SINTEF

SINTEF ICT

Address: NO-7465 Trondheim, NORWAY
Location: Forskningsveien 1
Telephone: +47 22 06 73 00
Fax: +47 22 06 73 50

Enterprise No.: NO 948 007 029 MVA

SINTEF REPORT

TITLE

Compositional Refinement of Policies in UML – Exemplified for Access Control

AUTHOR(S)

Bjørnar Solhaug and Ketil Stølen

CLIENT(S)

REPORT NO.	CLASSIFICATION	CLIENTS REF.	
SINTEF A11359	Open		
CLASS. THIS PAGE	ISBN	PROJECT NO.	NO. OF PAGES/APPENDICES
Open	978-82-14-04436-2	90B22000 90B245	33/2
ELECTRONIC FILE CODE		PROJECT MANAGER (NAME, SIGN.) Ketil Stølen	CHECKED BY (NAME, SIGN.) Gyrd Brændeland
FILE CODE	DATE	APPROVED BY (NAME, POSITION, SIGN.)	
	2009-03-25	Bjørn Skjellaug	

ABSTRACT

The UML is the de facto standard for system specification, but offers little specialized support for the specification and analysis of policies. This paper presents Deontic STAIRS, an extension of the UML sequence diagram notation with customized constructs for policy specification. The notation is underpinned by a denotational trace semantics. We formally define what it means that a system satisfies a policy specification, and introduce a notion of policy refinement. We prove that the refinement relation is transitive and compositional, thus supporting a stepwise and modular specification process. The approach is exemplified with access control policies.

Key words: Policy specification, policy refinement, policy adherence, UML sequence diagrams, access control

KEYWORDS	ENGLISH	NORWEGIAN
GROUP 1	Policy	Policy
GROUP 2	Modeling	Modellering
SELECTED BY AUTHOR	UML	UML
	Refinement	Raffinering

Compositional Refinement of Policies in UML – Exemplified for Access Control

Bjørnar Solhaug[1,2] and Ketil Stølen[2,3]

[1]Dep. of Information Science and Media Studies, University of Bergen
[2]SINTEF ICT [3]Dep. of Informatics, University of Oslo

{bjornar.solhaug,ketil.stolen}@sintef.no

Abstract

The UML is the *de facto* standard for system specification, but offers little specialized support for the specification and analysis of policies. This paper presents Deontic STAIRS, an extension of the UML sequence diagram notation with customized constructs for policy specification. The notation is underpinned by a denotational trace semantics. We formally define what it means that a system satisfies a policy specification, and introduce a notion of policy refinement. We prove that the refinement relation is transitive and compositional, thus supporting a stepwise and modular specification process. The approach is exemplified with access control policies.

Key words: Policy specification, policy refinement, policy adherence, UML sequence diagrams, access control

1 Introduction

Policy based management of information systems has the last decade been subject to increased attention, and several frameworks, see e.g. [24], have been introduced for the purpose of policy specification, analysis and enforcement. At the same time the UML 2.1 [17] has emerged as the *de facto* standard for the modeling and specification of information systems. However, the UML offers little specialized support for the specification and analysis of policies.

Policy specifications are used in policy based management of systems. The domain of management may vary, but typical purposes are access control, security and trust management, and management of networks and services. Whatever the management domain, the purpose is to control behavioral aspects of a system. This is reflected in our definition of a policy,

adopted from [23], viz. that *a policy is a set of rules governing the choices in the behavior of a system.*

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" [24]. This means that the capabilities or potential behavior of the system generally span wider than what is prescribed by the policy, i.e. the system can potentially violate the policy. A policy can therefore be understood as a set of normative rules about a system, defining the ideal, desirable or acceptable behavior of the system. In our approach, each rule is classified as either a permission, an obligation or a prohibition. This classification is based on standard deontic logic [15], and several of the existing approaches to policy specification have language constructs of such a deontic type, e.g. [1, 4, 11, 23]. This categorization is furthermore implemented in the ISO/IEC standard for open distributed processing [10].

The contribution of this paper is firstly an extension of the UML sequence diagram notation suitable for specifying policies. In [25] we evaluated UML sequence diagrams as a notation for policy specification, and argued that although the notation to a large extent is sufficiently expressive, it is not suitable for policy specification. The reason for this lies heavily in the fact that there are no constructs for expressing deontic modalities. In this paper we propose a customized notation, referred to as Deontic STAIRS, which is underpinned by the denotational trace semantics of the STAIRS approach to system development with UML sequence diagrams [7, 21]. The notation is not tailored for a specific type of policy, thus allowing the specification of policies for access control, security management, trust management, etc. In this paper the approach is exemplified with access control policies, whereas the work presented in [18] demonstrates the suitability of the notation to express trust management policies.

Secondly, this paper contributes by introducing a notion of policy adherence that formally defines what it means that a system satisfies a policy specification.

As pointed out also elsewhere [3, 19], although recognized as an important research issue, policy refinement still remains poorly explored in the literature. This paper contributes thirdly by proposing a notion of policy refinement that supports an incremental policy specification process from the more abstract and high-level to the more concrete and low-level. We show that the refinement relation is transitive, which is an important property as it allows a stepwise development process. We also show that each of a set of composition operators is monotonic with respect to the refinement relation. In the literature this is often referred to as compositionality, and means that a policy specification can be refined by refining individual parts of the specification separately.

Through refinement more details are added, and the specification is typ-

ically tailored towards an intended system (possibly including an enforcement mechanism). The set of systems that adhere to the policy specification thereby decreases. We show that the refinement relation ensures that if a system adheres to a concrete, refined policy specification, it also adheres to the more abstract specifications. Enforcement of the final specification thus implies the enforcement of the specifications from the earlier phases.

For specific domains a special purpose policy language, e.g. XACML [16] 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.

The next section introduces UML sequence diagrams and the STAIRS denotational semantics. In Section 3 we propose the customized syntax and semantics for policy specification with sequence diagrams. Section 4 formalizes the notion of policy adherence, whereas policy refinement is defined and analyzed in Section 5. Related work is discussed in Section 6 before we conclude in Section 7. For sake of readability of the main sections of the paper, a set of formal definitions are presented separately in Appendix A, whereas the full proofs of results are presented in Appendix B.

This paper is the full technical report on the results published in [26].

2 UML Sequence Diagrams and STAIRS

In this section we introduce the UML 2.1 sequence diagram notation and give a brief introduction to the denotational semantics as defined in the STAIRS approach. STAIRS formalizes, and thus precisely defines, the trace semantics that is only informally described in the UML 2.1 standard.

UML interactions describe system behavior by showing how entities interact by the exchange of messages. The behavior is described by traces which are sequences of event occurrences ordered by time. Several UML diagrams can specify interactions, and in this paper we focus on sequence diagrams where each entity is represented with a lifeline. To illustrate language constructs and central notions, we use a running example throughout the paper in which the interaction between a user U and an application A is defined. The diagram M to the left in Fig. 1 is very basic and has only two events, the sending of the message *login(id)* on U (which we denote $!l$) and the reception of the same message on A (denoted $?l$). The send event must occur before the receive event. The semantics of the diagram M is given by the single trace of these two events, denoted $\langle !l, ?l \rangle$.

The diagram W to the right in Fig. 1 shows the sending of the two messages l and r from U to A, where r denotes *read(doc)*. The order of the events on each lifeline is given by their vertical positions, but the two lifelines are independent. The semantics for each of the messages is as for

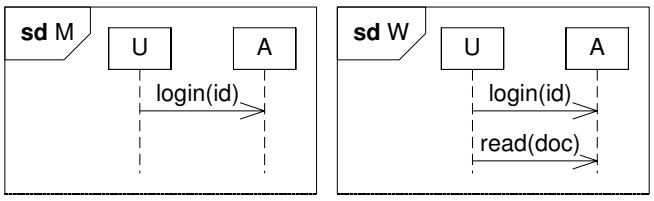

Figure 1: Sequence diagrams

the message in diagram M, and the semantics of W is given by weak sequencing of the two messages. Weak sequencing takes into account the independence of lifelines, so the semantics for the diagram W is given by the set $\{\langle !l, ?l, !r, ?r\rangle, \langle !l, !r, ?l, ?r\rangle\}$. The two traces represent the valid interpretations of the diagram; the sending of l is the first event to occur, but after that both the reception of l and the sending of r may occur.

The UML sequence diagram notation has further constructs for combining diagrams, most notably alt for specifying alternatives, par for parallel composition, and loop for several sequential compositions of one diagram with itself.

The traces of events defined by a diagram are understood as representing system runs. In each trace a send event is ordered before the corresponding receive event, and $\mathcal{H}$ denotes the trace universe, i.e. the set of all traces that complies with this requirement. A message is in the STAIRS denotational semantics given by a triple (s, tr, re) of a signal s, a transmitter tr and a receiver re. The transmitter and receiver are lifelines. $\mathcal{L}$ denotes the set of all lifelines and $\mathcal{M}$ denotes the set of all messages. An event is a pair of kind and message, $(k, m) \in \{!, ?\} \times \mathcal{M}$. By $\mathcal{E}$ we denote the set of all events, and we define the functions $k\_ \in \mathcal{E} \to \{!, ?\}$, $tr\_ , re\_ \in \mathcal{E} \to \mathcal{L}$ to yield the kind, transmitter and receiver of an event, respectively.

