*Int J Software Informatics*, Volume 5, Issue 1-2 (2011), Part I, pp. 139–157
International Journal of Software and Informatics, ISSN 1673-7288
©2011 by ISCAS. All rights reserved.
E-mail: ijsi@iscas.ac.cn
http://www.ijsi.org
Tel: +86-10-62661040

# Preservation of Policy Adherence under Refinement

Bjørnar Solhaug[1] and Ketil Stølen[1,2]

[1] (SINTEF ICT, Norway)
[2] (Dept. of Informatics, University of Oslo, Norway)

**Abstract** In this paper we present a method based on UML sequence diagrams for integrating policy requirements with requirements to system design and functionality in the development process. The approach allows policy requirements to be taken into account throughout the system development instead of in a post hoc manner. The method supports the formalization of system specifications and policy specifications at various levels of abstraction, where the abstraction levels are related by refinement. The notion of policy adherence formally captures what it means that a system specification satisfies a policy specification. For analysis with respect to adherence at abstract levels to be meaningful, the results must be preserved under refinement. This paper gives a characterization of conditions under which adherence is preserved under refinement, and identifies development rules that guarantee adherence preservation. By results of transitivity and modularity, the development process, as well as analysis tasks, may be conducted in a stepwise manner addressing individual parts of the specifications separately.

**Key words:** policy adherence; policy refinement; adherence preservation; UML sequence diagrams

**Solhaug B, Stølen K. Preservation of policy adherence under refinement.** *Int J Software Informatics*, **Vol.5, No.1-2(2011), Part I: 139–157. http://www.ijsi.org/1673-7288/5/i79.htm**

## 1 Introduction

Policy-based management of information systems is an approach to administer and control distributed systems with respect to issues such as security, access control, service level and trust. A policy is commonly defined as a set of rules governing the choices in the behavior of a system[21]. Several frameworks for the specification, development, analysis and enforcement of policies have been proposed[3,22], but although recognized as an important research issue from the very initial research on policy-based management[12], policy refinement still remains poorly explored in the literature[1,15]. Policy refinement is in Ref.[12] referred to as the process of transforming a high-level, abstract policy specification into a low-level, concrete one. At the initial, abstract level, policies may be derived from business goals, service level agreements, risk analyses, security requirements, etc., and policy refinement should ensure that the enforcement of the final, concrete policy implies the enforcement of the initial, abstract one.

This work is partly supported by the European Commission through the NESSoS network of excellence under Grant No.256980.
Corresponding author: Bjørnar Solhaug, Email: Bjornar.Solhaug@sintef.no
Received 2010-09-20; Accepted 2011-01-03; Final revised version 2011-01-17.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

Abstraction involves the perspective or purpose of the viewer, and different purposes result in different abstractions[16]. During the initial activity of policy capturing, details about system entities, architecture and functionality that are irrelevant or unimportant from a given viewpoint can therefore be ignored. Abstraction is desirable also because detection and correction of errors, as well as policy analysis, are cheaper and easier at an abstract and high level[26]. For analysis of abstract specifications to be meaningful, however, the results of the proofs or verifications conducted at the abstract level must be preserved under refinement and by the eventual implementation. Otherwise the analysis must be conducted from scratch after every refinement step[11]. This paper addresses the problem of preservation of policy adherence under refinement. Adherence of a system to a policy specification means that the system satisfies the policy specification.

Figure 1 illustrates two development scenarios involving policies. In case (a) the policy and the system are developed in sequential order. In practice this scenario often occurs in relation to policies at the enterprise or business level that are not to be enforced by computers. The abstract policy specification $P_1$ may, for example, have been derived from a risk analysis and a set of security requirements and then further developed and refined ($\rightsquigarrow$) into the low-level and detailed policy specification $P_n$ that is to be enforced. Subsequently the system to which the policy applies is developed. In order to ensure that the security requirements are satisfied by the system, the system development is conducted in such a way that policy adherence ($\rightarrow_a$) is maintained during the development process, once established between $P_n$ and $S_1$. Preservation of adherence is denoted by the dashed arrow from $P_n \rightarrow_a S_1$ to $P_n \rightarrow_a S_m$.

Case (b) illustrates a combined development of the policy specification and the system specification. This scenario is often relevant when the policy enforcement is computer based, e.g. in terms of monitoring. A policy specification $P_i$, $i \in \{1, \ldots, n\}$, at an arbitrary level of abstraction is taken into account in the system specification $S_k$, $k \in \{1, \ldots, m\}$, also at any level of abstraction, in order to ensure adherence. As in the previous case, $S_k$ may be further developed into a specification $S_l$, but the difference is that the development process should allow the policy specification to be strengthened into a refined policy specification $P_j$.

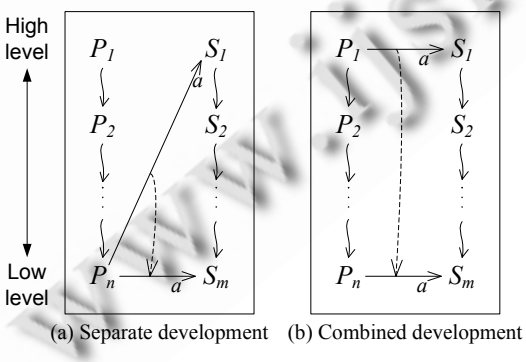



Figure 1. Development cases

In Ref.[24] we presented a method based on UML sequence diagrams[13] for the formalization, analysis and development of policies. The method extends sequence

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

diagrams with constructs customized for policy specification, and is generic in the sense that it is applicable to various domains of management, such as security, access control and trust. This extension of UML is conservative in the sense that all language constructs are used and formalized in a manner that is faithful to the informal semantics in the UML standard to the extent this is possible[17]. In Ref.[19] we addressed case (a) of Fig.1 and showed that policy adherence is preserved under refinement of system specifications formalized with UML sequence diagrams[7,18]. In this paper we address case (b) of Fig.1. In particular, this paper contributes by proposing a formal, denotational semantics of policy specification that characterizes the requirements imposed by the combined rules of a policy. This paper further proposes a formal notion of policy adherence that precisely captures what it means, for arbitrary levels of abstraction, that a system specification satisfies a policy specification. A characterization of conditions under which adherence is preserved when both policy specifications and system specifications may undergo refinement is then presented. Finally, this paper contributes by presenting specific development rules the application of which guarantees preservation of adherence under refinement in the combined development process.

In Section 2 we present the syntax and formal semantics of the policy specification language. In Section 3 we present the notion of policy adherence, and in Section 4 the notion of policy refinement. Section 5 and Section 6 are devoted to the problem of preservation of adherence under refinement; we first give a characterization of conditions for adherence preservations, and then we identify rules the application of which ensures preservation of adherence in the development process. In Section 7 we relate our work to the Focus method[4], which is an approach to the specification and development of interactive systems that was developed by Manfred Broy and Ketil Stølen. Finally, in Section 8, we conclude and discuss related work. We refer to Ref.[25] for the full proofs of all the results presented in this paper.

## 2 Formalizing Policies

We introduce and illustrate the syntax of our approach, which we refer to as Deontic STAIRS[1)], by the example diagrams depicted in Fig.2. The reader is referred to Ref.[24] for a more detailed presentation and for the full formalization. In the example we consider a fragment of a policy that administers the access of users $U$ to an application $A$. The application is for file sharing, and registered users can upload and retrieve data. The example is simplified due to space constraints and is intended to serve illustrative purposes only. The reader is referred to Ref.[23] for more elaborate cases and examples on policy-based management using Deontic STAIRS.

Sequence diagrams specify behavior by showing how entities interact by the exchange of messages, where the behavior is described by traces. A trace is a sequence of events ordered by time representing a system run, and an event is either the transmission or the reception of a message by an entity. Entities are represented by lifelines and messages are represented by arrows from lifeline to lifeline. An arrow tail denotes the transmitting of a message and an arrow head denotes the reception of a message.

