rule to trigger are captured by the triggering behavior and the triggering state, respectively, specified in the 'balance' row of Table 1. Notice that the triggering state is specified by a proposition; a system state fulfills the trigger if the proposition evaluates to true in that state.

In the specification of access control policies, a possible strategy is to deny all services for which there are no explicit permissions, i.e. to hold all services as prohibited by default. In many cases it may, however, be useful to be able to explicitly specify prohibitions. With respect to the user access to account balance, for example, this should be prohibited in case of authentication failure.

The rule with ID 'denial' in Table 1 captures this prohibition. Notice that there is no state trigger specified for this rule since it suffices that the authentication protocol fails for user access to be denied. Additionally a second prohibition rule could be specified for the case in which the user credential has expired.

For the purpose of enhancing the security of the system the relying party may wish to log significant transactions. The rule with ID 'logFailure' in Table 1 states that in case of the user failing to authenticate to the server, the server is obliged to log the transaction data as well as the user credential.

The collection of the rows of a policy table represents the rules that make up the policy that is captured during the initial phase. In our example we have shown only three rules, but further rules could be captured, e.g. for specifying the conditions under which a user credential should be disabled, or rules for access to more critical services in which a stronger e-authentication is required. For the purpose of this paper of introducing the Deontic STAIRS notation and demonstrating its use, the three above policy rules suffice.

5 Policy Specification Using Deontic STAIRS

In this section we show the formalization in Deontic STAIRS of the policy rules informally captured in the previous section. The formalization demonstrates that Deontic STAIRS has the expressiveness to capture the information of each cell for each row of a policy table. This also shows that the notation has language constructs corresponding to each element of the conceptual model in Figure 1.

Two of the cells in each row describe system behavior of relevance for a rule, namely the triggering behavior and the constrained behavior. In Deontic STAIRS system behavior is described as interactions using standard UML sequence diagrams. For the permission rule 'balance' represented in Table 1 the triggering behavior is specified by the sequence diagram

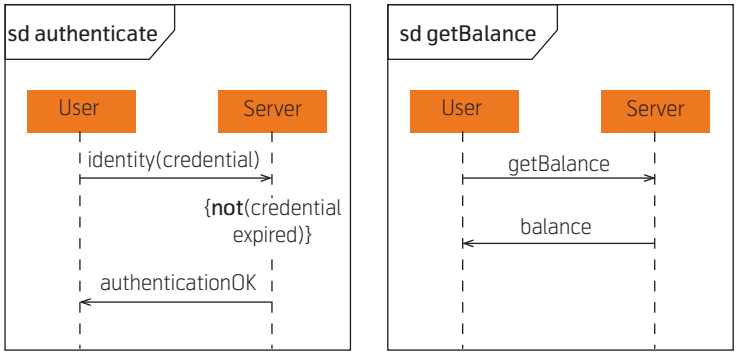

Figure 3 The sequence diagram authenticate *specifies the successful authentication of a user to the banking server. The sequence diagram* getBalance *specifies user retrieval of its account balance from the banking server*

authenticate in Figure 3, whereas the constrained behavior is specified by the diagram *getBalance*.

Note that we have omitted the specification of the authentication protocol in the diagram *authenticate* since these details are irrelevant for the purposes of this paper. The interested reader is referred to e.g. [14].

The triggering state is captured by a constraint within the specification of the triggering behavior. In UML constraints are Boolean expressions, i.e. expressions that evaluate to true or false, written in curly brackets as illustrated in the diagram *authenticate* of Figure 3. A system state satisfies the constraint, i.e. fulfills the triggering state, if the expression evaluates to true in that state.

Having captured the triggering behavior, triggering state and the constrained behavior, the permission rule is formalized with the Deontic STAIRS diagram balance of Figure 4.