The functions $\frown$, $\circledS$ and $\circledT$ are for concatenation of sequences, filtering of sequences and filtering of pairs of sequences, respectively. Concatenation is to glue sequences together, so $h_1 \frown h_2$ is the sequence that equals h_1 if h_1 is infinite. Otherwise it denotes the sequence that has h_1 as prefix and h_2 as suffix, where the length equals the sum of the length of h_1 and h_2.

By $E \circledS a$ we denote the sequence obtained from the sequence a by removing all elements from a that are not in the set of elements E. For example, $\{1, 3\} \circledS \langle 1, 1, 2, 1, 3, 2\rangle = \langle 1, 1, 1, 3\rangle$.

The filtering function $\circledT$ is described as follows. For any set of pairs of elements F and pair of sequences t, by $F \circledT t$ we denote the pair of sequences obtained from t by truncating the longest sequence in t at the length of the shortest sequence in t if the two sequences are of unequal length; for each $j \in \{1, \ldots, k\}$, where k is the length of the shortest sequence in t, selecting or deleting the two elements at index j in the two sequences, depending on whether the pair of these elements is in the set F. For example, we have

that $\{(1,f),(1,g)\} \circledT (\langle 1,1,2,1,2\rangle, \langle f,f,f,g,g\rangle) = (\langle 1,1,1\rangle, \langle f,f,g\rangle)$.

Parallel composition ($\|$) of trace sets corresponds to the pointwise interleaving of their individual traces. The ordering of the events within each trace is maintained in the result. Weak sequencing ($\gtrsim$) is implicitly present in sequence diagrams and defines the partial ordering of the events in the diagram. For trace sets H_1 and H_2, the formal definitions are as follows.

Definition 1. Parallel composition.

$$H_1 \parallel H_2 \stackrel{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists s \in \{1,2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \circledT (s,h)) \in H_1 \wedge \pi_2((\{2\} \times \mathcal{E}) \circledT (s,h)) \in H_2\}$$

Definition 2. Sequential composition.

$$H_1 \gtrsim H_2 \stackrel{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in H_1, h_2 \in H_2 : \forall l \in \mathcal{L} : e.l \circledS h = e.l \circledS h_1 \frown e.l \circledS h_2\}$$

$\{1,2\}^\infty$ is the set of all infinite sequences over the set $\{1,2\}$, and π_2 is a projection operator returning the second element of a pair. The infinite sequence s in the definition can be understood as an oracle that determines which of the events in h that are filtered away. The expression $e.l$ denotes the set of events that may take place on the lifeline l. Formally

$$e.l \stackrel{\mathsf{def}}{=} \{e \in \mathcal{E} \mid (k.e = ! \wedge tr.e = l) \vee (k.e = ? \wedge re.e = l)\}$$

The semantics of a sequence diagram is defined by the function $[\![\ ]\!]$ that for a sequence diagram d yields a set of traces $[\![d]\!] \subseteq \mathcal{H}$ representing the behavior described by the diagram.

Definition 3. Semantics of sequence diagrams.

$$\begin{aligned}
[\![e]\!] &\stackrel{\mathsf{def}}{=} \{\langle e\rangle\} \text{ for any } e \in \mathcal{E} \\
[\![d_1 \ \mathsf{par}\ d_2]\!] &\stackrel{\mathsf{def}}{=} [\![d_1]\!] \parallel [\![d_2]\!] \\
[\![d_1 \ \mathsf{seq}\ d_2]\!] &\stackrel{\mathsf{def}}{=} [\![d_1]\!] \gtrsim [\![d_2]\!] \\
[\![d_1 \ \mathsf{alt}\ d_2]\!] &\stackrel{\mathsf{def}}{=} [\![d_1]\!] \cup [\![d_2]\!]
\end{aligned}$$

For the formal definition of further constructs and the motivation behind the definitions, see [7, 21].

3 Specifying Policies

In this section we present Deontic STAIRS, a customized notation for specifying policies with sequence diagrams. The notation is defined as a conservative extension of UML 2.1 sequence diagrams in the sense that the UML sequence diagram constructs of Deontic STAIRS are used in accordance with the standard [17]. We furthermore define a denotational trace semantics.

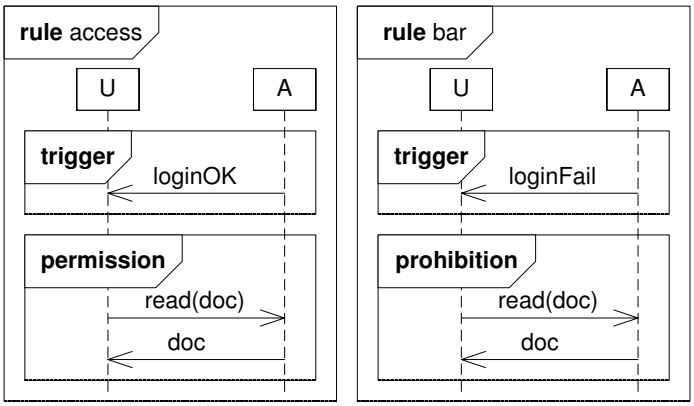

Figure 2: Policy rules

The notation constructs are illustrated by the examples of policy rules depicted in Fig. 2. We consider a policy that administrates the access of users U to an application A.

A policy rule is defined as a sequence diagram that consists of two parts, a trigger and a deontic expression. The trigger is a scenario that specifies the condition under which the given rule applies and is captured with the keyword trigger. The body of the deontic expression describes the behavior that is constrained by the rule, and the keywords permission, obligation and prohibition indicate the modality of the rule. The name of the rule consists of two parts, where the former part is the keyword rule, and the latter part is any chosen name for the rule.

The rule *access* to the left in Fig. 2 is a permission stating that by the sending of the message *loginOK* from the application to the user, i.e. the id of the user has been verified, the user is permitted to retrieve documents from the system. In case of login failure, the rule *bar* to the right in Fig. 2 specifies that document retrieval is prohibited, i.e. the user is barred from accessing the application.

Generally, a diagram specifying a policy rule contains one or more lifelines, each representing a participating entity. There can be any number of entities, but there must be at least one. In the examples we have for simplicity shown only two lifelines, U and A. We also allow the trigger to be omitted. In that case the rule applies under all circumstances and is referred to as a standing rule.

By definition of a policy, a policy specification is given as a set of rules, each specified in the form shown in Fig. 2.

The extension of the sequence diagram notation presented in this section is conservative with respect to the UML standard, so people that are familiar with UML should be able to understand and use the notation. All the constructs that are available in the UML for specification of sequence

diagrams can furthermore freely be used in the specification of the body of a policy rule.

Semantically, the triggering scenario and the body of a rule are given by trace sets $T \subseteq \mathcal{H}$ and $B \subseteq \mathcal{H}$, respectively. Additionally, the semantics must capture the deontic modality, which we denote by $dm \in \{pe, ob, pr\}$. The semantics of a policy rule is then given by the tuple $r = (dm, T, B)$. Notice that for standing rules, the trigger is represented by the set of all traces, i.e. $T = \mathcal{H}$. Since a policy is a set of policy rules, the semantics of a policy specification is given by a set $P = \{r_1, \ldots, r_m\}$, where each r_i is the semantic representation of a policy rule.

4 Policy Adherence

In this section we define the adherence relation $\rightarrow_a$ that for a given policy specification P and a given system S defines what it means that S satisfies P, denoted $P \rightarrow_a S$. We assume a system model in which the system is represented by a (possibly infinite) set of traces S, where each trace describes a possible system execution. In order to define $P \rightarrow_a S$, we first define what it means that a system adheres to a rule $r \in P$, denoted $r \rightarrow_a S$.

A policy rule applies if and when a prefix h' of an execution $h \in S$ triggers the rule, i.e. the prefix $h' \sqsubseteq h$ fulfills the triggering scenario T. The function $\sqsubseteq$ is a predicate that takes two traces as operand and yields true iff the former is equal to or a prefix of the latter. Since the trace set T represents the various executions under which the rule applies, it suffices that at least one trace $t \in T$ is fulfilled by h' for the rule to trigger. Furthermore, for h' to fulfill t, the trace t must be a sub-trace of h', denoted $t \lhd h'$.

For traces $h_1, h_2 \in \mathcal{H}$, if $h_1 \lhd h_2$ we say that h_1 is a sub-trace of h_2 and, equivalently, that h_2 is a super-trace of h_1. Formally, the sub-trace relation is defined as follows.