1) Deontic STAIRS is called so since it basically is an extension of STAIRS[7,18] with a trigger construct and deontic modalities for policy specification.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

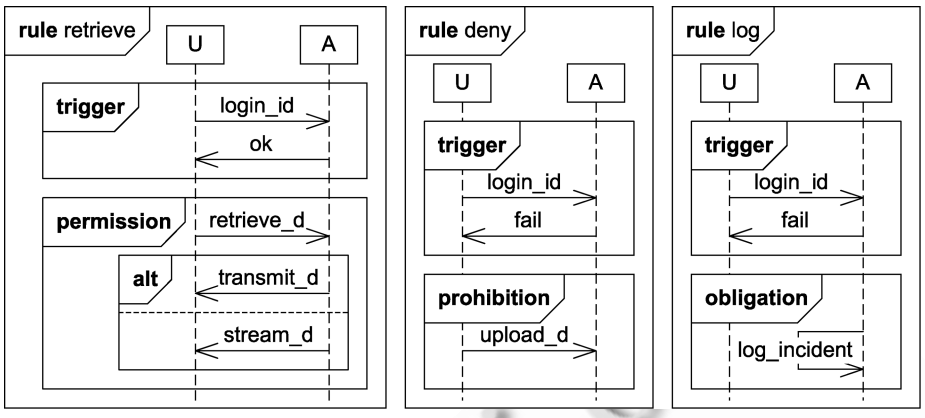



Figure 2. Policy rules

A policy rule is specified as a sequence diagram that consists of two parts, a trigger and a rule body. The keyword rule in the upper left corner denotes the kind of diagram. The trigger is a diagram that specifies the condition under which the rule applies and is captured with the keyword trigger. The rule body is a diagram that specifies the behavior that is constrained by the rule, and the keywords permission, obligation and prohibition indicate the kind of rule.

The rule *retrieve* in Fig.2 is a permission rule stating that in case of a valid user login, the user is permitted to retrieve data from the application. Notice that data retrieval is specified using the alt construct for alternative composition. The *transmit_ d* message specifies that the requested data is downloaded, and possibly saved, by the user, whereas the *stream_ d* message specifies that the data is streamed to the user. The alternative composition means that both alternatives are held as valid ways of fulfilling the permitted behavior.

Semantically the body of the rule *retrieve* is captured by the sequential composition of the *retrieve_ d* message with the alternative composition of the *transmit_ d* and *stream_ d* messages. Events are ordered vertically along each lifeline, and a transmit event is ordered before the corresponding receive event. The first event to occur is therefore the sending of the *retrieve_ d* message, which we denote $!r$, and is followed by the reception of the same message, which we denote $?r$. The trace representing the first message is denoted by $\langle !r, ?r\rangle$. In order to formally capture the variation between alternatives as exemplified by the alt construct, sequence diagrams are semantically captured by trace sets. The alternative composition of the two messages for data transfer is represented by the trace set $\{\langle !t, ?t\rangle, \langle !s, ?s\rangle\}$. Sequential composition is formalized by the $\gtrsim$ operator, so the semantics of the rule body of the *retrieve* diagram is given by $\{\langle !r, ?r\rangle\} \gtrsim \{\langle !t, ?t\rangle, \langle !s, ?s\rangle\}$, which yields the trace set $\{\langle !r, ?r, !t, ?t\rangle, \langle !r, ?r, !s, ?s\rangle\}$. The trigger of the rule *retrieve* is represented by the singleton trace set $\{\langle !l, ?l, !o, ?o\rangle\}$.

The set of all traces, i.e. the trace universe, is denoted by $\mathcal{H}$. The formalization of the sequence diagram semantics is defined by the function $[\![]\!]$ that for a sequence diagram $d$ yields a trace set $[\![d]\!] \subseteq \mathcal{H}$. The variation over traces representing a sequence diagram captures underspecification in the sense that each element represents a valid way of executing the behavior specified by the diagram. Underspecification can be

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

reduced by design choices or during implementation by choosing only some of the alternatives.

The remaining two diagrams of Fig.2 exemplify the specification of prohibitions and obligations. In case of login failure, the rule *deny* specifies that uploading data is prohibited. Finally, the rule *log* states that in case of a login failure, the application is obliged to log the incident. The only constructs of Deontic STAIRS that extends standard UML are the trigger construct and the deontic modalities. However modest, the extension provides expressiveness that allows the specification and analysis of policies for several domains and purposes.

As a shorthand notation we often represent policy rule specifications by triples $(dm, d_t, d_b)$, where $dm \in \{pe, ob, pr\}$ denotes the deontic modality, $d_t$ is the sequence diagram specifying the trigger, and $d_b$ is the sequence diagram specifying the rule body. We let $\mathcal{R}$ denote the set of all policy rule specifications. Since a policy is a set of policy rules, a policy specification is a set $P \subseteq \mathcal{R}$.

Given a trace $h$ describing a possible run of a system for which a policy rule $(dm, d_t, d_b)$ applies, $h$ triggers the rule if $h$ fulfills the triggering scenario $d_t$. Since $d_t$ is described by a set of traces, each representing a valid interpretation of the diagram, it suffices that $h$ fulfills at least one trace $h' \in [\![d_t]\!]$ to trigger the rule. For $h$ to fulfill $h'$, $h$ must be a super-trace of $h'$ which we denote $h' \prec h$. Equivalently, we say that $h'$ is a sub-trace of $h$. We have, for example, that $\langle a, b, c\rangle \prec \langle e, a, f, g, b, c, d\rangle$. The reason for operating with the sub-trace relation when formalizing what it means that a trace $h$ fulfills a diagram $d$ of a policy rule is that it allows the policy specification to ignore system events that are not relevant for the policy. The sub-trace relation means that arbitrary behavior may be interleaved with the behavior described by policy rules. For a trace set $H \subseteq \mathcal{H}$, $H \prec h$ denotes $\exists h' \in H : h' \prec h$. The complementary relations $\not\prec$ are defined by $h' \not\prec h \stackrel{\mathsf{def}}{=} \neg(h' \prec h)$ and $H \not\prec h \stackrel{\mathsf{def}}{=} \neg(H \prec h)$. The reader is referred to the technical report[25] for the formal definitions.

If $[\![d_t]\!] \prec h$ holds for a policy rule $(dm, d_t, d_b)$, i.e. $h$ triggers the rule, the rule imposes a constraint on the possible continuations of the execution of $h$ after the rule has been triggered. A permission requires that the behavior described by the rule body $d_b$ must be offered as a potential choice, i.e. there must exist a continuation of $h$ that fulfills the rule body. An obligation requires that all possible continuations fulfill the rule body, whereas a prohibition requires that none of the possible continuations fulfills the rule body.

In the following we define the semantics of policy specifications as a function that takes a policy specification $P \in \mathbb{P}(\mathcal{R})$ and a trace $h \in \mathcal{H}$ and yields a tuple $(a, u, C)$ that describes the requirements imposed by the rules in $P$ given the execution of the trace $h$, where $a, u \subseteq \mathcal{H}$ and $C$ is a set of trace sets $H \subseteq \mathcal{H}$. The interpretation of the tuple $(a, u, C)$ is that the set $a$ represents the acceptable traces as specified by the obligation rules of $P$ that are triggered by $h$; the set $u$ represents the unacceptable traces as specified by the prohibition rules of $P$ that are triggered by $h$; each set $H \in C$ represents the traces that must be offered as a potential choice as specified by a permission rule in $P$ that is triggered by $h$.