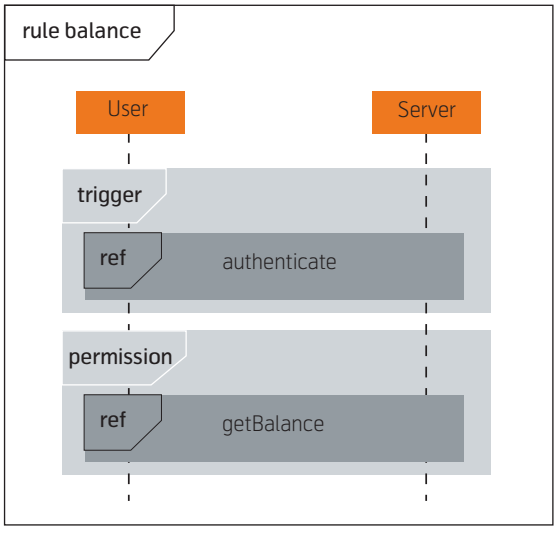

Figure 4 The Deontic STAIRS diagram balance *formalizes the permission rule 'balance' captured in Table 1*

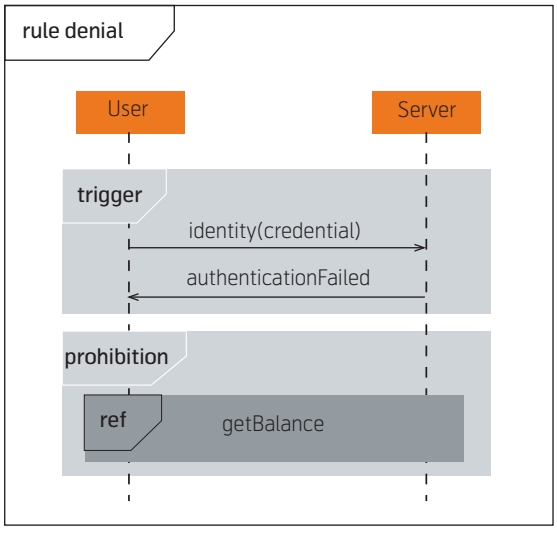

Figure 5 The Deontic STAIRS diagram denial *formalizes the prohibition rule 'denial' as captured in Table 1*

ISSN 0085-7130 © Telenor ASA 2009

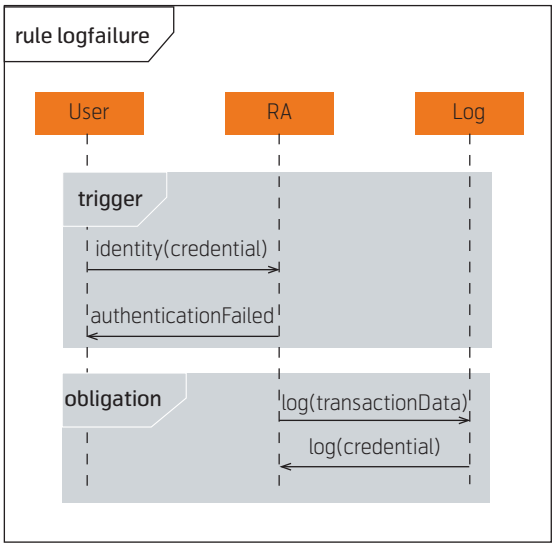

Figure 6 The Deontic STAIRS diagram logFailure *formalizes the obligation rule 'logFailure' as captured in Table 1*

The keyword rule at the top left denotes that the diagram specifies a policy rule. This keyword is followed by a chosen name for the rule. For convenience the chosen name should correspond to the rule ID from the policy table.

The diagram consists of two parts, a trigger and a body. The trigger specifies the conditions under which the rule applies and is denoted by the keyword trigger. The body specifies the behavior that is constrained by the rule and is denoted by a keyword specifying the modality of the rule, in this case permission.

The ref construct of UML sequence diagrams specifies a so-called interaction use, which is a reference to another diagram. This facilitates modularity and reuse of specifications; equivalently the interaction use can be replaced with the content of the referred diagram. As explained above, the diagram *authenticate* specifies the conditions under which the rule with ID 'balance' applies, whereas the diagram *getBalance* specifies the constrained behavior. Altogether the diagram *balance* of Figure 4 thereby specifies each element of the rule 'balance' as captured in Table 1.

The prohibition rule 'denial' is formalized in Deontic STAIRS with the diagram *denial* of Figure 5. The keyword prohibition captures the modality of the rule, and the interaction of the trigger specifies the triggering behavior. Since the constrained behavior of 'denial' is identical to the constrained behavior of 'balance' we simply refer to the previously specified interaction with the ref construct.

The diagram *logFailure* formalizes in the same way the rule 'logFailure' captured in Table 1. The keyword obligation captures the modality of the rule, and we see that the constrained behavior of logging the transaction data and the user credential is specified as an interaction between the server and the system log.