Definition 4. $h_1 \lhd h_2 \stackrel{\mathsf{def}}{=} \exists s \in \{1,2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \circledcirc (s, h_2)) = h_1$

The expression $h_1 \lhd h_2$ evaluates to true iff there exists a filtering such that when applied to h_2 the resulting trace equals h_1. For example, $\langle a, b, c\rangle \lhd \langle e, a, b, e, f, c\rangle$. For a trace set H and traces h and h' we define the following.

Definition 5.

$$
\begin{aligned}
H \lhd h &\stackrel{\mathsf{def}}{=} \exists h' \in H : h' \lhd h \\
h' \not\lhd h &\stackrel{\mathsf{def}}{=} \neg(h' \lhd h) \\
H \not\lhd h &\stackrel{\mathsf{def}}{=} \neg\exists h' \in H : h' \lhd h
\end{aligned}
$$

Since a sequence diagram is represented by a set of traces H, it suffices that a trace h is a super-trace of at least one element of H for h to fulfill the

sequence diagram. This is captured by the expression $H \lhd h$. Formally, the triggering of a rule (dm, T, B) by a trace $h \in S$ is then defined as follows.

Definition 6. The rule (dm, T, B) is triggered by the trace h iff $T \lhd h$.

To check whether a system S adheres to a rule (dm, T, B) we first need to identify all the triggering prefixes of traces of S. Then, for each triggering prefix, we need to check the possible continuations. As an example, consider the system $S = \{h_1, h_2, h_3\}$. Assume that h_1 and h_2 have a common prefix h_a that triggers the rule, i.e. h_1 and h_2 can be represented by the concatenations $h_a \frown h_b$ and $h_a \frown h_c$, respectively, such that $T \lhd h_a$. Assume, furthermore, that the system trace h_3 does not trigger the rule, i.e. $T \not\lhd h_3$.

The three runs can be structured into a tree as depicted in Fig. 3. Adherence to a policy rule intuitively means the following. The system adheres to the permission (pe, T, B) if at least one of the traces h_b and h_c fulfills B; so a permission requires the existence of a continuation that fulfills the behavior. The system adheres to the obligation (ob, T, B) if both of h_b and h_c fulfill B; so an obligation requires that all possible continuations fulfill the behavior. The system adheres to the prohibition (pr, T, B) if neither h_b nor h_c fulfill B; so a prohibition requires that none of the possible continuations fulfill the behavior. Notice that to fulfill the behavior given by the trace set B, it suffices to fulfill one of the traces since each element of B represents a valid way of executing the behavior described by the rule body. As for the trace h_3, since the rule is not triggered, the rule is trivially satisfied.

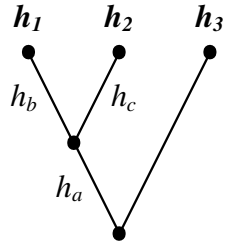

Figure 3: Structured traces

Adherence to policy rule r of system S, denoted $r \rightarrow_a S$ is defined as follows, where $h|_k$ is a truncation operation that yields the prefix of h of length $k \in \mathbb{N}$.

Definition 7. Adherence to policy rule of system S:

– $(pe, T, B) \rightarrow_a S \stackrel{\text{def}}{=} \forall h \in S : \forall t \in T : t \lhd h \Rightarrow$
$\exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge (\{t\} \gtrsim B) \lhd h'$

– $(ob, T, B) \rightarrow_a S \stackrel{\text{def}}{=} \forall h \in S : \forall t \in T : t \lhd h \Rightarrow (\{t\} \gtrsim B) \lhd h$

– $(pr, T, B) \rightarrow_a S \stackrel{\text{def}}{=} \forall h \in S : \forall t \in T : t \lhd h \Rightarrow (\{t\} \gtrsim B) \not\lhd h$

With these definitions of adherence to policy rule of a system S, we define adherence to a policy specification P as follows.

Definition 8. $P \rightarrow_a S \stackrel{\mathsf{def}}{=} \forall r \in P : r \rightarrow_a S$

Example 1. As an example of policy rule adherence, consider the permission rule *access* to the left in Fig. 2 stating that users U are allowed to retrieve documents from the application A after a valid login. Semantically, we have $access = (pe, T, B)$, where T is the singleton set

$$\{\langle(!, (loginOK, A, U)), (?, (loginOK, A, U))\rangle\}$$

and B is the singleton set containing the sequence of events depicted to the left in Fig. 4.

Trace of rule *access*	Partial trace of S
$(!, (read(doc), U, A))$	$\cdots$
$(?, (read(doc), U, A))$	$(!, (login(id), U, A))$
$(!, (doc, A, U))$	$(?, (login(id), U, A))$
$(?, (doc, A, U))$	$(!, (query(id), A, SA))$
	$(?, (query(id), A, SA))$
	$(!, (valid(id), SA, A))$
	$(?, (valid(id), SA, A))$
	$(!, (loginOK, A, U))$
	$(?, (loginOK, A, U))$
	$(!, (read(doc), U, A))$
	$(?, (read(doc), U, A))$
	$(!, (doc, A, U))$
	$(?, (doc, A, U))$
	$(!, (store(doc'), U, A))$
	$(?, (store(doc'), U, A))$
	$\cdots$

Figure 4: Traces of rule and system

To the right in Fig. 4 we have shown a partial trace of S in the case that $access \rightarrow_a S$. The user U sends a login message to the application A, after which the application sends a query to the security administrator SA to verify the id of the user. At some point in the execution the events $(!, (loginOK, A, U))$ and $(?, (loginOK, A, U))$ triggering the rule occur. The user then retrieves a document and finally stores a modified version. Since there exists a filtering of the system trace that equals the trace representing the body of the permission rule, the system adheres to the rule. Other system traces with the same triggering prefix need not fulfill the trace of the rule since the rule is a permission.

The definition of policy adherence is based on the satisfiability relation of deontic logic which defines what it means that a model satisfies a deontic expression. Standard deontic logic is a modal logic that is distinguished by the axiom $\mathbf{OB}p \supset \mathbf{PE}p$, stating that all that is obligated is also permitted. The next theorem states that this property as well as the definitions $\mathbf{OB}p \equiv \neg\mathbf{PE}\neg p$ (p is obligated iff the negation of p is not permitted) and $\mathbf{OB}p \equiv \mathbf{PR}\neg p$ (p is obligated iff the negation of p is prohibited) of deontic logic are preserved by our definition of adherence.

Theorem 1.

- $(ob, T, B) \rightarrow_a S \Rightarrow (pe, T, B) \rightarrow_a S$
- $(ob, T, B) \rightarrow_a S \Leftrightarrow (\neg pe, T, \neg B) \rightarrow_a S$
- $(ob, T, B) \rightarrow_a S \Leftrightarrow (pr, T, \neg B) \rightarrow_a S$

Notice that the use of negation in the theorem is pseudo-notation. The precise definitions are as follows.

Definition 9.

- $(\neg pe, T, \neg B) \rightarrow_a S \stackrel{\mathsf{def}}{=} \forall h \in S : \forall t \in T : t \lhd h \Rightarrow$
  $\neg\exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge \neg((\{t\} \gtrsim B) \lhd h')$
- $(pr, T, \neg B) \rightarrow_a S \stackrel{\mathsf{def}}{=} \forall h \in S : \forall t \in T : t \lhd h \Rightarrow \neg((\{t\} \gtrsim B) \not\lhd h)$

The first clause of Theorem 1 follows immediately from the definition of adherence, whereas the second and third clause are shown by manipulation of quantifiers, negations and set inclusions.

Generally, the inter-definability axioms of deontic logic linking obligations to permissions are not adequate for policy based management of distributed systems since permissions may be specified independently of obligations and by different administrators. An obligation rule of a network configuration policy, for example, does not imply the authorization to conduct the given behavior if authorizations are specified in the form of permission rules of a security policy.

However, an obligation for which there is no corresponding permission represents a policy conflict which must be resolved for the policy to be enforceable. A policy specification P is consistent, or conflict free, iff there exists a system S such that $P \rightarrow_a S$. Theorem 1 reflects properties of consistent policy specifications, and if any of these properties are not satisfied there are occurrences of modality conflicts, and the policy cannot be enforced.

There are five types of modality conflicts. First, obligation to conduct the behavior represented by the set of traces B, while the complement $\overline{B}$ (defined by $\mathcal{H} \setminus B$) is also obligated; second, prohibiting B while prohibiting the complement $\overline{B}$; third, prohibiting B while obligating B; four, permitting B while obligating $\overline{B}$; five, prohibiting B while also permitting B.

In policies for distributed systems conflicts are likely to occur since different rules may be specified by different managers, and since multiple policy rules may apply to the same system entities. The problem of detecting and resolving policy conflicts is outside the scope of this paper, but existing solutions to resolving modality conflicts, see e.g. [14], can be applied.