Formally, the semantics of a policy specification is defined by the following function.

$$[\![_]\!]_ \in \mathbb{P}(\mathcal{R}) \times \mathcal{H} \to \mathbb{P}(\mathcal{H}) \times \mathbb{P}(\mathcal{H}) \times \mathbb{P}(\mathbb{P}(\mathcal{H}))$$

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

We first define the function for singleton sets of policy rules $P$ and then define the composition of policy specifications $P_1$ and $P_2$. For a trace set $H$, $H{\lhd}$ denotes the set of all super-traces of elements in $H$, i.e. $H{\lhd} \stackrel{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists h' \in H : h' \lhd h\}$.

**Definition 1.** Semantics of policy rules.

$$[\![\{(dm, d_t, d_b)\}]\!](h) \stackrel{\mathsf{def}}{=} \begin{cases} (\mathcal{H}, \emptyset, \{([\![d_t]\!] \gtrsim [\![d_b]\!]){\lhd}\}) & \text{if } [\![d_t]\!] \lhd h \wedge dm = pe \\ (([\![d_t]\!] \gtrsim [\![d_b]\!]){\lhd}, \emptyset, \emptyset) & \text{if } [\![d_t]\!] \lhd h \wedge dm = ob \\ (\mathcal{H}, ([\![d_t]\!] \gtrsim [\![d_b]\!]){\lhd}, \emptyset) & \text{if } [\![d_t]\!] \trianglelefteq h \wedge dm = pr \\ (\mathcal{H}, \emptyset, \emptyset) & \text{if } [\![d_t]\!] \not\trianglelefteq h \end{cases}$$

For the special case in which the policy specification is empty, i.e. $P = \emptyset$, there are no requirements. The semantic representation of the empty policy specification is therefore defined as follows: $\forall h \in \mathcal{H} : [\![\emptyset]\!](h) \stackrel{\mathsf{def}}{=} (\mathcal{H}, \emptyset, \emptyset)$.

We refer to $[\![P]\!](h) = (a, u, C)$ as the denotation of the policy specification $P$ with respect to the trace $h$. Composition of policy denotations is formally defined as follows.

**Definition 2.** $(a_1, u_1, C_1) \otimes (a_2, u_2, C_2) \stackrel{\mathsf{def}}{=} (a_1 \cap a_2, u_1 \cup u_2, C_1 \cup C_2)$

Given the policy denotations $(a_1, u_1, C_1)$ and $(a_2, u_2, C_2)$, the sets $a_1$ and $a_2$ represent the obliged behavior. The composition of the policy denotations means that both $a_1$ and $a_2$ are obliged, which explains the composition of these sets using intersection. Since the sets $u_1$ and $u_2$ represent prohibited behavior, the composition using union ensures that both sets remain prohibited after composition. The sets $C_1$ and $C_2$ represent the various behaviors that must be offered as potential alternatives as specified by the permission rules. The union operator ensures that all the alternatives are still represented after the composition

It follows immediately from the properties of associativity and commutativity of $\cap$ and $\cup$ that also the composition operator $\otimes$ is associative and commutative.

Composition of policy specifications is defined by the union operator since policy specifications are given by sets of policy rule specifications. The following defines the semantics of composed policy specifications.

**Definition 3.** $[\![P_1 \cup P_2]\!](h) \stackrel{\mathsf{def}}{=} [\![P_1]\!](h) \otimes [\![P_2]\!](h)$

The definition of the semantics of policy specifications captures the set of traces that are characterized by the policy rules in question. The notion of policy adherence that is introduced next characterizes what it means for a system (specification) to satisfy a policy specification. As we will see, policy adherence means that obligation rules implicitly specify unacceptable behavior also as they limit the set of acceptable behavior. Conversely, prohibition rules for the same reason implicitly specify acceptable behavior also.

## 3 Policy Adherence

The notion of policy adherence formally captures what it means that a system specification satisfies a policy specification, where system specifications are formalized using STAIRS. The reader is referred to Refs.[7, 18] for a detailed presentation of the STAIRS formalization of the sequence diagram syntax and semantics. In the following we present STAIRS by giving the example depicted in Fig.3.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

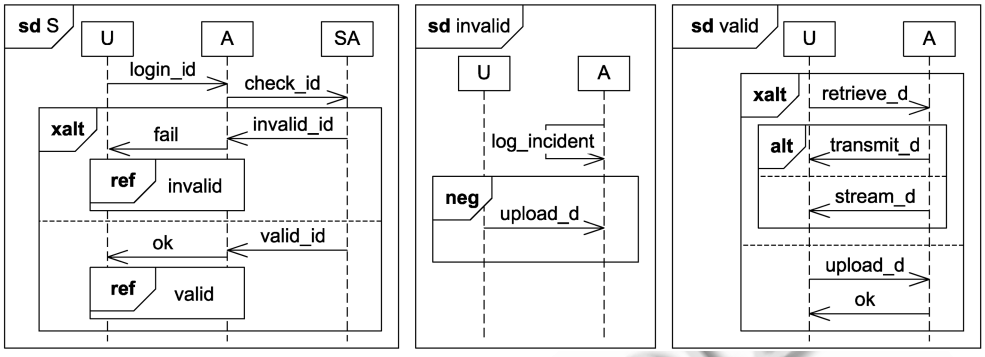



Figure 3. System specification

The diagram $S$ shows parts of the specification of the file sharing system to which the policy exemplified in Fig.2 applies. The full system specification would of course be much more extensive, but we focus only on the part of relevance for the policy. The ref construct specifies an interaction use and is a UML construct for referring to a diagram within the specification of another diagram; the interaction use can equivalently be replaced with the content of the referred diagram.

In addition to specifying behavior that must be offered by the system, sequence diagrams can specify illegal behavior. Sequence diagrams are therefore semantically represented by a pair $(p, n)$ of positive and negative traces, respectively, $p, n \subseteq \mathcal{H}$. Traces that are neither positive nor negative, i.e. $\mathcal{H} \setminus p \cup n$, are referred to as inconclusive. A correct implementation must offer at least one of the positive traces and none of the negative traces. As exemplified in Fig.3, negative behavior can be specified using the neg construct; following an attempt of user login, the security administrator *SA* checks whether the user id is valid. In case of a login failure, the subsequent behavior of the user uploading data to the application is negative. Notice that the behavior preceding the neg construct is positive; it is only the sequential composition of the preceding positive behavior with the subsequent negative behavior that is negative.

An important feature of STAIRS is the distinction between underspecification and so called inherent non-determinism, captured by the alt and xalt constructs, respectively. The alt construct specifies choices between alternatives that are held as equivalent for fulfilling a certain behavior or purpose. The xalt construct on the other hand specifies choices of behavior that each must be offered as potential alternatives by the implementation. The two operands of the xalt in the diagram *valid*, for example, specifies that the user must be offered the choice between retrieving and uploading data. The use of alt in the former operand means that the manner of retrieving data is underspecified.

In order to capture inherent non-determinism, the semantics of sequence diagrams is in STAIRS defined by the function $[\![]\!]$ that for a sequence diagram $d$ yields a set $[\![d]\!] = \{(p_1, n_1), \ldots, (p_m, n_m)\}$ of pairs of trace sets. Each pair is referred to as an interaction obligation to convey the fact that a correct implementation must offer each of them as a potential choice of behavior. Diagram $S$ in Fig.3 is therefore represented by a set of three interaction obligations, one representing the scenario of invalid user login, and two representing the choices between behaviors following a valid login. If

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

the positive and negative traces of an interaction obligation overlap, i.e. if $p \cap n \neq \emptyset$, the traces in the intersection are interpreted as negative. We refer to the set $p \setminus n$ as the implementable traces of the interaction obligation.

Let, now, the policy specification $P$ be given by the set of the three rules depicted in Fig.2. The permission rule *retrieve* states that following a valid user login, the user is permitted to retrieve data from the application. After the event of the user receiving the *ok* message this rule is triggered by the system specification $S$ in Fig.3. Policy adherence requires that data retrieval subsequently must be offered as a potential alternative, which it is by the first operand of the xalt in the diagram *valid*. The prohibition rule *deny* and the obligation rule *log* of Fig.2 constrain the behavior of all potential alternatives following the triggering behavior. The specification $S$ adheres to both these rules by the diagram *invalid*. The system specification $S$ therefore adheres to the policy specification $P$, which is denoted by $P \rightarrow_a S$. The adherence relation is formally defined as follows.