The diagrams of this section exemplifying Deontic STAIRS are all quite simple since the purpose of this paper is to present the basic constructs of Deontic STAIRS and to demonstrate the suitability of formalizing policy rules. Generally, however, the triggering behavior and the constrained behavior can be specified as sequence diagrams of any size, and can be composed by using the UML composition operators such as alt, seq, par and loop.

Policy: A policy is with Deontic STAIRS specified as a set of policy rule diagrams

Policy rule: A policy rule is specified with the Deontic STAIRS diagram denoted by the keyword **rule**

Permission: This modality of a rule is represented by the Deontic STAIRS keyword **permission**

Obligation: This modality of a rule is represented by the Deontic STAIRS keyword **obligation**

Prohibition: This modality of a rule is represented by the Deontic STAIRS keyword **prohibition**

Constrained behavior: Represented by a UML sequence diagram denoted by the keyword representing the modality of the constraint

Trigger: Represented by a UML sequence diagram denoted by the keyword **trigger**

Triggering behavior: Represented by the interaction of the sequence diagram specifying the trigger

Triggering state: Represented by a UML guard in the sequence diagram specifying the trigger

Figure 7 Relating each concept of the class diagram of Figure 1 to language constructs in Deontic STAIRS

6 How Deontic STAIRS Extends UML

In this section we characterize and discuss more closely the extensions of UML that define Deontic STAIRS. In particular we explain why precisely these extensions are required for being capable of properly specifying policies using sequence diagrams.

The language constructs of Deontic STAIRS are summarized in Figure 7 by linking each of the policy concepts of Figure 1 to the language constructs. The extensions of the UML sequence diagram notation defining Deontic STAIRS are basically the construct trigger for specifying conditionals and the constructs permission, obligation and prohibition for specifying the modality of deontic constraints.

The expressiveness to specify conditionals is important for properly specifying policy rules, and with the designated trigger construct this is straightforward in

Deontic STAIRS. In relation to a system for which a policy specification applies, the trigger of a policy rule does not require the triggering behavior to be fulfilled by the system. Rather it requires that *if* the triggering behavior occurs, the rule imposes a constraint on system behavior. In standard UML, on the other hand, sequence diagrams can only specify behavior that should or should not occur. Since triggers may be omitted in the specification of a policy rule, the specification of such unconditional requirements to system behavior is also supported in Deontic STAIRS by standing policy rules.

The constructs for specifying deontic modalities facilitate the explicit characterization of behavior as permitted, obliged or prohibited. In standard UML, sequence diagrams can specify behavior that is valid and behavior that is invalid. The latter is similar to the specification of prohibited behavior in Deontic STAIRS and requires that the given behavior should never occur.

In a sense both obligations and permissions in Deontic STAIRS specify valid behavior, but with a distinction that cannot be captured in standard UML. Obligations require that the corresponding behavior must occur, whereas permissions require that the corresponding behavior must be offered as a potential alternative. This means that permitted behavior must be offered as a choice, yet allowing alternative behavior to be conducted or chosen instead. This distinction is important for properly capturing the various types of policy rules and as such represents a crucial element in the extension of the sequence diagram notation that defines Deontic STAIRS.

7 Relating Policy Specifications to Systems

In this section we explain the semantics of the policy specifications and show how policy specifications relate to systems to which the policies apply.

Semantically a standard UML sequence diagram is represented by traces of events ordered by time. Events occur on lifelines and are either the transmission of a message or the reception of a message. The transmission of a message is ordered before the corresponding reception of the message, and events are ordered downwards along each lifeline.

Consider, for example, the obligation rule *logFailure* of Figure 6. The interaction specifying the triggering behavior is the sequence of the two messages *identity*(*credential*), abbreviated *i*(*c*), and *authenticationFailed*, abbreviated *a*. A transmit event is denoted by ! and a receive event is denoted by ?, so the trace representing the triggering behavior is ‹!*i*(*c*), ?*i*(*c*), !*a*, ?*a*›.

When relating a policy specification to an implemented system, we need to create a model of the system. Such a model can be the set of possible system executions, where a system execution is represented by a trace describing the sequence of events that occur during the execution. Adherence of a system to a policy specification means that the system executions satisfy the rules of the policy specification. Since policy rules are conditional, a policy rule imposes a constraint on a system execution only if the execution fulfills the policy trigger. Furthermore, when specifying a policy we consider only the system entities and the system behavior of relevance for the policy, which means that the system may have behavior that is not mentioned in the policy specification. For a system trace to fulfill a trace of a policy rule, the latter need therefore only be a sub-trace of the former.