5 Policy Refinement

We aim for a notion of refinement that allows policy specifications to be developed in a stepwise and modular way. Stepwise refinement is ensured by transitivity, which means that a policy specification that is the result of a number of refinement steps is a valid refinement of the initial, most abstract specification. Modularity means that a policy specification can be refined by refining individual parts of the specification separately.

Refinement of a policy rule means to weaken the trigger or strengthen the body. A policy specification may also be refined by adding new rules to the specification. Weakening the trigger means to increase the set of traces that trigger the rule. For permissions and obligations, the body is strengthened by reducing the set of traces representing the behavior, whereas the body of a prohibition is strengthened by increasing the set of prohibited traces. The refinement relation $\rightsquigarrow_{tr}$ for the triggering scenario, and the refinement relations $\rightsquigarrow_{pe}$, $\rightsquigarrow_{ob}$ and $\rightsquigarrow_{pr}$ for the body of permissions, obligations and prohibitions, respectively, are defined as follows.

Definition 10. Refinement of policy trigger and body:

- $T \rightsquigarrow_{tr} T' \stackrel{\mathsf{def}}{=} T' \supseteq T$
- $B \rightsquigarrow_{pe} B' \stackrel{\mathsf{def}}{=} B' \subseteq B$
- $B \rightsquigarrow_{ob} B' \stackrel{\mathsf{def}}{=} B' \subseteq B$
- $B \rightsquigarrow_{pr} B' \stackrel{\mathsf{def}}{=} B' \supseteq B$

Obviously, these relations are transitive and reflexive. The relations are furthermore compositional, which means that the different parts of a sequence diagram d can be refined separately. Compositionality is ensured by monotonicity of the composition operators with respect to refinement as expressed in the following theorem. The instances of the relation $\rightsquigarrow$ denote any of the above four refinement relations.

Theorem 2. If $d_1 \rightsquigarrow d_1'$ and $d_2 \rightsquigarrow d_2'$, then the following hold.

- d_1 seq $d_2 \rightsquigarrow d_1'$ seq d_2'
- d_1 alt $d_2 \rightsquigarrow d_1'$ alt d_2'
- d_1 par $d_2 \rightsquigarrow d_1'$ par d_2'

The theorem follows directly from the definition of the composition operators. Since the refinement relations are defined by the subset and the superset relations, the theorem is proven by showing that the operators $\succsim$, $\cup$ and $\|$ on trace sets (defining sequential, alternative and parallel composition, respectively) are monotonic with respect to $\subseteq$ and $\supseteq$. For seq and $\subseteq$, the result

$$[\![d_1']\!] \subseteq [\![d_1]\!] \wedge [\![d_2']\!] \subseteq [\![d_2]\!] \Rightarrow [\![d_1']\!] \succsim [\![d_2']\!] \subseteq [\![d_1]\!] \succsim [\![d_2]\!]$$

holds since the removal of elements from $[\![d_1]\!]$ or $[\![d_2]\!]$ yields a reduction of set of traces that results from applying the $\succsim$ operator. The case of monotonicity of $\succsim$ with respect to $\supseteq$ is symmetric. The argument for par, i.e. monotonicity of $\|$, is similar to seq, whereas the case of the union operator $\cup$ defining alt is trivial.

We now define refinement of a policy rule as follows.

Definition 11.

$$(dm, T, B) \rightsquigarrow (dm', T', B') \stackrel{\mathsf{def}}{=} dm = dm' \wedge T \rightsquigarrow_{tr} T' \wedge B \rightsquigarrow_{dm} B'$$

It follows immediately from reflexivity and transitivity of the refinement relations $\rightsquigarrow_{tr}$ and $\rightsquigarrow_{dm}$ that the refinement relation $\rightsquigarrow$ for policy rules is also reflexive and transitive.

A policy is a set of rules, and for a policy specification P' to be a refinement of a policy specification P, we require that each rule in P must be refined by a rule in P'.

Definition 12. $P \rightsquigarrow P' \stackrel{\mathsf{def}}{=} \forall r \in P : \exists r' \in P' : r \rightsquigarrow r'$

Theorem 2 addresses composition of interactions within a policy rule r. At the level of policy specifications, composition is simply the union of rule sets P. It follows straightforwardly that policy composition is monotonic with respect to refinement, i.e. $P_1 \rightsquigarrow P_1' \wedge P_2 \rightsquigarrow P_2' \Rightarrow P_1 \cup P_2 \rightsquigarrow P_1' \cup P_2'$. Refinement of policy specifications is furthermore transitive, i.e. $P_1 \rightsquigarrow P_2 \wedge P_2 \rightsquigarrow P_3 \Rightarrow P_1 \rightsquigarrow P_3$.

Development of policy specifications through refinement allows an abstract and general view of the system in the initial phases, ignoring details of system behavior, design and architecture. Since the specification is strengthened through refinement and more detailed aspects of the system are considered, the set of systems that adhere to the policy specification decreases. However, a system that adheres to a concrete, refined specification also adheres to the initial, abstract specification. This means that if a policy specification is further refined before it is enforced, the enforcement ensures that the initial, abstract specification is also enforced. This is expressed in the next theorem.

Theorem 3. Given a system S and policy specifications P and P', if $P \rightsquigarrow P'$ and $P' \rightarrow_a S$, then $P \rightarrow_a S$.

Policy composition and refinement do not rely on the assumption that the rules are mutually consistent or conflict free, which means that inconsistencies may be introduced during the development process. However, potential conflicts are generally inherent in policies for distributed systems [14]. Development of policy specification with refinement is in this respect desirable since conflicts and other errors are generally easier to detect and correct at abstract levels.

Example 2. In the following we give an example of policy specification refinement. Let, first, $P_1 = \{access, bar\}$ be the policy specification given by the permission and the prohibition depicted in Fig. 2. Refinement allows adding rules to the specification, so assume the obligation rule *loginFail* in Fig. 5 and the obligation rule *disable* in Fig. 6 are added to the rule set such that $P_2 = \{access, bar, loginFail, disable\}$.

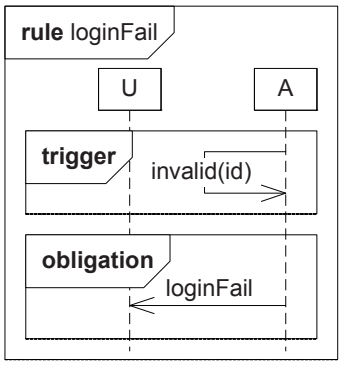

Figure 5: Login failure

The former rule states that the application is obligated to alert the user in case of a login failure, i.e. when the user id is invalid. The latter rule, adapted from [4], states that in case of three consecutive login failures, the application is obligated to disable the user, log the incident and alert the user.

The body of the rule to the left in Fig. 6 is specified with the UML 2.1 sequence diagram construct called interaction use which is a reference to another diagram. The interaction use covers the lifelines that are included in the referenced diagram. The body is defined by the parallel composition of the three diagrams d (disable the user), l (log the incident) and a (alert the user) to the right in Fig. 6. Equivalently, the referenced diagrams can be specified directly in place of the respective interaction uses.

By reflexivity, the permission and prohibition of P_2 are refinements of the same rules in P_1. Since adding rules is valid in refinement, P_2 is a refinement of P_1. Obviously, a system that adheres to P_2 also adheres to P_1.

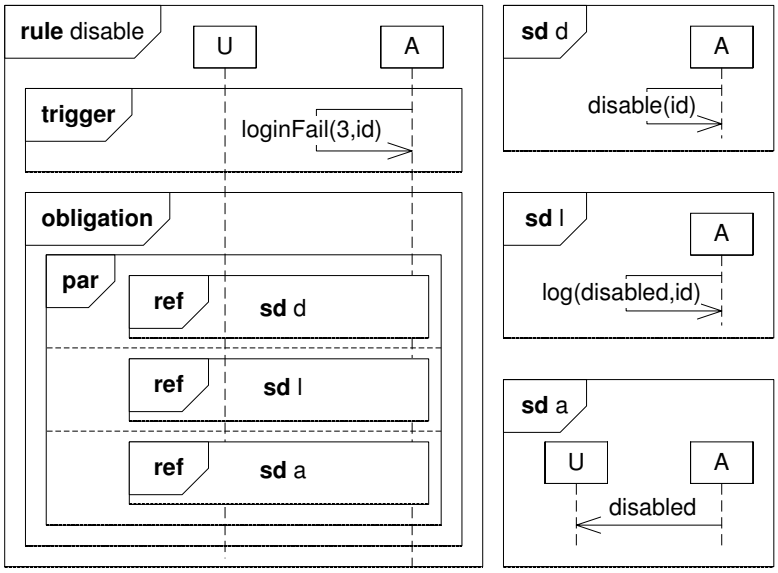

Figure 6: Disable user

The rules in both P_1 and P_2 refer to interactions only between the application and the users, which may be suitable at the initial development phases. At later stages, however, the policy specification is typically specialized towards a specific system, and more details about the system architecture is taken into account. This is supported through refinement by decomposition of a single entity into several entities, thus allowing behavior to be specified in more detail. Due to space limits refinement by detailing is only exemplified in this paper. See [7] for a formal definition.