**Definition 4.** Adherence of system specifications $S$ to policy specifications $P$.

$$
\begin{aligned}
P \rightarrow_a S \stackrel{\mathsf{def}}{=} \forall (p,n) \in [\![S]\!] \quad : \quad & \forall h \in (p \setminus n) : \\
& h \in a \wedge \\
& h \notin u \wedge \\
& \forall H \in C : \exists (p',n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H \\
& \text{where } [\![P]\!](h) = (a,u,C)
\end{aligned}
$$

The first and second conjuncts ensure that the trace $h$ adheres to the obligation rules and prohibition rules, respectively, of $P$. The third conjunct ensures adherence to the permission rules of $P$ by requiring that the behavior specified by each of these rules is offered as a potential choice by $S$.

By the definition of adherence, the implementable traces $p \setminus n$ of each interaction obligation of the system specification must be within the acceptable behavior $a$ as specified by the obligation rules of the policy specification. The obligation rules therefore implicitly specify unacceptable behavior also by the complement of $a$. Conversely, the prohibition rules require that the behavior must be within the complement of the unacceptable behavior $u$. These properties of the adherence relation compare to the inter-definability axioms of deontic logic where the various deontic modalities can be defined in terms of each other. The reader is referred to Ref.[24] for a formal account of the relations between the deontic operators in Deontic STAIRS.

Policy adherence is modular in the sense that a system adheres to a set of policy specifications if and only if the system adheres to the composition of the policy specifications. This is expressed by the following theorem, and implies that the problem of verification of adherence can be broken down into sub-problems.

**Theorem 1.** $P_1 \rightarrow_a S \wedge P_2 \rightarrow_a S \Leftrightarrow (P_1 \cup P_2) \rightarrow_a S$

## 4 Policy Refinement

In this section we formally define the notion of policy refinement as a relation between policy specifications. For policy denotations $(a,u,C)$, refinement is defined as follows.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

**Definition 5.** Refinement of policy denotations.

$$
\begin{aligned}
(a_1, u_1, C_1) \rightsquigarrow (a_2, u_2, C_2) \quad \stackrel{\mathsf{def}}{=} \quad & a_2 \subseteq a_1 \wedge \\
& u_1 \subseteq u_2 \wedge \\
& \forall H_1 \in C_1 : \exists H_2 \in C_2 : H_2 \subseteq H_1 \wedge \\
& \forall H_2 \in C_2 : \exists H_1 \in C_1 : H_2 \subseteq H_1
\end{aligned}
$$

Refinement of a policy specification $P_1$ to a policy specification $P_2$ then means that for all traces $h \in \mathcal{H}$, the denotation of $P_2$ with respect to $h$ is a refinement of the denotation of $P_1$ with respect to $h$, formally defined as follows.

**Definition 6.** $P_1 \rightsquigarrow P_2 \stackrel{\mathsf{def}}{=} \forall h \in \mathcal{H} : [\![P_1]\!](h) \rightsquigarrow [\![P_2]\!](h)$

The first and second conjuncts of Def. 5 mean that the requirements imposed by the obligation rules and prohibition rules of the policy specifications are strengthened under policy refinement. The strengthening may stem from reduction of underspecification in existing rules, or from the addition of new policy rules. The third and fourth conjuncts of Def.5 mean that all behaviors that are required to be offered potentially by permission rules of $P_1$ are also required by permission rules of $P_2$ and vice versa. The behavior described by the rule body of the permission rules may also be subject to reduction of underspecification.

Under this notion of policy refinement, the variation over potential choices of behavior that is required by the policy is fixed under refinement. The reduction of underspecification, however, means that traces that are admissible at the abstract level may be inadmissible at the refined level.

Consider again the policy specification $P$ consisting of the three rules in Fig.2. This specification can be refined, for example, by removing the alternative *transmit_ d* from the body of the permission rule *retrieve*. The refined policy rule is shown by the diagram *retrieve2* in Fig.4.

A further refinement option is to add alternatives to the body of the prohibition rule using alt, for example to specify that also data retrieval should be prohibited. This is exemplified by the diagram *deny2* of Fig.4 which is a refinement of the prohibition rule *deny* of Fig.2.

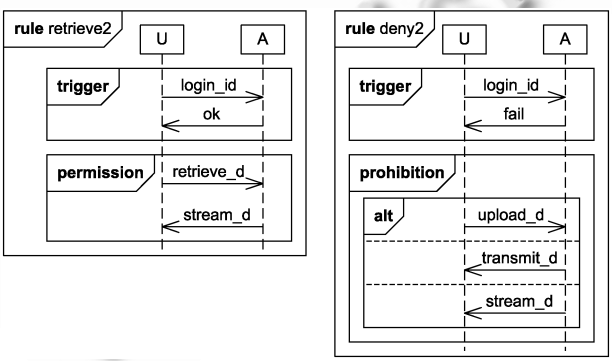



Figure 4. Refined policy rules

It follows immediately from Def. 5 and Def. 6 that the policy refinement relation is reflexive and transitive. A modularity property of the policy refinement relation

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

is that a policy specification $P$ can be refined by refining subsets of $P$ separately, as expressed by the following theorem.

**Theorem 2.** $P_1 \rightsquigarrow P_1' \wedge P_2 \rightsquigarrow P_2' \Rightarrow (P_1 \cup P_2) \rightsquigarrow (P_1' \cup P_2')$

By Theorem 2, the lowest level of granularity for modular refinement of policy specifications is to refine singleton sets of rules, i.e. to refine policy rules. In our example of Fig.2, the policy specification is the set $P = \{retrieve, deny, log\}$. Replacing the former two with the rules of Fig.4 we obtain the policy specification $P_2 = \{retrieve2, deny2, log\}$. Because we have the refinements *retrieve* $\rightsquigarrow$ *retrieve2* and *deny* $\rightsquigarrow$ *deny2*, we get by Theorem 2 the policy refinement $P \rightsquigarrow P_2$.

In Ref.[24] we showed that each of a set of sequence diagram composition operators is monotonic with respect to refinement of policy rules, which implies that a policy rule, and therefore a policy specification, can be refined by refining individual parts of the trigger and rule body separately.

Since refinement of policy specifications is a strengthening of the requirements imposed by the policy specification, the requirements from the abstract levels are preserved under refinement. The enforcement of a concrete policy specification therefore implies the enforcement of the previous, more abstract specifications.

**Theorem 3.** $P_1 \rightsquigarrow P_2 \wedge P_2 \rightarrow_a S \Rightarrow P_1 \rightarrow_a S$

## 5 A Characterization of Adherence Preserving Refinements

In this section we give a general characterization of conditions under which adherence is preserved in the case of the combined refinement of policy specifications and system specifications. Subsequently, in Section 6 we identify refinement rules that fulfill these conditions, such that the application of the rules in the development process guarantees preservation of adherence.

Refinement of system specification in STAIRS allows reduction of underspecification, but requires the preservation of the inherent non-determinism captured by the variation over interaction obligations $(p, n)$. The property of preservation of inherent non-determinism ensures that policy adherence is preserved under refinement of system specifications.

Reduction of underspecification in system specifications is achieved by redefining previously positive traces as negative, or by redefining previously inconclusive traces as negative. By $(p, n) \rightsquigarrow (p', n')$ we denote that the interaction obligation $(p', n')$ is a refinement of the interaction obligation $(p, n)$, formally defined as follows.

$$(p, n) \rightsquigarrow (p', n') \stackrel{\mathsf{def}}{=} (n \subseteq n') \wedge (p \subseteq p' \cup n') \wedge (p' \subseteq p)$$

For sequence diagrams $d$ and $d'$, refinement is defined as follows.