Assume, for example, that at some point during a system execution the following sequence of events occur.

‹..., !*r*, !*d*, ?*d*, !*i*(*c*), ?*r*, ?*i*(*c*), !*a*, ?*a*, !*l*(*t*), !*f*, !*l*(*c*), ?*l*(*t*), ?*l*(*c*), ?*f*, ...›

Since the trace ‹!*i*(*c*), ?*i*(*c*), !*a*, ?*a*› representing the trigger of the obligation rule *logFailure* is a sub-trace of the given system trace, i.e. the four events occur in the same order, the system traces fulfill the trigger.

At the point of the occurrence of the event ?*a* in the given system trace, the obligation rule therefore applies and imposes a constraint on the continuation of the system trace. It is easily verified that the continuation fulfills the obliged behavior specified in *logFailure*. This means that the given system execution adheres to this policy rule.

More precisely, system adherence to a policy specification means that the system adheres to each rule of the specification. Adherence to an obligation rule means that if the rule is triggered at some point during a system execution, all possible continuations of the system execution fulfill the obliged behavior. Adherence to a prohibition rule means that none of the possible continuations fulfill the prohibited behavior. Finally, adherence to a permission rule means that there must exist at least one possible continuation that fulfills the permitted behavior, meaning that the system must offer the permitted behavior as a potential choice.

Standard UML sequence diagrams also specify requirements to system executions and are in the same way represented by traces in the semantics. As we

have seen, however, these requirements are not conditional. The traces representing a sequence diagram are rather describing behavior that the system should or should not have. Standard sequence diagrams are furthermore normally interpreted as describing exactly the traces of events that must occur in order to fulfill the specified behavior, i.e. there is no notion of other, irrelevant behavior that may be interleaved.

This interpretation of Deontic STAIRS specifications is formalized with the adherence relation [13] and is together with the customized constructs of trigger and deontic modalities important for facilitating policy specification; through the capturing and formalization of a policy we need only consider the relevant issues, such as security, trust and service level. Furthermore, together with the support for specifying conditional requirements, this interpretation facilitates the specification of policy rules that applies for several systems or several implementations of a system.

8 Conclusion

In this paper we have presented Deontic STAIRS, a customized notation for specifying policies using UML 2.1 sequence diagrams. The notation is defined as a modest and conservative extension of UML, allowing the specification of conditional deontic rules in the form of permissions, obligations and prohibitions.

The UML is the *de facto* standard for the modeling and specification of information systems. Customizing the UML for specific purposes may be advantageous since the knowledge about the UML is widespread, but there is a potential pitfall: If UML constructs are not used in accordance with the UML standard, the customized language may easily be used erroneously. This pitfall is avoided in the development of Deontic STAIRS, so people that are familiar with the UML should be able to understand and use the notation.

Basing a policy specification language on the UML also has the advantage that the graphical appearance facilitates communication between personnel of different backgrounds, such as decision makers, security personnel and developers.

Furthermore, providing the UML with support for specifying policies facilitates the analysis of the relation between a policy specification and the specification of a system for which the policy applies, where also the latter is given in UML. STAIRS is an approach to system development with UML sequence diagrams, and in [15] the notion of adherence of a STAIRS system specification to a Deontic STAIRS policy specification is formalized.

Deontic STAIRS is a generic approach to policy specification, meaning that the notation supports the specification of policies of various types. In [13] and [16] Deontic STAIRS is presented within the domains of access control and trust management, respectively. In this paper we have exemplified the formalization of policy rules in the setting of e-authentication and access control in a telecommunication scenario. The initial capturing of the policy rules is in this example conducted by an informal structuring of the rules in a table format. In the table format several details about the system architecture, system entities, system functionality, etc. are not taken into account, so the formalization of the rules in Deontic STAIRS may represent a shift that brings the policy specification closer to implementation and enforcement.

In a practical setting of formalizing and developing a policy specification there is usually a process in which the most important issues are taken into account during the initial phases, and as the process evolves more details are considered, rules may be strengthened and other rules may be added. This process can be conducted by several iterations with backtracking to the policy table where rules are modified and added, before the formalization in Deontic STAIRS is modified and extended accordingly. A development process based on such iterations may, however, be unfortunate if it requires constant redesign of the specification and if it is not precisely understood what it means that one policy specification is a strengthening of another.