The rule *loginFail2* in Fig. 7 shows a refinement of the rule *loginFail* in Fig. 5. Here, the application A has been decomposed into the entities security administrator SA and log L. The refined obligation rule states that by the event of login failure, the security administrator must log the incident before alerting the user. The log also reports to the security administrator the current number n of consecutive login failures. Observe that the modality as well as the trigger are the same in both *loginFail* and *loginFail2*, and that the interactions between the application and the user are identical. This implies that *loginFail2* is a detailing of *loginFail*. Hence, *loginFail* $\rightsquigarrow$ *loginFail2*. It is easily seen that adherence to the latter rule implies adherence to the former.

Compositionality of refinement means that for a given policy specification, the individual rules can be refined separately. This means that for the policy specification $P_3 = \{access, bar, loginFail2, disable\}$ we have $P_2 \rightsquigarrow P_3$ and that for all systems S, $P_3 \rightarrow_a S$ implies $P_2 \rightarrow_a S$. By transitivity of refinement we also have that $P_1 \rightsquigarrow P_3$ and that adherence to P_3 implies adherence to P_1.

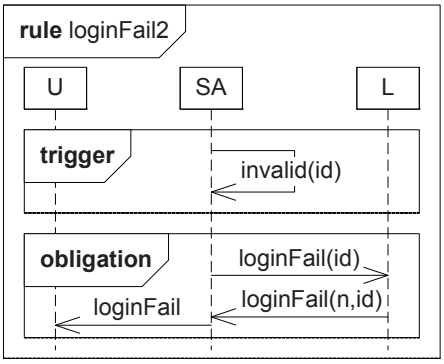

Figure 7: Login failure refined

Compositionality of refinement also means that in order to refine a policy rule, the individual parts of the body of a rule can be refined separately. We illustrate this by showing a refinement of the body of the rule *disable* of Fig. 6. The body shows the parallel composition of three diagrams, denoted d par l par a.

Fig. 8 shows refinement of the diagram elements d and l into $d2$ and $l2$, respectively. In $d2$ the lifeline A has been decomposed into the components security administrator *SA* and user store *US* and shows the security administrator disabling a user by sending a message to the user store. We now have that $d \rightsquigarrow d2$ and, similarly, that $l \rightsquigarrow l2$ for the other diagram element. By compositionality of refinement of rule body, we get that $(d \text{ par } l \text{ par } a) \rightsquigarrow (d2 \text{ par } l2 \text{ par } a)$.

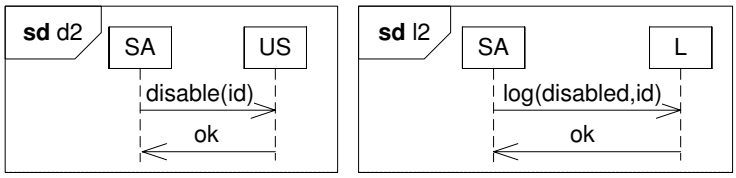

Figure 8: Refined diagrams

Let the obligation rule *disable2* be defined by replacing the references to d and l in *disable* of Fig. 6 with references to $d2$ and $l2$, respectively, of Fig. 8. We now have that *disable* $\rightsquigarrow$ *disable2*. The policy specification $P_4 = \{access, bar, loginFail2, disable2\}$ is a refinement of P_3 and, by transitivity, a refinement of P_2 and P_1 also. As before, $P_4 \rightarrow_a S$ implies $P_1 \rightarrow_a S$ for all systems S.

These examples show how more detailed aspects of system architecture and behavior may be taken into account at more refined levels. Another feature of refinement is that the behavior defined at abstract levels can be constrained at more concrete levels by ruling out alternatives. As an exam-

ple, consider the body of the rule *disable2* which semantically is captured by the set trace set $[\![d2 \text{ par } l2 \text{ par } a]\!]$. This defines an interleaving of the traces of the three elements; there are no constraints on the ordering between them. The ordering can, however, be constrained by using sequential composition instead of parallel composition. If, for example, it is decided that the disabling of the user and the logging of the incident should be conducted before the user is alerted, this is defined by $(d2 \text{ par } l2) \text{ seq } a$. Sequential composition is a special case of parallel composition, so semantically we now have that $[\![(d2 \text{ par } l2) \text{ seq } a]\!] \subseteq [\![d2 \text{ par } l2 \text{ par } a]\!]$. For the obligation rule, the former set of traces represents a refinement of the latter set of traces.

Let *disable3* be defined as *disable2* where $d2 \text{ par } l2 \text{ par } a$ of the latter is replaced with $(d2 \text{ par } l2) \text{ seq } a$ in the former. Then *disable2* $\rightsquigarrow$ *disable3*. By defining the specification $P_5 = \{access, bar, loginFail2, disable3\}$ we have $P_4 \rightsquigarrow P_5$. By transitivity, P_5 is a refinement of all the previous policy specifications of this example, and adherence to P_5 implies adherence to them all.

6 Related Work

Although a variety of languages and frameworks for policy based management has been proposed the last decade or so, policy refinement is still in its initial phase and little work has been done on this issue. After being introduced in [2] the goal-based approach to policy refinement has emerged as a possible approach and has also later been further elaborated [3, 19, 20].

In the approach described in [2], system requirements that eventually are fulfilled by low-level policy enforcement are captured through goal refinement. Initially, the requirements are defined by high-level, abstract policies, and so called strategies that describe the mechanisms by which the system can achieve a set of goals are formally derived from a system description and a description of the goals. Formal representation and reasoning are supported by the formalization of all specifications in event calculus.

Policy refinement is supported by the refinement of goals, system entities and strategies, allowing low-level, enforceable policies to be derived from high-level, abstract ones. Once the eventual strategies are identified, these are specified as policies the enforcement of which ensures the fulfillment of the abstract goals. As opposed to our approach, there is no refinement of policy *specifications*. Instead, the final polices are specified with Ponder [4], which does not support the specification of abstract policies that can be subject to refinement. The goal-based approach to policy refinement hence focus on refinement of policy requirements rather than policy specifications.

The same observations hold for the goal-based approaches described in [3, 19, 20], where the difference between [3, 2] and [19, 20] mainly is on the strategies for how to derive the policies to ensure the achievement of

a given goal. The former use event calculus and abduction in order to derive the appropriate strategies, whereas the latter uses automated state exploration for obtaining the appropriate system executions. All approaches are, however, based on requirements capturing through goal refinement, and Ponder is used as the notation for the eventual policy specification.

In [3] a policy analysis and refinement tool supporting the proposed formal approach is described. In [2], the authors furthermore show that the formal specifications and results can be presented with UML diagrams to facilitate usability. The UML is, however, used to specify goals, strategies, etc., and not the policies *per se* as in our approach. In our evaluation of the UML as a notation for specifying policies [25] we found that sequence diagrams to a large extent have the required expressiveness, but that the lack of a customized syntax and semantics makes them unsuitable for this purpose. The same observation is made in attempts to formalize policy concepts from the reference model for open distributed processes [10] using the UML [1, 13]. Nevertheless, in this paper we have shown that with minor extensions, policy specification and refinement can be supported.

UML sequence diagrams extend message sequence charts (MSCs) [9], and both MSCs and a family of approaches that have emerged from them, e.g. [5, 6, 12, 22], could be considered as alternatives to notations for policy specification. These approaches, however, lack the expressiveness to specify policies and capture a notion of refinement with the properties demonstrated in this paper.

Live sequence charts (LSCs)[6] and modal sequence diagrams (MSDs) [5] are two similar approaches based on a distinction between existential and universal diagrams. This distinction can be utilized to specify permissions, obligations and prohibitions. However, conditionality is not supported for existential diagrams in LSCs which means that diagrams corresponding to our permissions cannot be specified with triggers. A precise or formal notion of refinement is also not defined for these approaches. In [12], a variant of MSCs is provided a formal semantics and is supported by a formal notion of refinement. MSCs are interpreted as existential, universal or negative (illegal) scenarios, which is related to the specification of permissions, obligations and prohibitions, respectively, in Deontic STAIRS. There are, however, no explicit constructs in the syntax for distinguishing between these interpretations. Conditional scenarios with a triggering construct are supported in [12], but as for LSCs the composition of the triggering scenario and the triggered scenario is that of strong sequencing. This can be unfortunate in the specification of distributed systems in which entities behave locally and interact with other entities asynchronously.

Triggered message sequence charts (TMSCs) [22] allow the specification of conditional scenarios and is supported by compositional refinement. There is, however, no support for distinguishing between permitted, obligated and prohibited scenarios; a system specification defines a set of valid

traces, and all other traces are invalid.