**Definition 7.**

$$\begin{aligned} d \rightsquigarrow d' \quad \stackrel{\mathsf{def}}{=} \quad & \forall o \in [\![d]\!] : \exists o' \in [\![d']\!] : o \rightsquigarrow o' \wedge \\ & \forall o' \in [\![d']\!] : \exists o \in [\![d]\!] : o \rightsquigarrow o' \end{aligned}$$

It follows from the definition that the refinement relation is reflexive and transitive. Modular development is also supported by monotonicity of several sequence diagram operators with respect to refinement[18].

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

In Ref.[19] we showed that adherence is preserved under refinement of system specifications for a fixed policy specification, as expressed by the following.

**Theorem 4.** $P \rightarrow_a S_1 \wedge S_1 \rightsquigarrow S_2 \Rightarrow P \rightarrow_a S_2$

We explained above that the system specification $S$ of Fig.3 adheres to the policy specification $P$ of Fig.2. The system specification may be refined for example by redefining the *transmit_ d* operand of the alt as negative. This is shown by the sequence diagram $S2$ of Fig.5. By replacing the interaction use *valid* in Fig.3 with that of *valid2* of Fig.5, the *transmit_ d* option is now characterized as negative by the refuse construct. This construct is a STAIRS operator for specifying negative behavior. For a sequence diagram $d$ such that $[\![d]\!] = (p, n)$, the semantics is defined by $[\![\mathsf{refuse}\ d]\!] \stackrel{\mathsf{def}}{=} (\emptyset, p \cup n)$.

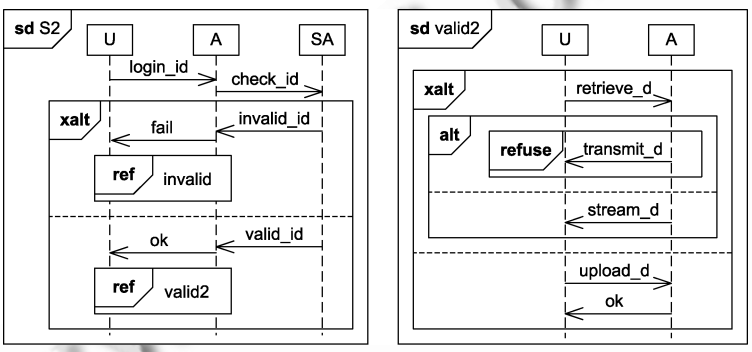



Figure 5. Refined system specification

Since the remaining *stream_ d* alternative is a valid fulfillment of the body of the permission rule *retrieve*, the refined system specification still adheres to the policy specification. Generally, when the policy specification is fixed, adherence is preserved under refinement of system specifications since the reduction of allowed behavior never introduces a policy breach.

When the policy specification is not fixed, policy adherence is not preserved under refinement in the general case. However, because policy refinement implies only a reduction of the admissible behavior as specified by the policy, adherence of a system specification is preserved under refinement by reducing the positive behavior of the system specification accordingly. In other words, if $P_1 \rightarrow_a S_1$ has been established at the abstract level and the policy refinement $P_1 \rightsquigarrow P_2$ is conducted, a system specification $S_2$ can be derived from $S_1$ based on the reduction of admissible behavior when shifting from $P_1$ to $P_2$ such that $S_1 \rightsquigarrow S_2$ and $P_2 \rightarrow_a S_2$.

Policy adherence is obviously preserved if the requirements imposed by the refined policy specification is equivalent to the requirements imposed by the abstract specification. Under this condition, preservation of adherence under refinement can be formulated as follows.

$$\frac{\forall h \in H : [\![P_1]\!](h) = [\![P_2]\!](h)}{P_1 \rightarrow_a S_1 \wedge S_1 \rightsquigarrow S_2 \Rightarrow P_2 \rightarrow_a S_2} \tag{5.1}$$

Notice that the premise implies $P_1 \rightsquigarrow P_2$ by definition. Under this condition, there is no reduction of admissible behavior, and adherence is preserved by Theorem 4. The modularity properties of policy adherence and policy refinement captured

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

by Theorem 1 and Theorem 2, respectively, can be utilized by identifying parts of the refined policy specification that is semantically equivalent to the abstract policy specification or a subset of the abstract policy specification. If, for example, $P_1 \cup P_2 \rightsquigarrow P_1' \cup P_2'$ and $\forall h \in \mathcal{H} : [\![P_1]\!](h) = [\![P_1']\!](h)$, policy adherence of a system specification $S$ needs only be checked with respect to $P_2'$ if $P_1 \cup P_2 \rightarrow_a S$ has been verified.

For policy refinements $P_1 \rightsquigarrow P_2$ in which the semantics of the policy specifications are not equivalent, and $P_1 \rightarrow_a S$ has been verified, we need to identify the traces $h$ of the system specification $S$ that are admissible under $P_1$ and inadmissible under $P_2$. A refinement $S \rightsquigarrow S'$ of the system specification by eliminating these traces $h$ from $S$ then ensures adherence at the refined level, i.e. $P_2 \rightarrow_a S'$. We let $\mathcal{D}$ denote the set of all sequence diagrams, and define the function

$$\Delta(_, _, _) \in \mathbb{P}(\mathcal{R}) \times \mathbb{P}(\mathcal{R}) \times \mathcal{D} \rightarrow \mathbb{P}(\mathcal{H})$$

that takes the policy specifications $P_1$ and $P_2$ and the system specification $S$ as operands and yields the set $H$ of positive traces from $S$ that are admissible under $P_1$ and inadmissible under $P_2$. Formally, the function $\Delta$ is defined as follows.

**Definition 8.** For policy specifications $P_1$ and $P_2$ and system specifications $S$ such that $P_1 \rightsquigarrow P_2$ and $P_1 \rightarrow_a S$:

$$\begin{aligned}\Delta(P_1, P_2, S) \stackrel{\mathsf{def}}{=} \{h \in \mathcal{H} \mid\ & (\exists (p, n) \in [\![S]\!] : h \in (p \setminus n)) \wedge \\ & (h \in (a_1 \setminus a_2) \vee \\ & \ h \in (u_2 \setminus u_1) \vee \\ & \ \exists H_1 \in C_1 : \exists H_2 \in C_2 : H_2 \subseteq H_1 \wedge h \in (H_1 \setminus H_2))\} \\ \text{where } [\![P_1]\!](h) = (a_1, u_1, C_1) & \text{ and } [\![P_2]\!](h) = (a_2, u_2, C_2)\end{aligned}$$

The singleton set $\{(\emptyset, \Delta(P_1, P_2, S))\}$ of interaction obligations can be understood as the semantic representation of a sequence diagram that specifies as negative the behavior that should be inadmissible after the refinement of the system specification. We denote this representation by $[\![S]\!]^{\Delta(P_1, P_2, S)}$, abbreviated by $[\![S]\!]^{\Delta}$ when the given policy specifications are irrelevant or clear from the context.

For a system specification $S$ such that $P_1 \rightarrow_a S$ and $P_1 \rightsquigarrow P_2$ hold, the following represents a refined system specification in which the inadmissible behavior $[\![S]\!]^{\Delta}$ has been removed: $[\![S]\!]^{\Delta} \uplus [\![S]\!]$. The operator $\uplus$ takes two sets of interaction obligations as operands and yields their inner union, formally defined as follows.

$$O_1 \uplus O_2 \stackrel{\mathsf{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2\}$$

Since $[\![S]\!]^{\Delta}$ is a singleton set of interaction obligations, the result of $[\![S]\!]^{\Delta} \uplus [\![S]\!]$ is equal to the result of adding the set $\Delta(P_1, P_2, S)$ to the negative traces of each interaction obligation of $[\![S]\!]$. More formally,

$$[\![S]\!]^{\Delta} \uplus [\![S]\!] = \{(p, n \cup \Delta(P_1, P_2, S)) \mid (p, n) \in [\![S]\!]\}$$

The result of this composition with inner union is furthermore a refinement of $[\![S]\!]$ as expressed by the next theorem.