Iterations with backtracking and redesign is avoided when the development process is supported by a well defined notion of refinement, precisely capturing what it means to strengthen a policy specification and increase the level of detail. In [13] the notion of refinement of specifications in Deontic STAIRS is formalized, allowing rules to be added, rules to be strengthened, and new details to be taken into account in a stepwise and incremental manner. Refinement in Deontic STAIRS ensures that a correct enforcement of the final specification implies the enforcement of all the previous specifications throughout the development process.

In summary this means that with Deontic STAIRS we are provided not only a customized notation for specifying policies, but also support for a stepwise and incremental development of specifications, facilitating the process from the initial policy capturing down to the final specification to be enforced.

Acknowledgments

The research on which this paper reports has partly been funded by the Research Council of Norway through the projects ENFORCE (164382/V30) and

DIGIT (180052/S10), and partly by the European Commission through the S3MS project (Contract no. 27004) under the IST Sixth Framework Programme.

References

1 Sloman, M, Lupu, E. Security and management policy specification. *Network, IEEE*, 16 (2), 10-19, 2002.

2 Sloman, M. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2, 333-360, 1994.

3 OMG. *Unified Modeling Language : Superstructure, version 2.1.1*. Object Management Group, 2007.

4 Haugen, Ø, Husa, K E, Runde, R K, Stølen, K. STAIRS Towards Formal Design with Sequence Diagrams. *Journal of Software and Systems Modeling*, 4, 355-367, 2005.

5 OASIS. *eXstensible Access Control Markup Language (XACML) Version 2.1*, 2005.

6 McNamara, P. Deontic Logic. **In**: Gabbay, D M, Woods, J (eds). *Logic and the Modalities in the Twentieth Century*, volume 7 of *Handbook of the History of Logic*, 197-288. Elsevier, 2006.

7 Aagedal, J Ø, Milosevic, Z. ODP Enterprise Language: UML Perspective. **In**: *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, 60-71. IEEE CS Press, 1999.

8 Kagal, L, Finin, T, Joshi, A. A Policy Language for a Pervasive Computing Environment. **In**: *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY'03)*, 63-74. IEEE Computer Society, 2003.

9 Steen, M, Derrick, J. Formalising ODP Enterprise Policies. **In**: *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, 84-93. IEEE CS Press, 1999.

10 Wies, R. Policy Definition and Classification: Aspects, Criteria, and Examples. **In**: *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems : Operation and Management*, 1994.

11 ISO/IEC. *FCD 15414, Information Technology – Open Distributed Processing – Reference Model – Enterprise Viewpoint*, 2000.

12 Burr, W E, Dodson, D F, Polk, W T. *Electronic Authentication Guideline*. National Institute of Standards and Technology, 2006.

13 Solhaug, B, Stølen, K. Compositional refinement of policies in UML – Exemplified for access control. **In**: *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS'08)*, vol 5283 of *LNCS*, 300-316. Springer, 2008.

14 Stallings, W. *Cryptography and Network Security*. Prentice Hall, fourth edition, 2006.

15 Seehusen, F, Solhaug, B, Stølen, K. Adherence preserving refinement of trace-set properties in STAIRS : Exemplified for information flow properties and policies. *Journal of Software and Systems Modeling*, 8 (1), 45-65, 2009.

16 Refsdal, A, Solhaug, B, Stølen, K. A UML-based Method for the Development of Policies to Support Trust Management. **In**: *Trust Management II – Proceedings of 2nd Joint iTrust and PST Conference on Privacy, Trust Management and Security (IFIPTM'08)*, 33-49. Springer, 2008.

Bjørnar Solhaug received his Master's degree in Language, Logic and Information from the University of Oslo in 2004. He is currently working on his PhD at the University of Bergen and at SINTEF Information and Communication Technology. His main field of interest is modeling and development of policy specifications for policy based management of distributed systems, with particular focus on trust and risk management.

bjornar.solhaug@sintef.no

Tor Hjalmar Johannessen is Senior Adviser in Telenor R&I. He graduated from the University of Oslo in 1975 as Cand.Real. After working with military crypto systems at Alcatel Telecom since 1989, he joined Telenor R&I Security Group in 2000. His main interest and occupation has been security in general and deployment of PKI systems, especially in the SIM-card and mobile systems. His activities comprise participation in various EU projects, IETF, EURESCOM, CEN/ISSSWS, GSMA, ETSI SCP and ETSI ESI.

Tor-Hjalmar.Johannessen@telenor.com