7 Conclusion and Future Work

In this paper we have shown that the deontic notions of standard deontic logic [15] can be expressed in the UML by a conservative extension of the sequence diagram notation, thus enabling policy specification. We have defined both a formal notion of policy adherence and a formal notion of refinement. The refinement relation is transitive and also supports a compositional policy development, which means that individual parts of the policy specification can be developed separately. The refinement relation also ensures that the enforcement of a low-level policy specification implies the enforcement of the initial high-level specification.

Stepwise and compositional development of policy specifications is desirable as it facilitates the development process. Policy analysis is furthermore facilitated as analysis generally is easier and more efficient at abstract levels, and identified flaws are cheaper to fix. However, for policy analysis to be meaningful at an abstract level, the results must be preserved under refinement. In future work we will analyze the refinement relation with respect to such property preservation, particularly with respect to security, trust and adherence.

In the future we will also define language extensions to allow the specification of constraints in the form of Boolean expressions that limit the applicability of policy rules to specific system states. A refinement relation appropriate for this extension will also be defined.

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).

References

[1] J. Ø. Aagedal and Z. Milošević. ODP enterprise language: UML perspective. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 60–71. IEEE CS Press, 1999.

[2] A. K. Bandara, E. C. Lupu, J. Moffet, and A. Russo. A goal-based approach to policy refinement. In *Proceedings of the 5th International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, pages 229–239. IEEE Computer Society, 2004.

[3] A. K. Bandara, E. C. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou. Policy refinement for DiffServ qual-

ity of service management. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM'05)*, pages 469–482, 2005.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of the 2nd International Workshop on Policies for Distributed Systems and Networks (POLICY'01)*, volume 1995 of *LNCS*, pages 18–38. Springer, 2001.

[5] D. Harel and S. Maoz. Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software and Systems Modeling*, 7(2):237–252, 2007.

[6] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.

[7] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Software and Systems Modeling*, 4(4):355–367, 2005.

[8] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2006.

[9] International Telecommunication Union. *Recommendation Z.120 – Message Sequence Chart (MSC)*, 1999.

[10] ISO/IEC. *FCD 15414, Information Technology - Open Distributed Processing - Reference Model - Enterprise Viewpoint*, 2000.

[11] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY'03)*, pages 63–74. IEEE Computer Society, 2003.

[12] I. H. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Institut für Informatik, Ludwig-Maximilians-Universität München, July 2000.

[13] P. Linington. Options for expressing ODP enterprise communities and their policies by using UML. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 72–82. IEEE CS Press, 1999.

[14] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[15] P. McNamara. Deontic logic. In D. M. Gabbay and J. Woods, editors, *Logic and the Modalities in the Twentieth Century*, volume 7 of *Handbook of the History of Logic*, pages 197–288. Elsevier, 2006.

[16] OASIS. *eXstensible Access Control Markup Language (XACML) Version 2.1*, 2005.

[17] Object Management Group. *Unified Modeling Language: Superstructure, version 2.1.1*, 2007.

[18] A. Refsdal, B. Solhaug, and K. Stølen. 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)*, pages 33–49. Springer, 2008.

[19] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, and G. Pavlou. A functional solution for goal-oriented policy refinement. In *Proceedings of the 7th International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, pages 133–144. IEEE Computer Society, 2006.

[20] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G. Pavlou, and A. L. Lafuente. Using linear temporal model checking for goal-oriented policy refinement frameworks. In *Proceedings of the 6th International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 181–190. IEEE Computer Society, 2005.

[21] R. K. Runde, A. Refsdal, and K. Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice – Part 1: Underspecification and inherent nondeterminism. Technical Report 346, Department of Informatics, University of Oslo, 2007.

[22] B. Sengupta and R. Cleaveland. Triggered message sequence charts. *IEEE Transactions on Software Engineering*, 32(8):587–607, 2006.

[23] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[24] M. Sloman and E. Lupu. Security and management policy specification. *Network, IEEE*, 16(2):10–19, 2002.

[25] B. Solhaug, D. Elgesem, and K. Stølen. Specifying policies using UML sequence diagrams – An evaluation based on a case study. In *Proceedings of the 8th International Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, pages 19–28. IEEE Computer Society, 2007.

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

A Operations on Sequences

By E^∞ and E^ω, we denote the set of all infinite sequences and the set of all finite and infinite sequences over the set E of elements, respectively. $\mathbb{N}$ denotes the natural numbers, and $\mathbb{N}_0$ denotes $\mathbb{N} \cup \{0\}$. We use $\langle\rangle$ to denote the empty sequence, and by $\langle e_1, e_2, \ldots, e_n\rangle$ we denote the sequence of n elements, whose first element is e_1, second element is e_2, etc.

We define the functions

$$\#\_ \in E^\omega \to \mathbb{N}_0 \cup \{\infty\}, \quad \_[\_] \in E^\omega \times \mathbb{N} \to E$$

to yield respectively the length and the nth element of a sequence. We define the function

$$\_ \frown \_ \in E^\omega \times E^\omega \to E^\omega$$

for concatenation of sequences, i.e., gluing together sequences. Formally, concatenation is defined by the following.

$$(s_1 \frown s_2)[n] \stackrel{\mathsf{def}}{=} \begin{cases} s_1[n] & \text{if } 1 \le n \le \#s_1 \\ s_2[n - \#s_1] & \text{if } \#s_1 < n \le \#s_1 + \#s_2 \end{cases}$$

The prefix relation on sequences,

$$\_ \sqsubseteq \_ \in E^\omega \times E^\omega \to \mathbb{B}\mathrm{ool}$$

is formally defined as follows.

$$s_1 \sqsubseteq s_2 \stackrel{\mathsf{def}}{=} \exists s \in E^\omega : s_1 \frown s = s_2$$

The complementary relation is defined by the following.

$$s_1 \not\sqsubseteq s_2 \stackrel{\mathsf{def}}{=} \neg(s_1 \sqsubseteq s_2)$$

The truncation operator

$$\_|\_ \in E^\omega \times \mathbb{N} \cup \{\infty\} \to E^\omega$$

is used to truncate a sequence at a given length.

$$s|_j \stackrel{\mathsf{def}}{=} \begin{cases} s' & \text{if } 0 \le j \le \#s, \text{where } \#s' = j \wedge s' \sqsubseteq s \\ s & \text{if } j > \#s \end{cases}$$

$\mathbb{P}(E)$ denotes the set of all subsets of E. The filtering operator

$$\_ \circledS \_ \in \mathbb{P}(E) \times E^\omega \to E^\omega$$

is used to filter away elements. $A \circledS s$ denotes the sub-trace of s obtained by removing elements of s that are not in A. For a finite sequence s, this operator is completely defined by the following conditional equations.

$$\begin{array}{rcll} & A \circledS \langle\rangle & = & \langle\rangle \\ e \in A & \Rightarrow & A \circledS (\langle e \rangle \frown s) & = & \langle e \rangle \frown (A \circledS s) \\ e \notin A & \Rightarrow & A \circledS (\langle e \rangle \frown s) & = & A \circledS s \end{array}$$

For an infinite sequence s, we need one additional equation.

$$\forall n \in \mathbb{N} : s[n] \notin A \Rightarrow A \circledS s = \langle\rangle$$

The filtering operator $\circledT$ is defined for pairs of sequences:

$$\_ \circledT \_ \in \mathbb{P}(E \times E) \times (E^\omega \times E^\omega) \to (E^\omega \times E^\omega)$$

In order to formally define this operator, we first generalize some of the above operators on sequences to pairs of sequences.

$$\begin{array}{rcl} \#(s_1, s_2) & = & \mathsf{min}\{\#s_1, \#s_2\} \\ (s_1, s_2)[n] & = & (s_1[n], s_2[n]) \\ (s_1, s_2) \frown (s_1', s_2') & = & (s_1 \frown s_1', s_2 \frown s_2') \\ (s_1, s_2)|_j & = & (s_1|_j, s_2|_j) \end{array}$$

Furthermore, for elements $e_1, e_2 \in E$, $\langle (e_1, e_2) \rangle$ denotes $(\langle e_1 \rangle, \langle e_2 \rangle)$.

For a pair of sequences $c = (s_1, s_2)$, the filtering operator $\circledT$ is now defined by the following conditional equations.

$$\begin{array}{rcll} & & B \circledT c & = & B \circledT (c|_{\#c}) \\ & & B \circledT (\langle\rangle, \langle\rangle) & = & (\langle\rangle, \langle\rangle) \\ f \in B & \Rightarrow & B \circledT (\langle f \rangle \frown c) & = & \langle f \rangle \frown B \circledT c \\ f \notin B & \Rightarrow & B \circledT (\langle f \rangle \frown c) & = & B \circledT c \\ \forall n < \#c + 1 : c[n] \notin B & \Rightarrow & B \circledT c & = & (\langle\rangle, \langle\rangle) \end{array}$$

B Proofs

Theorem 1.