**Theorem 5.** For all interaction obligations $(\emptyset, n)$ and all sets of interaction obligations $O$: $O \rightsquigarrow \{(\emptyset, n)\} \uplus O$

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

The refinement $[\![S]\!]^{\Delta} \uplus [\![S]\!]$ of $[\![S]\!]$ then characterizes a system specification where the adherence $P_1 \rightarrow_a S$ is preserved when the policy refinement $P_1 \rightsquigarrow P_2$ has been conducted. The result is expressed by the following theorem.

**Theorem 6.** $P_1 \rightarrow_a S \wedge P_1 \rightsquigarrow P_2 \Rightarrow P_2 \rightarrow_a ([\![S]\!]^{\Delta} \uplus [\![S]\!])$

Notice that adherence as formalized in Def.4 is a relation between a policy specification and a system specification, whereas $([\![S]\!]^{\Delta} \uplus [\![S]\!])$ in Theorem 6 denotes a set of interaction obligations. The formulation of Theorem 6 is for sake of brevity and readability.

The result of adherence preservation under the condition that the abstract system specification is $S$ is refined by the composition with $[\![S]\!]^{\Delta}$ can then be formulated as follows.

$$\frac{[\![S_2]\!] = [\![S_1]\!]^{\Delta} \uplus [\![S_1]\!]}{P_1 \rightarrow_a S_1 \wedge P_1 \rightsquigarrow P_2 \Rightarrow P_2 \rightarrow_a S_2} \tag{5.2}$$

So far we have only given a general characterization of the refined system specification for which adherence is preserved by describing its semantics. In a practical setting, the development process should, however, be supported by syntactical rules that ensure the desired refinements.

The trace set $\Delta(P_1, P_2, S)$ which is derived as expressed in Def.8 characterizes precisely the traces in $S$ that are inadmissible after the policy refinement $P_1 \rightsquigarrow P_2$. By identifying a sequence diagram $d$ where these inadmissible traces are negative, the diagram $d$ can be syntactically composed with $S$ to yield the desired result. More precisely, if $[\![d]\!] = [\![S]\!]^{\Delta}$ we have $[\![d \ \mathsf{alt} \ S]\!] = [\![S]\!]^{\Delta} \uplus [\![S]\!]$, since the $\mathsf{alt}$ operator is formalized by $\uplus$ in STAIRS. Given the system specification $S$ and the identified sequence diagram $d$, the specification $d \ \mathsf{alt} \ S$ then denotes the desired refinement of the system specification. Theorem 6 therefore means that in principle, the desired system refinement can be conducted.

In the following we describe strategies for identifying such diagrams $d$, and we identify rules for refinement of system specifications that guarantee adherence preservation.

## 6 Adherence Preserving Refinement Rules

Our strategy is to utilize the modularity properties of policy adherence and policy refinement as expressed by Theorem 1 and Theorem 2, respectively. These properties allow the problem of preservation of adherence under policy refinement to be addressed as a problem of preservation of adherence under policy rule refinement. Given a refinement of a policy rule $r$ to a policy rule $r'$, the challenge is to identify a sequence diagram $d$ that characterizes the strengthening of the policy rule under the refinement step. Ideally, the identified sequence diagram $d$ should capture exactly the set of traces that characterizes the strengthening of the policy rule. In a practical setting it may, however, be infeasible to identify the precise sequence diagram because of limitations in the expressiveness of UML sequence diagrams. Since there exist sets $H \subseteq \mathcal{H}$ of well formed traces for which there are no syntactical representations, it may be that the traces representing the strengthening cannot be precisely specified. The sequence diagram $d$ is in the sequel therefore required to characterize at least the strengthening of the policy rule, i.e. $[\![d]\!]$ must be a superset of the desired trace set.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

Since the removal of traces from the system specification never introduces a policy breach, this requirement is sufficient for the results of adherence preservation.

The formula for strengthening a system specification $S$ into a system specification $S'$ with respect to an identified sequence diagram $d$ is the following.

$$S' = \mathsf{refuse}(d \ \mathsf{par} \ \mathsf{any}) \ \mathsf{alt} \ S$$

The par construct is for parallel composition and is formalized by the $\|$ operator which yields all the possible interleavings of is operands as result. The sequence diagram any denotes the maximal sequence diagram the semantics of which is defined by $[\![\mathsf{any}]\!] = (\mathcal{H}, \emptyset)$.[2)], The reason for the parallel composition of the sequence diagram $d$ and the maximal sequence diagram any in the formula is that it yields all the super-traces of traces in $[\![d]\!]$, i.e. the traces that fulfill the behavior specified by $d$. This is formally expressed by the following result: $\forall H \subseteq \mathcal{H} : H{\lhd} = (H \parallel \mathcal{H})$.

The fulfillment of the requirement that $\mathsf{refuse}(d \ \mathsf{par} \ \mathsf{any}) \ \mathsf{alt} \ S$ is a refinement of $S$ is shown by the following theorem.

**Theorem 7.** For all sequence diagrams $d$ and $d'$ such that $[\![d']\!]$ is singleton: $d \rightsquigarrow (\mathsf{refuse}(d') \ \mathsf{alt} \ d)$

We now turn to the approach for adherence preserving refinement of each kind of policy rule. A permission rule $(pe, d_t, d_b)$ is refined by a permission rule $(pe, d_t, d'_b)$ if $[\![d'_b]\!] \subseteq [\![d_b]\!]$. By identifying sequence diagrams $d_{b1}$ and $d_{b2}$ such that $[\![d_b]\!] = [\![d_{b1} \ \mathsf{alt} \ d_{b2}]\!]$ and $d'_b = d_{b1}$, the sequence diagram $d_{b2}$ represents the reduction of the admissible behavior after the refinement. By refining a system specification $S$ for which adherence to $(pe, d_t, d_b)$ has been verified by characterizing traces fulfilling $d_t \ \mathsf{seq} \ d_{b2}$ as negative, the adherence result that was established at the abstract level is preserved.

In this case, the singleton rule set $\{(pe, d_t, d_{b1} \ \mathsf{alt} \ d_{b2})\}$ is the abstract policy specification and $\{(pe, d_t, d_{b1})\}$ the refined policy specification. Preservation of adherence under this approach can then be formulated as follows, where the premise implies that $S_1 \rightsquigarrow S_2$ by Theorem 7.

$$\frac{S_2 = \mathsf{refuse}((d_t \ \mathsf{seq} \ d_{b2}) \ \mathsf{par} \ \mathsf{any}) \ \mathsf{alt} \ S_1}{\{(pe, d_t, d_{b1} \ \mathsf{alt} \ d_{b2})\} \rightarrow_a S_1 \Rightarrow \{(pe, d_t, d_{b1})\} \rightarrow_a S_2} \tag{6.3}$$

The policy rule *retrieve* of Fig. 2, for example, may be refined by removing the downloading alternative *transmit_ d* from the rule body. In that case adherence of the system specification $S$ of Fig. 3 is lost since *transmit_ d* is positive. Refining $S$ by applying the identified rule, however, removes this alternative from the system specification, and adherence is restored at the refined level.

Notice that for the sequence diagram $d_{b2}$ to characterize the exact strengthening of the permission rule, the requirement $[\![d_{b1}]\!] \cap [\![d_{b2}]\!] = \emptyset$ must be fulfilled. The result still holds if this requirement is not fulfilled, but implies that the removal of traces under the refinement of the system specification is wider than required for adherence to be ensured.

The same approach to adherence preservation is applicable to obligation rules, as captured by the following.

---

2) The name any is adopted from Ref.[9] where it denotes the maximal MSC[8].