– $(ob, T, B) \rightarrow_a S \Rightarrow (pe, T, B) \rightarrow_a S$
– $(ob, T, B) \rightarrow_a S \Leftrightarrow (\neg pe, T, \neg B) \rightarrow_a S$
– $(ob, T, B) \rightarrow_a S \Leftrightarrow (pr, T, \neg B) \rightarrow_a S$

Proof. The third clause follows immediately from Definition 7 of adherence and the definition of the sub-trace relation. The first and second clauses are proved in Lemma 1 through Lemma 3. □

Lemma 1. $(ob, T, B) \rightarrow_a S \Rightarrow (pe, T, B) \rightarrow_a S$

Proof.

ASSUME: $(ob, T, B) \rightarrow_a S$
PROVE: $(pe, T, B) \rightarrow_a S$
$\langle 1 \rangle 1$. CASE: $\forall h \in S : \forall t \in T : t \not\lhd h$
PROOF: Adherence holds trivially by Def. 7
$\langle 1 \rangle 2$. CASE: $\exists h \in S : \exists t \in T : t \lhd h$
$\langle 2 \rangle 1$. Choose arbitrary $h \in S$ and $t \in T$ such that $t \lhd h$
PROOF: The traces exist by case assumption
$\langle 2 \rangle 2$. $\exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge (\{t\} \gtrsim B) \lhd h'$
$\langle 3 \rangle 1$. $(\{t\} \gtrsim B) \lhd h$
PROOF: By assumption and Def. 7
$\langle 3 \rangle 2$. $\exists k \in \mathbb{N} : h|_k \sqsubseteq h \wedge t \lhd h|_k$
PROOF: Case assumption and choosing $k = \#h$
$\langle 3 \rangle 3$. Q.E.D.
PROOF: $\langle 3 \rangle 1$, $\langle 3 \rangle 2$ and choosing $h' = h$
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and Def. 7
$\langle 1 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

□

Lemma 2. $(ob, T, B) \rightarrow_a S \Rightarrow (\neg pe, T, \neg B) \rightarrow_a S$

Proof.

ASSUME: $(ob, T, B) \rightarrow_a S$
PROVE: $(\neg pe, T, \neg B) \rightarrow_a S$
$\langle 1 \rangle 1$. CASE: $\forall h \in S : \forall t \in T : t \not\lhd h$
PROOF: Adherence holds trivially by Def. 9
$\langle 1 \rangle 2$. CASE: $\exists h \in S : \exists t \in T : t \lhd h$
$\langle 2 \rangle 1$. Choose arbitrary $h \in S$ and $t \in T$ such that $t \lhd h$
PROOF: The traces exist by case assumption
$\langle 2 \rangle 2$. $\neg \exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge \neg((\{t\} \gtrsim B) \lhd h')$
$\langle 3 \rangle 1$. ASSUME: $\exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge \neg((\{t\} \gtrsim B) \lhd h')$
PROVE: $\bot$
$\langle 4 \rangle 1$. Choose arbitrary $h' \in S$ and $k \in \mathbb{N}$ such that $h|_k \sqsubseteq h'$ and $t \lhd h|_k$ and $(\{t\} \gtrsim B) \not\lhd h'$
PROOF: The trace and number exist by assumption $\langle 3 \rangle 1$
$\langle 4 \rangle 2$. $t \lhd h'$
PROOF: $\langle 4 \rangle 1$
$\langle 4 \rangle 3$. $(\{t\} \gtrsim B) \lhd h'$
PROOF: $\langle 4 \rangle 2$, assumption and Def. 7
$\langle 4 \rangle 4$. Q.E.D.
PROOF: $\langle 4 \rangle 1$ and $\langle 4 \rangle 3$
$\langle 3 \rangle 2$. Q.E.D.
PROOF: By contradiction
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and Def. 9
$\langle 1 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

□

Lemma 3. $(ob, T, B) \rightarrow_a S \Leftarrow (\neg pe, T, \neg B) \rightarrow_a S$

Proof.

ASSUME: $(\neg pe, T, \neg B) \rightarrow_a S$
PROVE: $(ob, T, B) \rightarrow_a S$
$\langle 1 \rangle 1$. CASE: $\forall h \in S : \forall t \in T : t \not\lhd h$
PROOF: Adherence holds trivially by Def. 7
$\langle 1 \rangle 2$. CASE: $\exists h \in S : \exists t \in T : t \lhd h$
$\langle 2 \rangle 1$. Choose arbitrary $h \in S$ and $t \in T$ such that $t \lhd h$
PROOF: The traces exist by case assumption
$\langle 2 \rangle 2$. $(\{t\} \succsim B) \lhd h$
$\langle 3 \rangle 1$. $\forall h' \in S : \forall k \in \mathbb{N} : (h|_k \sqsubseteq h' \wedge t \lhd h|_k \Rightarrow (\{t\} \succsim B) \lhd h')$
PROOF: By assumption and Def. 9
$\langle 3 \rangle 2$. Q.E.D.
PROOF: $\langle 3 \rangle 1$ with $h' = h$ and $k = \#h$
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and Def. 7
$\langle 1 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

□

Theorem 2. If $d_1 \rightsquigarrow d_1'$ and $d_2 \rightsquigarrow d_2'$, then the following hold.

– d_1 seq $d_2 \rightsquigarrow d_1'$ seq d_2'
– d_1 alt $d_2 \rightsquigarrow d_1'$ alt d_2'
– d_1 par $d_2 \rightsquigarrow d_1'$ par d_2'

Proof. Since refinement is defined in terms of the relations $\subseteq$ and $\supseteq$, the theorem is proven by showing the monotonicity of these relations with respect to the operators $\succsim$, $\cup$ and $\|$ defining seq, alt and par, respectively. The proofs are given in Lemma 4 through Lemma 9. □

Lemma 4. Monotonicity of $\subseteq$ with respect to $\succsim$.

$$H_1 \subseteq H_1' \wedge H_2 \subseteq H_2' \Rightarrow H_1 \succsim H_2 \subseteq H_1 \succsim H_2'$$

Proof. Lemma 27 in [8]. □

Lemma 5. Monotonicity of $\supseteq$ with respect to $\succsim$.

$$H_1 \supseteq H_1' \wedge H_2 \supseteq H_2' \Rightarrow H_1 \succsim H_2 \supseteq H_1' \succsim H_2'$$

Proof.

ASSUME: 1. $H_1 \supseteq H_1'$
2. $H_2 \supseteq H_2'$
PROVE: $H_1 \succsim H_2 \supseteq H_1' \succsim H_2'$
⟨1⟩1. CASE: $H_1' \succsim H_2' = \emptyset$
PROOF: Trivial as $H \supseteq \emptyset$ for all sets H
⟨1⟩2. CASE: $H_1' \succsim H_2' \neq \emptyset$
⟨2⟩1. Choose arbitrary $h \in H_1' \succsim H_2'$
PROOF: Non-empty by case assumption
⟨2⟩2. $h \in H_1 \succsim H_2$
⟨3⟩1. Choose $h_1 \in H_1'$ and $h_2 \in H_2'$ such that
$\forall l \in \mathcal{L} : e.l\circledS h = e.l\circledS h_1 \frown e.l\circledS h_2$
PROOF: ⟨2⟩1 and Def. 2 of $\succsim$
⟨3⟩2. $h_1 \in H_1$
PROOF: ⟨3⟩1 and assumption 1
⟨3⟩3. $h_2 \in H_2$
PROOF: ⟨3⟩1 and assumption 2
⟨3⟩4. Q.E.D.
PROOF: ⟨3⟩1, ⟨3⟩2, ⟨3⟩3 and Def. 2 of $\succsim$
⟨2⟩3. Q.E.D.
PROOF: ⟨2⟩1, ⟨2⟩2 and definition of $\supseteq$
⟨1⟩3. Q.E.D.
PROOF: The cases are exhaustive

□

Lemma 6. Monotonicity of $\subseteq$ with respect to $\cup$.

$$H_1 \subseteq H_1' \wedge H_2 \subseteq H_2' \Rightarrow H_1 \cup H_2 \subseteq H_1' \cup H_2'$$

Proof. The result follows immediately from the definitions of $\subseteq$ and $\cup$. □

Lemma 7. Monotonicity of $\supseteq$ with respect to $\cup$.

$$H_1 \supseteq H_1' \wedge H_2 \supseteq H_2' \Rightarrow H_1 \cup H_2 \supseteq H_1' \cup H_2'$$

Proof. The result follows immediately from the definitions of $\supseteq$ and $\cup$. □

Lemma 8. Monotonicity of $\subseteq$ with respect to $\|$.

$$H_1 \subseteq H_1' \wedge H_2 \subseteq H_2' \Rightarrow H_1 \parallel H_2 \subseteq H_1' \parallel H_2'$$

Proof. Lemma 28 in [8]. □

Lemma 9. Monotonicity of $\supseteq$ with respect to $\|$.

$$H_1 \supseteq H_1' \wedge H_2 \supseteq H_2' \Rightarrow H_1 \parallel H_2 \supseteq H_1' \parallel H_2'$$

Proof.

ASSUME: 1. $H_1 \supseteq H_1'$
2. $H_2 \supseteq H_2'$
PROVE: $H_1 \parallel H_2 \supseteq H_1' \parallel H_2'$
$\langle 1 \rangle 1$. CASE: $H_1' \parallel H_2' = \emptyset$
PROOF: Trivial as $H \supseteq \emptyset$ for all sets H
$\langle 1 \rangle 2$. CASE: $H_1' \parallel H_2' \neq \emptyset$
$\langle 2 \rangle 1$. Choose arbitrary $h \in H_1' \parallel H_2'$
PROOF: Non-empty by case assumption
$\langle 2 \rangle 2$. $h \in H_1 \parallel H_2$
$\langle 3 \rangle 1$. Choose $s \in \{1,2\}^\infty$ such that
$\pi_2((\{1\} \times \mathcal{E}) \circledT (s,h)) \in H_1'$ and
$\pi_2((\{2\} \times \mathcal{E}) \circledT (s,h)) \in H_2'$
PROOF: $\langle 2 \rangle 1$ and Def. 1 of $\|$
$\langle 3 \rangle 2$. $\pi_2((\{1\} \times \mathcal{E}) \circledT (s,h)) \in H_1$
PROOF: $\langle 3 \rangle 1$ and assumption 1
$\langle 3 \rangle 3$. $\pi_2((\{2\} \times \mathcal{E}) \circledT (s,h)) \in H_2$
PROOF: $\langle 3 \rangle 1$ and assumption 2
$\langle 3 \rangle 4$. Q.E.D.
PROOF: $\langle 3 \rangle 1$, $\langle 3 \rangle 2$ and $\langle 3 \rangle 3$ and Def. 1 of $\|$
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and definition of $\supseteq$
$\langle 1 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

□

Theorem 3. Given a system S and policy specifications P and P', if $P \rightsquigarrow P'$ and $P' \rightarrow_a S$, then $P \rightarrow_a S$.

Proof.

ASSUME: 1. $P \rightsquigarrow P'$
2. $P' \rightarrow_a S$
PROVE: $P \rightarrow_a S$
$\langle 1 \rangle 1$. CASE: $P = \emptyset$
PROOF: Adherence holds trivially by Def. 8
$\langle 1 \rangle 2$. CASE: $P \neq \emptyset$
$\langle 2 \rangle 1$. Choose arbitrary $r \in P$
PROOF: The rule exists by case assumption
$\langle 2 \rangle 2$. $r \rightarrow_a S$
$\langle 3 \rangle 1$. Choose $r' \in P'$ such that $r \rightsquigarrow r'$
PROOF: The rule exist by assumption 1 and Def. 12 of policy refinement
$\langle 3 \rangle 2$. $r' \rightarrow_a S$
PROOF: Assumption 2 and Def. 8 of policy adherence
$\langle 3 \rangle 3$. Q.E.D.
PROOF: $\langle 3 \rangle 1$, $\langle 3 \rangle 2$ and Lemma 10, Lemma 11 and Lemma 12 for the respective modalities of r
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and Def. 8 of policy adherence
$\langle 1 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

□

Lemma 10.

$$(pe, T, B) \rightsquigarrow (pe, T', B') \wedge (pe, T', B') \rightarrow_a S \Rightarrow (pe, T, B) \rightarrow_a S$$

Proof.

ASSUME: 1. $(pe, T, B) \rightsquigarrow (pe, T', B')$
2. $(pe, T', B') \rightarrow_a S$
PROVE: $(pe, T, B) \rightarrow_a S$
$\langle 1 \rangle 1$. CASE: $\forall h \in S : \forall t \in T : t \not\lhd h$
PROOF: Adherence holds trivially by Def. 7
$\langle 1 \rangle 2$. CASE: $\exists h \in S : \exists t \in T : t \lhd h$
$\langle 2 \rangle 1$. Choose arbitrary $h \in S$ and $t \in T$ such that $t \lhd h$
PROOF: The traces exist by case assumption
$\langle 2 \rangle 2$. $\exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge (\{t\} \succsim B) \lhd h'$
$\langle 3 \rangle 1$. $t \in T'$
PROOF: $\langle 2 \rangle 1$, assumption 1 and Def. 11 of rule refinement
$\langle 3 \rangle 2$. $\exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge (\{t\} \succsim B') \lhd h'$
PROOF: $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, assumption 2 and Def. 7 of adherence
$\langle 3 \rangle 3$. $\exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge \exists h'' \in (\{t\} \succsim B') : h'' \lhd h'$
PROOF: $\langle 3 \rangle 2$ and Def. 4 of $\lhd$
$\langle 3 \rangle 4$. $(\{t\} \succsim B') \subseteq (\{t\} \succsim B)$
PROOF: Assumption 1, Def. 11 of rule refinement and Lemma 4 of monotonicity of $\subseteq$ wrt. $\succsim$
$\langle 3 \rangle 5$. $\exists h' \in S : \exists k \in \mathbb{N} : h|_k \sqsubseteq h' \wedge t \lhd h|_k \wedge \exists h'' \in (\{t\} \succsim B) : h'' \lhd h'$
PROOF: $\langle 3 \rangle 3$ and $\langle 3 \rangle 4$
$\langle 3 \rangle 6$. Q.E.D.
PROOF: $\langle 3 \rangle 5$ and Def. 4 of $\lhd$
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and Def. 7 of adherence
$\langle 1 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

□

Lemma 11.

$$(ob, T, B) \rightsquigarrow (ob, T', B') \wedge (ob, T', B') \rightarrow_a S \Rightarrow (ob, T, B) \rightarrow_a S$$

Proof.

ASSUME: 1. $(ob, T, B) \rightsquigarrow (ob, T', B')$
2. $(ob, T', B') \rightarrow_a S$
PROVE: $(ob, T, B) \rightarrow_a S$
$\langle 1 \rangle 1$. CASE: $\forall h \in S : \forall t \in T : t \not\lhd h$
PROOF: Adherence holds trivially by Def. 7
$\langle 1 \rangle 2$. CASE: $\exists h \in S : \exists t \in T : t \lhd h$
$\langle 2 \rangle 1$. Choose arbitrary $h \in S$ and $t \in T$ such that $t \lhd h$
PROOF: The traces exist by case assumption
$\langle 2 \rangle 2$. $(\{t\} \succsim B) \lhd h$
$\langle 3 \rangle 1$. $t \in T'$
PROOF: $\langle 2 \rangle 1$, assumption 1 and Def. 11 of rule refinement
$\langle 3 \rangle 2$. $(\{t\} \succsim B') \lhd h$
PROOF: $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, assumption 2 and Def. 7 of adherence
$\langle 3 \rangle 3$. $(\{t\} \succsim B') \subseteq (\{t\} \succsim B)$
PROOF: Assumption 1, Def. 11 of rule refinement and Lemma 4 of monotonicity of $\subseteq$ wrt. $\succsim$
$\langle 3 \rangle 4$. Q.E.D.
PROOF: $\langle 3 \rangle 2$, $\langle 3 \rangle 3$ and Def. 4 of $\lhd$
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and Def. 7 of adherence
$\langle 1 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

□

Lemma 12.

$$(pr, T, B) \rightsquigarrow (pr, T', B') \wedge (pr, T', B') \rightarrow_a S \Rightarrow (pr, T, B) \rightarrow_a S$$

Proof.

ASSUME: 1. $(pr, T, B) \rightsquigarrow (pr, T', B')$
2. $(pr, T', B') \rightarrow_a S$
PROVE: $(pr, T, B) \rightarrow_a S$
$\langle 1 \rangle 1$. CASE: $\forall h \in S : \forall t \in T : t \not\lhd h$
PROOF: Adherence holds trivially by Def. 7
$\langle 1 \rangle 2$. CASE: $\exists h \in S : \exists t \in T : t \lhd h$
$\langle 2 \rangle 1$. Choose arbitrary $h \in S$ and $t \in T$ such that $t \lhd h$
PROOF: The traces exist by case assumption
$\langle 2 \rangle 2$. $(\{t\} \succsim B) \not\trianglelefteq h$
$\langle 3 \rangle 1$. $t \in T'$
PROOF: $\langle 2 \rangle 1$, assumption 1 and Def. 11 of rule refinement
$\langle 3 \rangle 2$. $(\{t\} \succsim B') \not\trianglelefteq h$
PROOF: $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, assumption 2 and Def. 7 of adherence
$\langle 3 \rangle 3$. $(\{t\} \succsim B') \supseteq (\{t\} \succsim B)$
PROOF: Assumption 1, Def. 11 of rule refinement and Lemma 5 of monotonicity of $\supseteq$ wrt. $\succsim$
$\langle 3 \rangle 4$. Q.E.D.
PROOF: $\langle 3 \rangle 2$, $\langle 3 \rangle 3$ and Def. 5 of $\not\trianglelefteq$
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and Def. 7 of adherence
$\langle 1 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

□