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

$$\frac{S_2 = \mathsf{refuse}((d_t \ \mathsf{seq}\ d_{b2})\ \mathsf{par\ any})\ \mathsf{alt}\ S_1}{\{(ob, d_t, d_{b1}\ \mathsf{alt}\ d_{b2})\} \rightarrow_a S_1 \Rightarrow \{(ob, d_t, d_{b1})\} \rightarrow_a S_2} \tag{6.4}$$

A prohibition rule may be refined by increasing the set of traces that represents the prohibited behavior as specified by the rule body, i.e. a prohibition rule $(pr, d_t, d_b)$ is refined by a prohibition rule $(pr, d_t, d'_b)$ if $[\![d'_b]\!] \supseteq [\![d_b]\!]$. By identifying sequence diagrams $d_{b1}$ and $d_{b2}$ such that $d_b = d_{b1}$ and $[\![d'_b]\!] = [\![d_{b1}\ \mathsf{alt}\ d_{b2}]\!]$, the sequence diagram $d_{b2}$ represents the reduction of the admissible behavior after the refinement. By refining a system specification $S$ for which adherence to $(pr, d_t, d_b)$ has been verified by characterizing traces fulfilling $d_t\ \mathsf{seq}\ d_{b2}$ as negative, the adherence result that was established at the abstract level is preserved.

$$\frac{S_2 = \mathsf{refuse}((d_t \ \mathsf{seq}\ d_{b2})\ \mathsf{par\ any})\ \mathsf{alt}\ S_1}{\{(pr, d_t, d_{b1})\} \rightarrow_a S_1 \Rightarrow \{(pr, d_t, d_{b1}\ \mathsf{alt}\ d_{b2})\} \rightarrow_a S_2} \tag{6.5}$$

As for the rules for preservation of adherence under refinement of permissions and obligations, the identification of the sequence diagram that characterizes the exact strengthening requires that $[\![d_{b1}]\!] \cap [\![d_{b2}]\!] = \emptyset$ holds. However, since $d_{b1}$ is already characterized as inadmissible by $S$ since adherence to $\{(pr, d_t, d_{b1})\}$ has been verified, the strengthening of $S$ by removing the traces that fulfill $d_t\ \mathsf{seq}\ d_{b2}$ yields precisely the desired refinement of $S$.

Returning to the policy example of Fig.2, the prohibition rule *deny* may e.g. be refined by replacing the *upload_ d* message with the composition of the messages *upload_ d* and *transmit_ d* using alt. Adherence of the system specification $S$ of Fig.3 to the policy specification is then lost, since $S$ specifies only *upload_ d* as negative following login failure. By applying the identified rule the desired strengthening of $S$ is, however, imposed and adherence restored.

By properties of modularity, the approaches for adherence preserving refinement of policy rules stated in (6.3), (6.4) and (6.5) can be combined into an approach for adherence preserving refinement of policies. Assume, for example, that $P_1 \rightarrow_a S_1$ has been verified at an abstract level and that $P_1 = \{r_1, \ldots, r_m\}$ is refined following the above described patterns. For each rule $r_i \in P_1$, let $d_i$ denote the sequence diagram characterizing the strengthening of $r_i$. The desired strengthening of $S_1$ into $S_2$ such that $P_2 \rightarrow_a S_2$ can then be obtained by the following formula.

$$S_2 = \mathsf{refuse}(d_1\ \mathsf{par\ any})\ \mathsf{alt}\ \ldots\ \mathsf{alt}\ \mathsf{refuse}(d_m\ \mathsf{par\ any})\ \mathsf{alt}\ S_1$$

The modularity properties also allow the approach to be applied to a subset of the policy specification. For a policy specification $P_1 \cup P$ such that $P_1 \cup P \rightarrow_a S_1$, we have $P_1 \rightarrow_a S_1$ and $P \rightarrow_a S_1$ by Theorem 1. By refining the rules of $P_1$ into $P_2$ following the above pattern, the resulting policy specification $P_2 \cup P$ is a refinement of $P_1 \cup P$ by Theorem 2. Since $S_1 \rightsquigarrow S_2$ for the derived system specification $S_2$, and $P \rightarrow_a S_1$, we get $P \rightarrow_a S_2$ by Theorem 4. By Theorem 1 we finally have $P_2 \cup P \rightarrow_a S_2$.

## 7 Laudatio: The Relationship between STAIRS and Focus

The second author of this paper, Ketil Stølen, had the great pleasure of working closely with Manfred Broy while being a member of Broy's research group in Munich from 1991 until 1996. The main achievement of their collaboration was the Focus

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

book published in 2001[4]. The understanding and experience gained through the years of research leading to the publication of this book has greatly influenced Stølen and his collaboration partners in their work on the STAIRS method and its various specializations[7,10,14,18,19]. As a tribute to Manfred Broy we will in the following outline how the approach presented in this paper builds on the ideas and principles embedded in Focus. We first draw the lines from STAIRS to Focus; then we look at the relationship between the latter and the specific features of Deontic STAIRS.

### *7.1 The relationship to STAIRS*

Focus is similar to STAIRS in that the semantics of a specification refers to the messages that may be passed between system entities in order to fulfill certain functionality. A further similarity is the support for abstraction, as well as system development under stepwise and modular refinement. In Focus, a specification $S_2$ is a refinement of $S_1$ if the specifications have the same syntactic interface, and—for each input history—any output history of $S_2$ is also an output history of $S_1$. This notion of refinement corresponds to refinement by reduction of underspecification in STAIRS.

Focus and STAIRS are also similar in that a specification may be viewed as a predicate on sequences. In Focus the sequences are input and output histories while STAIRS operates on traces which basically are interleavings of input and output histories.

From a semantic point of view the main difference between Focus and STAIRS as described in Ref.[4] and Ref.[7], respectively, is that STAIRS is more general in the following two respects.

**STAIRS may express incomplete specifications.** STAIRS has been developed to support sequence diagrams. Sequence diagrams are special in the sense that they are used to express incomplete specifications like example runs. Semantically, STAIRS therefore distinguishes between positive, negative and incomplete behaviors. The positive ones are those behaviors the sequence diagram explicitly allows, i.e. the example runs, the negative ones are the runs the sequence diagrams explicitly disallows, while the inconclusive behaviors are those the sequence diagram does not describe. In Focus the behaviors captured by a specification are positive, and those that are not are negative. Hence, Focus—like most other specification languages—has been designed to capture complete specifications.

**STAIRS may express trace set properties.** For system development under refinement, trace set properties may be problematic if there is no support for distinguishing between underspecification and inherent non-determinism. Refinement typically reduces underspecification, and thereby also non-determinism. If there is no support for distinguishing the non-determinism that can be reduced from the non-determinism that should be preserved, we have no guarantee that trace set properties are preserved. Therefore, although a trace set property is verified at an abstract level, the property may not hold for a system obtained by refinement from the abstract specification. In STAIRS, it is precisely the support for capturing inherent non-determinism—syntactically by the xalt

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

construct and semantically by the variation over interaction obligations—and preserving inherent non-determinism under refinement that ensures adherence preservation.

Focus offers support for capturing trace properties such as safety and liveness, but there is not explicit support for capturing trace set properties and preserving these under refinement. This corresponds to STAIRS without the xalt construct. Semantically, each interaction obligation corresponds to a single Focus specification, and the semantics of a STAIRS specification, a set of interaction obligations, corresponds to a set of Focus specifications.

### *7.2 The relationship to deontic STAIRS*

In the setting of Deontic STAIRS and the problem of preservation of policy adherence under refinement, the distinction between trace properties and trace set properties is important[19]. Trace properties are properties that can be falsified on single traces, and include safety and liveness properties. Trace set properties, on the other hand, are properties that can be falsified on sets of traces only, and include information flow properties. In order to show adherence of a system specification to a Deontic STAIRS policy specification, we need to show adherence to permissions, obligations and prohibitions. Adherence to the latter two is a trace property since they can be falsified by a single trace, whereas adherence to permissions is a trace set property.

## 8 Conclusion and Related Work

In this paper we have presented a method based on UML sequence diagram for the integration of policy requirements into the overall system development process. Both policy requirements and requirements to system design and functionality can be specified at various levels of abstraction. For analysis of abstract specifications to be meaningful, however, the results must be preserved under refinement; otherwise the analysis must be conducted from scratch after each refinement step. The approach presented in this paper allows policy adherence to be taken into account throughout the development process by identifying the conditions under which adherence may be preserved under refinement. The practical development setting is furthermore supported by the identification of explicit development rules the application of which guarantees adherence preservation. In the following we present and discuss some related work.

Live sequence charts (LSCs)[5] extend message sequence charts[8] and are particularly directed towards specifying liveness properties. LSCs are related to Deontic STAIRS by the support for specifying conditional scenarios using triggering precharts. The distinction between universal diagrams that must be satisfied by all system runs and existential diagrams that must be satisfied by at least one system run can furthermore be utilized to express obligations and permissions, whereas the expressiveness of LSCs to capture forbidden scenarios can be utilized to express prohibitions. Existential diagrams can, however, not be specified as conditional scenarios. System development using LSCs is intended to undergo a shift from existential to universal diagrams, but a precise or formal notion of refinement is not provided. Modal se-

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

quence diagrams (MSDs)[6] are defined as a UML 2.0 profile, and are based on the universal/existential distinction of LSCs. MSDs also allow the specification of conditional scenarios, but the approach is not supported by a formal notion of refinement.

Triggered message sequence charts (TMSCs)[20] allow the specification of conditional scenarios and are supported by a formal notion of refinement. The distinction between the internal choice and delayed choice operators is related to the distinction between alt and xalt in STAIRS, and could perhaps be utilized to capture policy adherence. The paper stresses the importance of preserving properties such as safety and liveness under refinement, but the problem of capturing and preserving inherent non-determinism is not discussed. Support for the specification of negative or prohibited behavior is furthermore not provided.

Model Driven Security is Ref.[2] a UML-based approach to the integration of security models into the overall system development process. A UML-based language called SecureUML is presented that supports the specification of access control requirements. The paper addresses only access control in the form of permission rules, and the specification of other security aspects is pointed out as a direction for future work. Refinement is also not addressed, but the authors state that by considering diagrams such as UML sequence diagrams and UML use case diagrams, the modeling of the system from different views and at different abstraction levels would be supported.

As mentioned in the introduction, the problem of policy refinement is poorly explored in the research on policy-based management. The investigation of policy refinement has gained interest only recently[3], and the literature on the issue is still scarce. One aspect of policy refinement as proposed in Ref.[12] is that of goal refinement, where the set of low-level goals derived from a high-level goal intends to fulfill the latter. Goal refinement has been adopted by approaches to policy refinement that focus on the problem of deriving low-level policies the enforcement of which ensures the fulfillment of the initial high-level goals[1,15]. Adherence to refined policy specifications therefore guarantees adherence to the abstract policy specifications, but the problem of preservation of adherence under refinement is not discussed. The problem of integrating requirements from policy specifications with system specifications, or understanding the relation between policy specifications and system specifications where both may be represented at various abstraction levels, is also not addressed.

## Acknowledgments

The research on which this paper reports has partly been funded by the European Commission through the NESSoS network of excellence (contract no.256980).

## References

[1] Bandara AK, Lupu EC, Russo A, Dulay N, Sloman M, Flegkas P, Charalambides M, Pavlou G. Policy Refinement for DiffServ Quality of Service Management. Proc. of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM'05). 2005. 469–482.

[2] Basin D, Doser J, Lodderstedt T. Model driven security: from UML models to access control infrastructures. ACM Trans. on Software Engineering and Methodology, 2006, 15(1): 39–91.

[3] Boutaba R, Aib I. Policy-based management: a historical perspective. Journal of Network and Systems Management, 2007, 15(4): 447–480.

[4] Broy M, Stølen K. Specification and Development of Interactive Systems: FOCUS on Streams,

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

Interfaces and Refinement. Springer, 2001.
[5] Damm W, Harel D. LSCs: Breathing life into message sequence charts. Formal Methods in System Design, 2001, 19(1): 45–80.
[6] Harel D, Maoz S. Assert and negate revisited: Modal Semantics for UML Sequence Diagrams. Software and Systems Modeling, 2007, 7(2): 237–252.
[7] Haugen Ø, Husa KE, Runde RK, Stølen K. STAIRS towards formal design with sequence diagrams. Software and Systems Modeling, 2005, 4(4): 355–367.
[8] International Telecommunication Union. Recommendation Z.120 – Message Sequence Chart (MSC), 2004.
[9] Krüger IH. Distributed System Design with Message Sequence Charts [Ph.D. Thesis]. Institut für Informatik, Ludwig-Maximilians-Universität München, July 2000.
[10] Lund MS, Stølen K. A fully general operational semantics for UML 2.0 sequence diagrams with potential and mandatory choice. 14th International Symposium on Formal Methods (FM'06). LNCS 4085, Springer, 2006. 380–395.
[11] Mantel H. Preserving information flow properties under refinement. Proc. the IEEE Symposium on Security and Privacy (SP'01). IEEE Computer Society, 2001. 78–91.
[12] Moffett JD, Sloman MS. Policy hierarchies for distributed systems management. IEEE Journal on Selected Areas in Communications, 1993, 11: 1404–1414.
[13] Object Management Group. Unified Modeling Language: Superstructure, version 2.1.1, 2007.
[14] Refsdal A, Runde RK, Stølen K. Underspecification, inherent nondeterminism and probability in sequence diagrams. Proc. the 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS). LNCS 4037, Springer, 2006. 138–155.
[15] Rubio-Loyola J, Serrat J, Charalambides M, Flegkas P, Pavlou G. A functional solution for goal-oriented policy refinement. Proc. of the 7th International Workshop on Policies for Distributed Systems and Networks (POLICY'06). IEEE Computer Society, 2006. 133–144.
[16] Rumbaugh J, Jacobson I, Booch G. The Unified Modeling Language Reference Manual. Second edition, Addison Wesley, 2005.
[17] Runde RK. STAIRS – Understanding and Developing Specifications Expressed as UML Interaction Diagrams. Faculty of Mathematics and Natural Sciences [Ph.D. Thesis]. University of Oslo, 2007.
[18] Runde RK, Haugen Ø, Stølen K. Refining UML interactions with underspecification and nondeterminism. Nordic Journal of Computing, 2005, 12(2): 157–188.
[19] Seehusen F, Solhaug B, Stølen K. Adherence preserving refinement of trace-set properties in STAIRS: exemplified for information flow properties and policies. Software and Systems Modeling, 2009, 8(1): 46–65.
[20] Sengupta B, Cleaveland R. Triggered message sequence charts. IEEE Transactions on Software Engineering, 2006, 32(8): 587–607.
[21] Sloman M. Policy driven management for distributed systems. Journal of Network and Systems Management, 1994, 2: 333–360.
[22] Sloman M, Lupu E. Security and management policy specification. Network. IEEE, 2002, 16(2): 10–19.
[23] Solhaug B. Policy Specification Using Sequence Diagrams – Applied to Trust Management [Ph.D. Thesis]. Faculty of Social Sciences, University of Bergen, 2009.
[24] Solhaug B, Stølen K. Compositional refinement of policies in UML – exemplified for access control. Proc. of the 13th European Symposium on Research in Computer Security (ESORICS'08). LNCS 5283, Springer, 2008. 300–316.
[25] Solhaug B, Stølen K. Preservation of Policy Adherence under Refinement. Technical Report A11358, SINTEF, 2009.
[26] Wing JM. A specifier's introduction to formal methods. IEEE Computer, 1990, 23(9): 8, 10–22, 24.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved