Softw Syst Model
DOI 10.1007/s10270-008-0102-3

EXPERT'S VOICE

Adherence preserving refinement of trace-set properties in STAIRS: exemplified for information flow properties and policies

Fredrik Seehusen · Bjørnar Solhaug · Ketil Stølen

Received: 17 July 2008 / Accepted: 18 July 2008

Abstract STAIRS is a formal approach to system development with UML 2.1 sequence diagrams that supports an incremental and modular development process. STAIRS is underpinned by denotational and operational semantics that have been proved to be equivalent. STAIRS is more expressive than most approaches with a formal notion of refinement. STAIRS supports a stepwise refinement process under which trace properties as well as trace-set properties are preserved. This paper demonstrates the potential of STAIRS in this respect, in particular that refinement in STAIRS preserves adherence to information flow properties as well as policies.

1 Introduction

The denotational semantics of STAIRS [9,33] is a formalization of the trace semantics for sequence diagrams that is informally described in the UML 2.1 standard [27]. A trace is a sequence of event occurrences (referred to as "events" in the following) ordered by time that describes an execution history, and the semantics of a sequence diagram is defined in the standard by a set of valid traces and a set of invalid traces.

A system implementation, i.e., an existing system, can be characterized by the set of traces representing all the possible runs of the system. Such a representation can serve as a basis for analysis as well as verification or falsification of different properties, e.g., safety and security, that the system is supposed to possess or not to possess. Property falsification can be classified into falsification on the basis of a single trace and falsification on the basis of a set of traces [25].

Properties that can be falsified on single traces are the kind of properties that were originally investigated by Alpern and Schneider [2,35] and include safety and liveness properties. Properties that can only be falsified on sets of traces are of the type that McLean [25] referred to as possibilistic properties and include information flow properties and permission rules of policies.

In system development with refinement, system documentation is organized in a hierarchical manner. The various specifications within the hierarchy are related by notions of refinement. Each refinement step adds details to the specification, making it more concrete and closer to an implementation. This facilitates efficiency of analysis and verification through abstraction. Moreover, it supports early discovery of design flaws.

STAIRS distinguishes between two kinds of non-determinism; namely, *underspecification* and *inherent non-determinism*. Underspecification is a feature of abstraction and means specifying several behaviors, each representing a potential alternative serving the same purpose. The fulfillment of only some of them (at least one) is acceptable for an implementation to be correct. The other kind of non-determinism is an explicit, inherent non-determinism that a correct implementation is required to possess. A simple example is the non-determinism between heads and tails in

Communicated by Prof. Bernhard Rumpe.

F. Seehusen · B. Solhaug · K. Stølen (✉)
SINTEF ICT, Oslo, Norway
e-mail: Ketil.Stolen@sintef.no

F. Seehusen · K. Stølen
Department of Informatics,
University of Oslo, Oslo, Norway

B. Solhaug
Department of Information Science and Media Studies,
University of Bergen, Bergen, Norway

the specification of the game of tossing a coin. An implementation that offers only one of the alternatives is obviously not correct.

If an abstract system specification does not have the expressiveness to distinguish between these two kinds of non-determinism, refinement of trace-set properties is problematic [15,31]. Refinement will typically reduce underspecification and thereby reduce non-determinism. If there is no way to distinguish 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. In that case, although a trace-set property is verified to hold for the abstract specification, it may not hold for a system obtained by refinement from the abstract specification. The potential of trace-set properties not to be preserved under refinement is in [16] referred to as the refinement paradox.

The objective of this paper is twofold. Firstly, the paper presents the semantic foundation of STAIRS. The denotational semantics that captures both underspecification and inherent non-determinism is presented, along with a formal notion of refinement that preserves inherent non-determinism. Secondly and more importantly, the potential of the expressiveness of STAIRS to capture trace-set properties and preserve these under refinement is demonstrated within the domains of information flow security and policy based management, both of which require this expressiveness.

Generally, secure information flow properties provide a way of specifying security requirements, e.g., confidentiality, by selecting a set of domains, i.e., abstractions of real system entities such as users or files, and then restricting allowed flow of information between these domains. Many information flow properties are trace-set properties, and we show that such properties can be expressed within the STAIRS approach.

Policies define requirements to systems and are typically used for the management of security, networks, and services. A policy is defined as a set of policy rules. We show that some policies are trace-set properties and that they can be expressed in the STAIRS approach.

As illustrated in Fig. 1, we distinguish between system specifications (of which a system is a special case) on the one hand and properties/policies on the other hand. The notion of refinement relates different levels of abstractions on the system side and is therefore represented by the vertical arrows in Fig. 1. Adherence, on the other hand, is depicted by an arrow relating the left and right sides, and characterizes what it means for a system specification to fulfill a property/policy.

To the right in Fig. 1, S_1 denotes an initial, abstract system specification, whereas S_2 and S_3 denote refined and more concrete system specifications. We show that refinement as defined in STAIRS preserves adherence. Given that the specification S_1 adheres to P, and S_2 is a refinement of S_1, it follows by adherence preservation that also S_2 adheres to P, as illustrated by the upper left dashed arrow. The same is

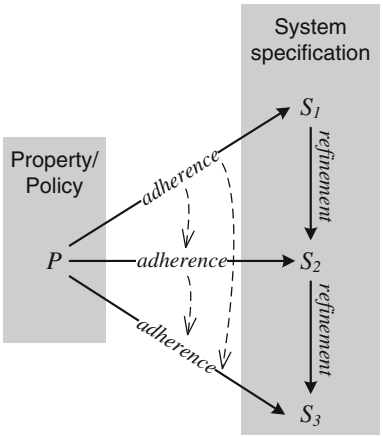


Fig. 1 Preservation of adherence under refinement

the case for refinement of S_2, resulting in the specification S_3. Moreover, the refinement relation is transitive, so if the development process begins with the specification S_1, and S_1 adheres to P, then adherence is preserved under any number of refinement steps. Hence, also the specification S_3 adheres to P, which motivates the dashed arrow from the uppermost adherence relation to the lowermost.

The structure of the paper is as follows. In Sect.2, we give an overview of the central concepts of the STAIRS approach and show how these are related to concepts of information flow security on the one hand and to concepts of policies on the other hand. In Sect. 3, we give an introduction to UML 2.1 sequence diagrams and the STAIRS denotational trace semantics. In Sect. 4, information flow properties are defined and formalized, information flow property adherence is defined, and it is shown that information flow property adherence is preserved under refinement. In Sect. 5, we introduce a syntax and a semantics for specifying policies in the setting of UML 2.1 sequence diagrams. We formally define the notion of policy adherence and show preservation of policy adherence under refinement. The results of the paper are related to other work in Sect. 6. Finally, in Sect. 7, we conclude. There is also an appendix with formal definitions of some notions and concepts that for the sake of readability are only defined informally in the main part of the paper.

2 Conceptual overview

The UML class diagram in Fig. 2 depicts the most important concepts and conceptual relations of relevance to this paper. In STAIRS, a *system specification* is given by a UML 2.1 sequence diagram. Semantically, the meaning of a sequence diagram is captured by a non-empty set of *interaction obligations*. Each interaction obligation represents one inherent choice of system behavior. The variation over interaction

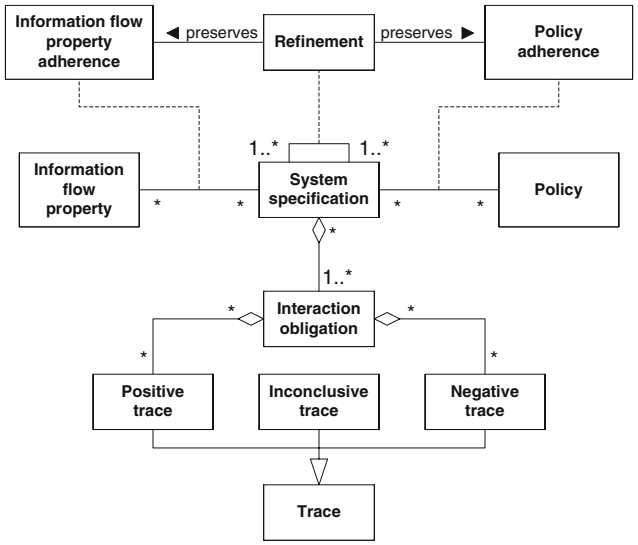


Fig. 2 Relationships of concepts

obligations captures the inherent non-determinism that is required for trace-set properties to be expressed.

Each interaction obligation is defined by a pair of trace sets, one positive and one negative. The set of positive traces defines valid or legal alternatives, and may capture underspecification since a correct implementation is only required to offer at least one of these traces. The set of negative traces defines invalid or illegal alternatives, and a correct implementation of an interaction obligation will never produce any of the negative traces. Notice that the positive traces in one interaction obligation may be negative in other interaction obligations. Hence, the scope of a negative trace is the interaction obligation in question.

In the UML 2.1 standard, a sequence diagram defines both valid and invalid traces that correspond to the positive and negative traces in STAIRS, respectively. The traces that are neither categorized as positive nor as negative are referred to as inconclusive. Inconclusive traces represent behavior that is irrelevant or uninteresting for the specification in question, at least for the given level of abstraction.

STAIRS relates system specifications via refinement. Hence, Fig. 2 depicts refinement as a relation between specifications.

The set of interaction obligations defined by a STAIRS system specification can be understood as a generalization of the set of traces representing a system; each interaction obligation must be fulfilled by at least one system run. Trace properties and trace-set properties in STAIRS are hence captured as properties of interaction obligations and sets of interaction obligations, respectively.

The upper left part of Fig. 2 shows that information flow property adherence is a relation between an information flow property and a system specification. Correspondingly, on the upper right part of the diagram, policy adherence is depicted as a relation between a policy and a system specification. The refinement relation between system specifications preserves both instances of adherence.

3 STAIRS

As background to the full treatment of trace-set properties and adherence, in this section, we give a more mathematical introduction to STAIRS with focus on the denotational semantics of sequence diagrams and the notion of refinement. We refer to [9,33] for the full treatment. See [21] for an equivalent operational semantics.

We first present the semantics of sequence diagrams without the STAIRS specific xalt operator that is used to specify inherent non-determinism. In the cases without the use of xalt, a single interaction obligation is sufficient. We then address the general situation with xalt, in which case the semantics of a sequence diagram becomes a set of interaction obligations. Finally, we define refinement.

3.1 Sequence diagrams without xalt

STAIRS formalizes UML sequence diagrams, and thus precisely defines the trace semantics that is only informally described in the UML 2.1 standard. Sequence diagrams specify system 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. An event is either the transmission or the reception of a message. In the STAIRS denotational semantics, a message is given by a triple (co, tr, re) of a message content co, 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 a kind and a message, $(k, m) \in \{!, ?\} \times \mathcal{M}$, where ! denotes that it is a transmit event and ? denotes that it is a receive event. By $\mathcal{E}$, we denote the set of all events, and $\mathcal{T}$ denotes the set of all traces.

The diagram *login* to the left in Fig. 3 is very basic and has only two events; the transmission of the message *login(id)* on the user lifeline and the reception of the same message on the server lifeline. The transmit event must obviously occur before the reception event. The diagram *login* therefore describes the single trace $\langle !l, ?l \rangle$ of these two events, where l is a shorthand for the message. Throughout the paper we will represent a message by the first letter in the message content for simplicity's sake.

The diagram *request* to the right in Fig. 3 shows the sequential composition of the transmission and reception of the two messages l and r. The order of the events on each lifeline

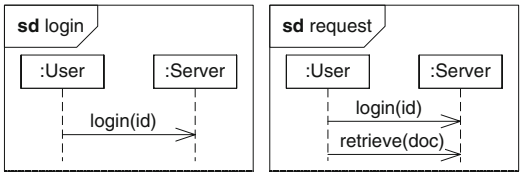
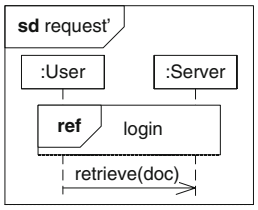


Fig. 3 Basic sequence diagrams

Fig. 4 Interaction use



is given by their vertical positions, but the two lifelines are independent. This corresponds to so-called weak sequencing. Weak sequencing defines a partial ordering of the events of the diagram, and requires that events on the same lifeline are ordered sequentially, and that the transmission of a message is ordered before its reception. For traces t_1 and t_2, $t_1 \gtrsim t_2$ is the set of traces obtained by their weak sequencing.[1] The weak sequencing in the diagram *request*, for example, is captured by

$$\langle !l, ?l \rangle \gtrsim \langle !r, ?r \rangle = \{\langle !l, ?l, !r, ?r \rangle, \langle !l, !r, ?l, ?r \rangle\}$$

The transmission of l is the first event to occur, but after that, both the reception of l and the transmission of r may occur. The $\gtrsim$ operator is lifted to sets of traces T_1 and T_2 in a pointwise manner as follows.

$$T_1 \gtrsim T_2 \stackrel{\text{def}}{=} \bigcup_{(t_1,t_2)\in T_1\times T_2} (t_1 \gtrsim t_2)$$

Notice that if T_1 or T_2 is $\emptyset$, the result is also $\emptyset$.

In the diagram *request'* in Fig. 4, we employ a construct referred to as an interaction use in the UML standard. The ref operator takes a sequence diagram as operand, which in this case is the diagram *login* to the left in Fig. 3. An interaction use is merely shorthand for the contents of the referred diagram, but is useful as it allows the reuse of the same specification in several different contexts. As a result, the diagram *request'* shows the weak sequencing of the transmission and reception of the two messages l and r. The diagram *request'* is hence semantically equivalent to the diagram *request* of Fig. 3.

In fact, the diagram *request"* of Fig. 5 is also equivalent to the diagram *request*. The only difference between *request* and *request"* is that in the latter, weak sequencing of the two message interchanges is stated explicitly by using the seq operator.

[1] For a formal definition of the weak sequencing operator $\gtrsim$, see Appendix A.2.

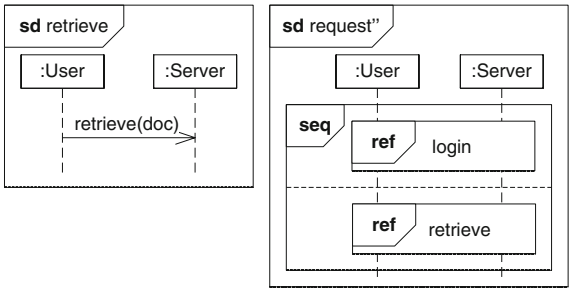
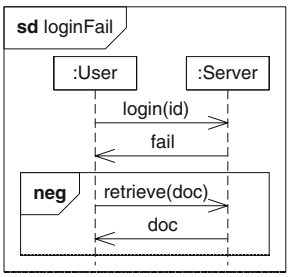


Fig. 5 Weak sequencing

Fig. 6 Negative interaction



Generally, a sequence diagram specifies both valid and invalid behavior that is semantically captured by a pair (p, n) of positive and negative traces, respectively. Such pairs are referred to in STAIRS as interaction obligations. Traces not specified as positive or negative, i.e., $\mathcal{T}\backslash(p \cup n)$, are referred to as inconclusive. For both diagrams in Fig. 3, the set of negative traces is the empty set $\emptyset$ since no behavior is specified as negative.

The neg construct may be used to specify behavior as negative, as exemplified in Fig. 6. This diagram specifies that following a login failure (positive scenario), the user is not allowed to retrieve documents from the server (negative scenario). The diagram is thus the weak sequencing of a positive and a negative scenario. In order to capture this formally, we lift the weak sequencing operator from sets of traces to interaction obligations as follows:

$$(p_1, n_1) \gtrsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \gtrsim p_2, (n_1 \gtrsim p_2) \cup (n_1 \gtrsim n_2) \cup (p_1 \gtrsim n_2))$$

Notice that any combination with a negative scenario is defined as negative.

To see what this definition means for the diagram in Fig. 6, consider first the negative scenario, i.e., the interaction within the scope of the neg operator. It defines exactly one negative trace; namely, $\langle !r, ?r, !d, ?d \rangle$. It also defines one positive trace; namely, the empty trace $\langle\rangle$, indicating that skipping the behavior specified by the negative scenario is valid. The interaction obligation corresponding to the negative scenario is therefore $(\{\langle\rangle\}, \{\langle !r, ?r, !d, ?d \rangle\})$.

The positive scenario of the diagram, i.e., the interaction above the neg operator, specifies a single positive trace,

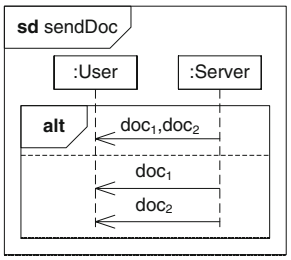


Fig. 7 Underspecification using alt

$\langle !l, ?l, !f, ?f \rangle$, and no negative traces. This gives the following interaction obligation.

$$(\{\langle !l, ?l, !f, ?f \rangle\}, \emptyset)$$

The semantics of the whole diagram in Fig. 6 is defined by weak sequencing of the interaction obligations corresponding to its positive and negative scenarios:

$$(\{\langle !l, ?l, !f, ?f \rangle\}, \emptyset) \gtrsim (\{\langle\rangle\}, \{\langle !r, ?r, !d, ?d \rangle\})$$

This yields an interaction obligation consisting of one positive trace and one negative trace:

$$(\{\langle !l, ?l, !f, ?f \rangle\}, \{\langle !l, ?l, !f, ?f, !r, ?r, !d, ?d \rangle\})$$

The positive traces of the sequence diagram *request* in Fig. 3 are equivalent in the sense that they both represent a valid way of fulfilling the behavior as described by the diagram. In STAIRS, this kind of non-determinism, often referred to as underspecification, may also be expressed directly by the alt operator. This is exemplified in the diagram *sendDoc* in Fig. 7, where two alternative ways of sending data from the server to the user is described. Both operands of the alt operator represent adequate alternatives of fulfilling the intended behavior of sending two documents to the user. It is left to the implementer of the server to decide whether the documents should be sent as one or two messages.

Semantically, the alt operator is captured by the operator $\uplus$ for inner union. Let (p_1, n_1) and (p_2, n_2) be the interaction obligations corresponding to the two operands of the alt. The semantics of the overall construct is then captured by $(p_1, n_1) \uplus (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \cup p_2, n_1 \cup n_2)$.

The *sendDoc* diagram has no negative traces since neither of the operands of the alt specify negative behavior. On the other hand, *sendDoc* has three positive traces, one from the first operand of the alt and two from the second, since the interleaving of $!d_2$ and $?d_1$ is left open.

Notice that a sequence diagram may be specified such that there is an overlap between the positive and the negative traces of the interaction obligation (p, n), i.e., $p \cap n \neq \emptyset$. In that case, the positive traces that are also negative are treated as negative. The valid or legal traces of a specification are represented by the set $p \backslash n$ and are referred to as the implementable traces of the interaction obligation.

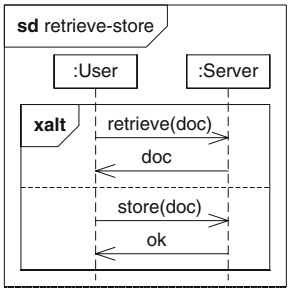


Fig. 8 Explicit alternatives

3.2 Sequence diagrams with xalt

In STAIRS, the alt operator captures non-deterministic choice in the meaning of underspecification, as already exemplified in Fig. 7. To specify inherent non-determinism, which may be understood as non-determinism the system is required to possess, we may use the xalt operator in STAIRS. This requires, however, a richer semantics. Instead of a single interaction obligation, we now need a set of interaction obligations to capture the meaning of a sequence diagram.

Figure 8 illustrates the use of the xalt operator. The specification says that the user can either retrieve data from or store data on the server. Importantly, the server must offer both alternatives.

Semantically, the xalt operator corresponds to ordinary union of the sets of interaction obligations for its operands. Since neither of the operands of this particular example have occurrences of xalt, they are singleton sets. The xalt construct then denotes the union of the two sets, i.e., $\{(\{\langle !r, ?r, !d, ?d \rangle\}, \emptyset), (\{\langle !s, ?s, !o, ?o \rangle\}, \emptyset)\}$.

The semantics of sequence diagrams is defined by a function $[\![\ ]\!]$ that for any given diagram d yields a set $[\![d]\!] = \{(p_1, n_1), \ldots, (p_m, n_m)\}$ of interaction obligations. It is formally defined as follows.

Definition 1 Semantics of sequence diagrams.

$$\begin{aligned}
[\![e]\!] &\stackrel{\text{def}}{=} \{(\{\langle e \rangle\}, \emptyset)\} \quad \text{for any } e \in \mathcal{E} \\
[\![\mathsf{seq}[d_1, d_2]]\!] &\stackrel{\text{def}}{=} \{o_1 \gtrsim o_2 \mid o_1 \in [\![d_1]\!] \wedge o_2 \in [\![d_2]\!]\} \\
[\![\mathsf{neg}[d]]\!] &\stackrel{\text{def}}{=} \{(\{\langle\rangle\}, p \cup n) \mid (p, n) \in [\![d]\!]\} \\
[\![\mathsf{alt}[d_1, d_2]]\!] &\stackrel{\text{def}}{=} \{o_1 \uplus o_2 \mid o_1 \in [\![d_1]\!] \wedge o_2 \in [\![d_2]\!]\} \\
[\![\mathsf{xalt}[d_1, d_2]]\!] &\stackrel{\text{def}}{=} [\![d_1]\!] \cup [\![d_2]\!]
\end{aligned}$$

3.3 Refinement

Refinement means adding information to a specification such that the specification becomes more complete and detailed. This may be achieved by reducing underspecification through redefining previously positive traces as negative or by redefining previously inconclusive traces as negative.

In STAIRS [9], there are also more general notions of refinement in which, for example, inconclusive traces may be

redefined as positive. These more general notions of refinement are mainly of relevance in very specific phases of system development and are not considered in this paper.

Refinement facilitates analysis through abstraction, but for analysis at an abstract level to be meaningful, the analysis results must be preserved under refinement. The refinement relation should also be transitive so as to support a stepwise development process ensuring that the final, most complete specification is a valid refinement of the initial specification. Modularity of the development process should also be supported by monotonicity of the composition operators with respect to refinement.

By $(p, n) \rightsquigarrow (p', n')$, we denote that the interaction obligation (p', n') is a refinement of the interaction obligation (p, n). Formally, refinement of interaction obligations is defined as follows.

Definition 2 Refinement of interaction obligations.

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

This means that in a refinement step, both inconclusive and positive traces can be redefined as negative. Once negative, a trace remains negative, while a positive trace cannot become inconclusive.

The variation over positive traces of an interaction obligation may reflect variations over the design choices under consideration. At later phases of the development, when the system in question is better understood and the specification is more complete, design choices are made by redefining some of the previously positive traces as negative. The negative traces of an interaction obligation explicitly state that these traces are held as invalid or illegal alternatives for fulfilling the interaction obligations. Hence, when design alternatives become more explicit through refinement, the negative traces remain negative.

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

Definition 3 Refinement of sequence diagrams.

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

By this definition, all interaction obligations of a specification must be represented by an interaction obligation in the refined specification, and adding interaction obligations that do not refine an interaction obligation of the abstract level is not allowed. This ensures that the inherent non-determinism expressed by the variation over interaction obligations is persistent through refinement.

Since trace-set properties are expressed as properties of sets of interaction obligations in STAIRS, the refinement relation ensures that trace-set properties are preserved under refinement by the preservation of interaction obligations.

The following theorem implies that the refinement relation supports a stepwise development process.

Theorem 1 *The refinement relation is transitive and reflexive. Formally, for all sequence diagrams d_1, d_2, and d_3,*

$$[\![\, d_1 \,]\!] \rightsquigarrow [\![\, d_2 \,]\!] \wedge [\![\, d_2 \,]\!] \rightsquigarrow [\![\, d_3 \,]\!] \Rightarrow [\![\, d_1 \,]\!] \rightsquigarrow [\![\, d_3 \,]\!]$$
$$[\![\, d_1 \,]\!] \rightsquigarrow [\![\, d_1 \,]\!]$$

For a formal proof of the theorem, see [34].

The following set of monotonicity results is established in [34]. These are important since they imply that the different parts of a sequence diagram can be refined separately.

Theorem 2 *If $[\![\, d_1 \,]\!] \rightsquigarrow [\![\, d_1' \,]\!]$ and $[\![\, d_2 \,]\!] \rightsquigarrow [\![\, d_2' \,]\!]$, then the following holds.*

$$\begin{aligned}[\![\, \mathsf{neg}[d_1] \,]\!] &\rightsquigarrow [\![\, \mathsf{neg}[d_1'] \,]\!] \\ [\![\, \mathsf{seq}[d_1, d_2] \,]\!] &\rightsquigarrow [\![\, \mathsf{seq}[d_1', d_2'] \,]\!] \\ [\![\, \mathsf{alt}[d_1, d_2] \,]\!] &\rightsquigarrow [\![\, \mathsf{alt}[d_1', d_2'] \,]\!] \\ [\![\, \mathsf{xalt}[d_1, d_2] \,]\!] &\rightsquigarrow [\![\, \mathsf{xalt}[d_1', d_2'] \,]\!]\end{aligned}$$

A sequence diagram is specified by composing sub-diagrams using the various operators. The monotonicity result, along with reflexivity, ensures that refinement of the sub-diagrams in isolation results in a valid refinement of the diagram as a whole. Moreover, the transitivity result means that each separate part of the sequence diagram can be subject to any number of refinement steps in order to refine the whole specification.

4 Secure information flow properties

Secure information flow properties impose requirements on information flow between different security domains. The underlying idea is that an observer residing in one security domain (call it low) shall not, based on its observations, be able to deduce whether behavior associated with another security domain (call it high) has, or has not occurred. Such requirements are referred to as *secure information flow properties* (or *information flow properties* for short).

Example 1 Suppose we require that the low-level user (*:UserL*) of Fig. 9 should not be able to deduce that the high-level user (*:UserH*) has done something. In the worst case, the low-level user has complete knowledge of the specification DS. If the low-level user makes an observation, he can use this knowledge to construct the set of all the positive traces that are compatible with that observation (the low-level equivalence set). He can conclude that one of the traces in this set has occurred, but not which one.

With respect to Fig. 9, the low-level user can make three observations: $\langle !s, ?o \rangle$, $\langle\rangle$, and $\langle !s, ?e \rangle$. If the observation

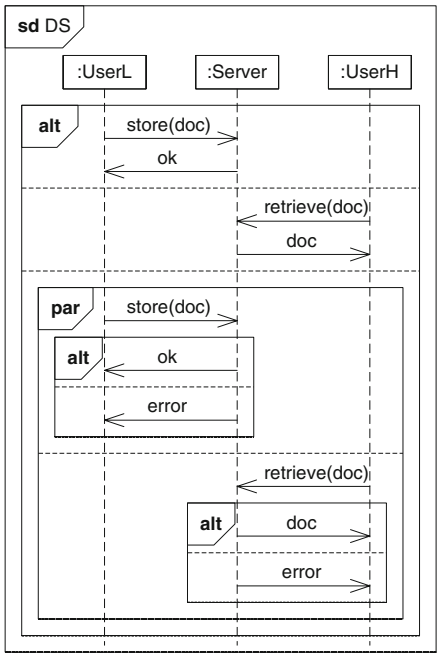


Fig. 9 Document storage and retrieval

$\langle !s, ?o \rangle$ is made, he can conclude that either the trace describing the top-most scenario of Fig. 9 has occurred or that one of the traces describing the lower-most scenario has occurred. The trace of the top-most scenario does not contain any events of the high-level user. Therefore, the low-level user cannot conclude with certainty that the high-level user has done something when the observation $\langle !s, ?o \rangle$ is made. However, if the low-level user makes the observation $\langle !s, ?e \rangle$, he can conclude with certainty that the high-level user has done something. This is because the low-level user receives an error message only when storing a document at the same time as the high-level user is attempting to retrieve the same document. Similarly, when observation $\langle\rangle$ is made, the low-level user knows for sure that the high-level user has done something. Hence, DS is not secure w.r.t. our requirement.

4.1 Capturing information flow properties

The above example indicates that we need two ingredients to define precisely what is meant by an information flow property:

- a notion of high-level behavior that should always be a possibility, which we represent by a predicate $H \in \mathcal{T} \rightarrow \mathbb{B}\text{ool}$ on traces; (this predicate may, as in the previous example, characterize all traces in which a high-level user has not done something);
- a set $L \subseteq \mathcal{E}$ of low-level events that can be observed by the low-level user.

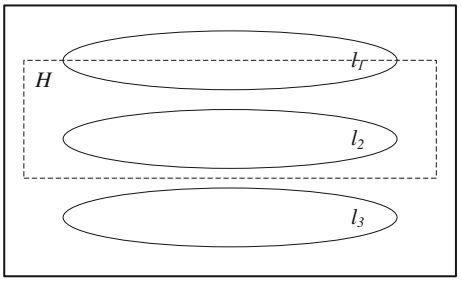


Fig. 10 Observation of high-level behavior

Intuitively, a system adheres to an information flow property (L, H) if for each trace t of the system, there is a system trace t' that satisfies H and is low-level equivalent to t. We say that two traces are low-level equivalent iff they cannot be distinguished by the observations made by the low-level user.

Let S be a system specification and suppose the low-level user has complete knowledge of S. The low-level user may then, for each (low-level) observation he can make of the system, use S to construct the set that contains traces that are compatible with that observation. He will know that one of the traces in this set has occurred, but not which one. However, if each trace that is compatible with a given (low-level) observation fulfills H, he can conclude with certainty that a high-level behavior has occurred.

This is illustrated in Fig. 10, where the outer rectangle depicts the set of all traces $\mathcal{T}$. The ellipses depict the sets of positive traces in S that are compatible with the observations the low-level user can make of the system. The low-level observations are described by the traces l_1, l_2, and l_3 of low-level events, so for all positive traces t in S, t is compatible with one of the observations l_1, l_2, or l_3. The dashed rectangle H depicts the high-level behavior. In this case, the set of traces compatible with the low-level observation l_2 is a subset of the high-level behavior. Hence, if the low-level user observes l_2, he knows with certainty that high-level behavior has taken place.

Likewise, if H is disjoint from a set of traces compatible with a low-level observation, the low-level user may with certainty deduce that no behavior characterized by H has taken place. In this case, information flows from the high-level domain to the low-level domain since the low-level user knows that the high-level behavior has not been conducted.

This is illustrated in Fig. 10 by the low-level observation l_3. Since all positive traces of S that are compatible with l_3 are not in H, the low-level user knows for sure that no high-level behavior has taken place by observing l_3.

To prevent the low-level user from deducing that high-level behavior has occurred or that high-level behavior has not occurred, there must for each low-level observation exist

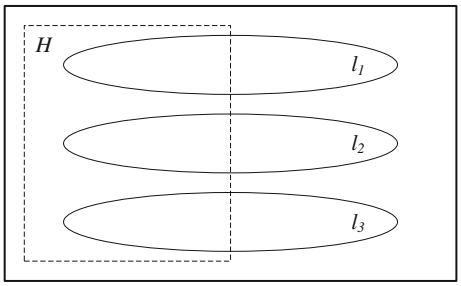


Fig. 11 No observation of high-level behavior

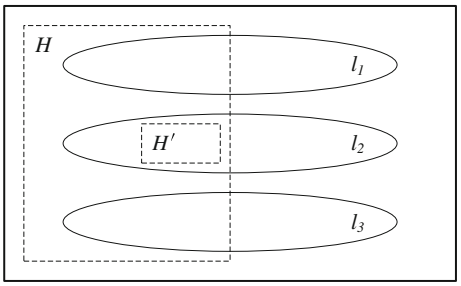


Fig. 12 Observing that high-level behavior has not occurred

both a compatible trace within H and a compatible trace within its complement $\overline{H}$.

This is illustrated in Fig. 11. For example, by observing the trace l_1 of low-level events, the low-level user cannot deduce whether or not high-level behavior has occurred since l_1 is compatible with both possibilities.

Example 2 The requirement of Example 1 is captured by the security property known as non-inference [28]. In order to formalize non-inference, we distinguish between the set L of events that, as already mentioned, can be observed by the low-level user and the set C of events that should be considered confidential, i.e., $L \cap C = \emptyset$.

Non-inference may now be captured by the information flow property (L, H), where the high level predicate is specified as follows.

$$H(t) \stackrel{\text{def}}{=} C \circledS t = \langle\rangle$$

For a trace t and a set of events E, the filtering operation $E \circledS t$ yields the trace obtained from t by filtering away every event not contained in E.[2]

In some cases, it is necessary to distinguish between several high-level behaviors in order to capture adequate information flow properties. Since the low-level user has full knowledge of the specification S, he may, for example, identify a subset H' of the high-level behavior characterized by H in which a specific high-level behavior is conducted, e.g., document retrieval. In Fig. 12, the set H' is within the intersection of H and the traces compatible with l_2. By observing l_1 or l_3, the low-level user can deduce with certainty that high-level behavior that fulfills H' has not occurred. In order to prevent this, we may specify a set of predicates, each characterizing a high-level behavior, and require that each of them are compatible with all low-level observations.

As a further illustration of the need to operate with a set of high-level behaviors, suppose we want to prevent a low-level user from deducing which login password has been chosen by a high-level user. The set H that characterizes all traces in which the high-level user has not chosen a password can

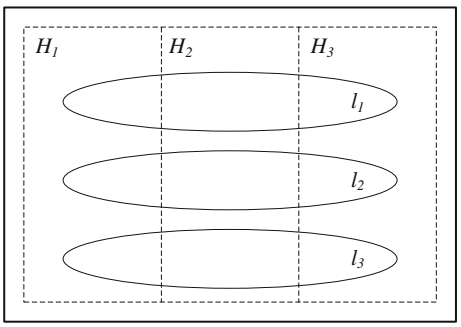


Fig. 13 Several high-level behaviors

be used to prevent the low-level user from deducing that some password has been selected, but we may be satisfied by preventing the low-level user from deducing which password has been selected. In that case, we need a predicate H for each password that can be selected.

This is illustrated in Fig. 13, where we distinguish between three different high-level behaviors, H_1, H_2 and H_3. The low-level user can make three different observations, and in each case may deduce that a high-level behavior has been conducted. However, the low-level user cannot deduce which one, since each of the observations l_1, l_2, and l_3 is compatible with all three high-level behaviors.

In the general case, we specify the high-level behaviors by a predicate on traces parameterized over the set of all traces, $H \in \mathcal{T} \to (\mathcal{T} \to \mathbb{B}\text{ool})$. Hence, the set of traces t' that fulfill the predicate $H(t)$ characterizes a specific high-level behavior dependent on t.

In some cases we may wish to impose requirements on information flow only if certain restrictions hold. We may, for example, wish to only prevent the low-level user from deducing that a particular document has been stored on a server if the low-level user has not been granted privileges to deduce this. Or we may wish to only prevent the low-level user from deducing that confidential events have not occurred if the occurrence is not influenced by the behavior of the low-level user. The latter assumption, for example, is made in the definition of the so-called perfect security property [44].

[2] For a formal definition of the filtering operator Ⓢ, see Appendix A.1.

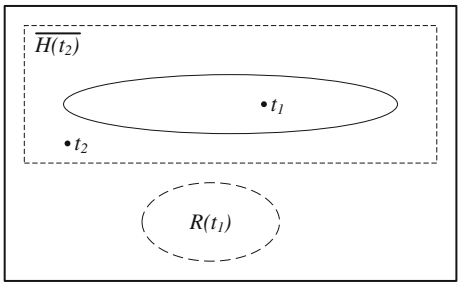


Fig. 14 Security requirement with restriction

Formally, the restriction is also specified by a predicate on traces parameterized over the set of all traces, $R \in \mathcal{T} \rightarrow (\mathcal{T} \rightarrow \mathbb{B}\text{ool})$. The idea is that if the low-level user makes the low-level observation corresponding to a trace t, we are then only concerned about the high-level behaviors of relevance for t, which are those generated from the traces that fulfill the restriction predicate $R(t)$.

This is illustrated in Fig. 14. The solid ellipse depicts the set that contains traces that are compatible with the low-level observation of the trace t_1, and the dashed rectangle represents those traces that are not high-level behaviors with respect to t_2. By observing the trace t_1, the low-level user can deduce with certainty that high-level behavior as characterized by $H(t_2)$ has not occurred.

Suppose, now, that the trace t_2 for some reason or another is not relevant when trace t_1 has occurred. Then the predicate $R(t_1)$ can be used to express this by defining $R(t_1)(t_2) \Leftrightarrow \mathit{False}$. Hence, with respect to the observation t_1, the trace t_2 is not held as relevant for the security requirement in question. With this weakening of the requirement, the situation depicted in Fig. 14 illustrates a secure case.

To summarize, inspired by [22], we specify a basic information flow property by a triple (R, L, H) of a restriction predicate $R \in \mathcal{T} \rightarrow (\mathcal{T} \rightarrow \mathbb{B}\text{ool})$, a set of low-level events $L \in \mathcal{E}$, and a high-level predicate $H \in \mathcal{T} \rightarrow (\mathcal{T} \rightarrow \mathbb{B}\text{ool})$. Intuitively, a system adheres to a basic information flow property (R, L, H) if, for any trace t_1, the system is secure with respect to any high-level predicate $H(t_2)$ such that $R(t_1)(t_2)$, i.e. any high-level predicate $H(t_2)$ of relevance for t_1. We specify an information flow property by a set P of such triples, each specifying a basic information flow property.

Example 3 In the following, we demonstrate how to specify an information flow property using the approach introduced above.

Assume we want to capture the following property: the low-level user should not be able to deduce (A) whether high-level behavior has occurred and (B) whether a particular high-level behavior has not occurred.

We formalize the requirements (A) and (B) by the triples (R_A, L, H_A) and (R_B, L, H_B), respectively. Let $L \subseteq \mathcal{E}$ be the set of events that has the low-level user as transmitter or receiver, and $C \subseteq \mathcal{E}$ be the set of confidential events, i.e., the events that have the high-level user as transmitter or receiver. For simplicity's sake, we assume that $L \cup C = \mathcal{E}$. The triple (R_A, L, H_A), may then be specified as in Example 2 with $R_A(t_1)(t_2) \Leftrightarrow \mathit{True}$ for all traces t_1, t_2.

Requirement (B) asserts not only the occurrence of high-level events as confidential, but also the particular sequence of high-level events in a trace. This means that we must also take relative timing into consideration. The traces $\langle l, c\rangle$ and $\langle c, l\rangle$ (where $l \in L$ and $c \in C$), for example, should not be considered equal at the high-level since c occurs at different points in the two traces.

We specify H_B in terms of a function $h \in \mathcal{T} \rightarrow \mathcal{T}$ that yields the sequence of high-level events of a trace in which relative timing is reflected. $h(t)$ is defined as the trace obtained from t by replacing each low-level event in t by a dummy event. The trace is truncated at the point in which the last high-level event occurs. Formally, the function h is defined by the following conditional equations.

$$
\begin{array}{lll}
 & h(\langle\rangle) & = \langle\rangle \\
C \circledS t = \langle\rangle \Rightarrow & h(t) & = \langle\rangle \\
e \in L \quad \Rightarrow & h(\langle e\rangle \frown t) & = \langle\surd\rangle \frown h(t) \\
e \notin L \quad \Rightarrow & h(\langle e\rangle \frown t) & = \langle e\rangle \frown h(t)
\end{array}
$$

Let us say, for example, that the events occurring on the *:UserH* lifeline in Fig. 9 are high-level and that all other events are low-level. For the uppermost scenario, we then have $h(\langle !s, ?s, !o, ?o\rangle) = \langle\rangle$. The lowermost scenario is specified by a parallel composition of two sub-scenarios that yields all possible interleavings as its semantics. The following are two examples of high-level sequences in which timing is considered.

$$h(\langle !s, ?s, !r, !o, ?r, ?o, !d, ?d\rangle) = \langle\surd, \surd, !r, \surd, \surd, \surd, \surd, ?d\rangle$$
$$h(\langle !r, !s, ?r, !d, ?s, ?d, !o, ?o\rangle) = \langle !r, \surd, \surd, \surd, \surd, ?d\rangle$$

With the help of h, H_B may then be specified as follows.

$$H_B(t)(t') \stackrel{\text{def}}{=} h(t) = h(t')$$

The restriction is that we are only concerned about the high-level behaviors that are not influenced by the low-level user. This means that we are only interested in those traces t_2 for which there exists a trace t_1 with a common prefix with t_2 and whose first deviation from t_1 involves a high-level event. This restriction may be formalized as follows.

$$
\begin{array}{l}
R_B(t_1)(t_2) \stackrel{\text{def}}{=} \\
\exists t \in \mathcal{T} : \exists e \in C : t \sqsubseteq t_1 \wedge t \frown \langle e\rangle \sqsubseteq t_2 \wedge t \frown \langle e\rangle \not\sqsubseteq t_1
\end{array}
$$

The expression $t' \sqsubseteq t$ yields true iff t' is a prefix of or equal to t.[3] If low-level behavior influences high-level behavior—for example, if low-level events are recorded at the high-level—the knowledge of this high-level behavior does not

[3] For a formal definition of the prefix relation $\sqsubseteq$ and its negation $\not\sqsubseteq$, see Appendix A.1.

represent a security breach since it is information flow that is already allowed. In the specification of the predicate R_B, these situations are ruled out.

The triple (R_B, L, H_B) requires that all observations the low-level user can make of the system are compatible with all possible sequences of high-level events in which relative timing is taken into consideration, provided the high-level behavior is not influenced by the low-level user.

4.2 Information flow property adherence

So far we have explained what it means for a system to adhere to an information flow property. In the following, we formally define what it means for a specification that is preserved under refinement to adhere to an information flow property.

Adherence of a system specification S to an information flow property P means that the specification S satisfies the given property, i.e., S is secure w.r.t the set P of basic information flow properties. This is captured by the adherence relation $\rightarrow_a$ and denoted $P \rightarrow_a S$. In order to formally define this relation, we first define what it means that a system specification adheres to a basic information flow property $(R, L, H) \in P$, denoted $(R, L, H) \rightarrow_a S$. In the following, we let $\widehat{S}$ denote the set of all implementable traces of the specification S, i.e.,

$$\widehat{S} = \bigcup_{(p,n)\in[\![S]\!]} p \backslash n$$

Definition 4 Adherence to basic information flow property (R, L, H) of system specification S.

$$\begin{aligned}&(R, L, H) \rightarrow_a S \stackrel{\text{def}}{=}\\ &\quad \forall t_1, t_2 \in \widehat{S} : R(t_1)(t_2) \Rightarrow\\ &\quad \exists (p, n) \in [\![\ S\ ]\!] : \forall t_3 \in (p \backslash n) : L \circledS t_1 = L \circledS t_3 \wedge H(t_2)(t_3)\end{aligned}$$

Observe firstly that this definition captures the requirement that a basic information flow property applies only for high-level properties of relevance for t_1, i.e., those $H(t_2)$ such that the antecedent $R(t_1)(t_2)$ is satisfied. Secondly, adherence to a basic information flow property is assured not by a single trace, but by an interaction obligation (p, n). Since interaction obligations are preserved by refinement, so is adherence. In summary, $R(t_1)$ identifies the set of high-level behaviors of relevance for t_1. To make sure that each of these high-level behaviors of relevance are fulfilled by the specification and maintained through later refinements, there must exist for each behavior an interaction obligation whose implementable traces all fulfill the high-level behavior and are indistinguishable from t_1 for the low-level user.

Adherence to an information flow property is now defined as follows.

Definition 5 Adherence to an information flow property P of system specification S.

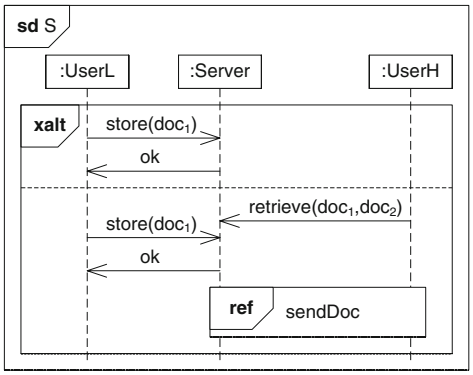


Fig. 15 System specification

$$P \rightarrow_a S \stackrel{\text{def}}{=} \forall (R, L, H) \in P : (R, L, H) \rightarrow_a S$$

Example 4 Consider the system specification S depicted in Fig. 15 showing low-level users and high-level users interacting with a server. The referred diagram *sendDoc* is depicted in Fig. 7, where *:User* is substituted by *:UserH*.

Assume that the property of non-inference as expressed in Example 2 is imposed as a requirement. In this example, L (resp. C) is the set of events that has *:UserL* (resp. *:UserH*) as transmitter or receiver.

The semantics of the specification in Fig. 15 is given by two interaction obligations in which there are no negative traces. The interaction obligation of the upper operand of the xalt operator consists of the singleton set $\{\langle !s, ?s, !o, ?o\rangle\}$ of positive traces, where the low-level user observes the trace $\langle !s, ?o\rangle$. The interaction obligation of the lower operand consists of several positive traces, all of which have high-level events and are observed as $\langle !s, ?o\rangle$ by the low-level user. Adherence is ensured by the former interaction obligation. Its only trace has no high-level events and is low-level equivalent to all positive traces in the specification.

Definition 4 demonstrates the STAIRS expressiveness to capture trace-set properties. Information flow property adherence requires the existence of an interaction obligation, the positive traces of which all comply with the high-level behavior. Adherence to an information flow property cannot be falsified by a single interaction obligation since compliance to the high-level behavior may be ensured by other interaction obligations of the specification.

4.3 Preservation of information flow properties under refinement

As illustrated in Fig. 1, where an information flow property is depicted to the left and refinements of system specifications are depicted to the right, the refinement relation preserves information flow property adherence. This result is captured by the following theorem.

Theorem 3 *Preservation of information flow property adherence under refinement.*

$$P \rightarrow_a S_1 \wedge S_1 \rightsquigarrow S_2 \Rightarrow P \rightarrow_a S_2$$

Proof Assume $P \rightarrow_a S_1$ and $S_1 \rightsquigarrow S_2$. Prove $P \rightarrow_a S_2$.

Let t_1 and t_2 be any elements of $\widehat{S_2}$ such that $R(t_1)(t_2)$. By Definition 3 and the assumption that $S_1 \rightsquigarrow S_2$, $t_1, t_2 \in \widehat{S_1}$. Since $R(t_1)(t_2)$ and $P \rightarrow_a S_1$ by assumption, $\exists (p, n) \in S_1 : \forall t \in (p \backslash n) : L(\circledS) t_1 = L(\circledS) t \wedge H(t_2)(t)$.

By Definition 3, $\exists (p', n') \in S_2 : (p, n) \rightsquigarrow (p', n')$. By Definition 2, $(p' \backslash n') \subseteq (p \backslash n)$, so $\forall t' \in (p' \backslash n') : L(\circledS) t_1 = L(\circledS) t' \wedge H(t_2)(t')$, i.e., $P \rightarrow_a S_2$ as desired. □

Importantly, by Theorem 3, if adherence is verified at an abstract level, this need not be verified again at the refined levels. Hence, STAIRS does not suffer from the so-called refinement paradox [15,16] in which trace-set properties are not preserved under refinement. In the following, we explain in more detail why this is so.

Underspecification arises when a term has several valid interpretations. The standard notion of refinement by underspecification [11] states that a system specification S' describing a set of traces T' is a refinement of a system specification S describing a set of traces T iff $T' \subseteq T$.

Intuitively, there are at least as many implementations that satisfy S as there are implementations that satisfy S'. In this sense, S' describes its set of implementations more accurately than S, i.e., S' is equally or less abstract than S. Adherence to secure information flow properties is, however, in general not preserved under this standard notion of refinement.

A system is secure with respect to an information flow property P if for all traces t_1 and t_2 such that $R(t_1)(t_2)$, there is a non-empty set of traces fulfilling $H(t_2)$. However, by defining refinement as set inclusion, there is no guarantee that a given refinement will keep any of the traces described by $H(t_2)$ at the abstract level. Hence, information flow properties are in general not preserved by the standard notion of refinement.

Intuitively, the cause of the problem is that information flow properties depend on unpredictability. The strength of one's password, for example, may be measured in terms of how hard it is for an attacker to guess the chosen password. The demand for the presence of traces fulfilling $H(t_2)$ may be seen as a requirement of unpredictability, but traces that provide this unpredictability may be removed during refinement (if we use the standard notion). This motivates Definition 4 of adherence to a basic security predicate in which the distinction between underspecification and unpredictability is taken into consideration by requiring the existence of an interaction obligation whose non-negative traces all fulfill $H(t_2)$. The STAIRS expressiveness to specify inherent nondeterminism as variation over interaction obligations captures this. Furthermore, the preservation of this variation under refinement ensures preservation of information flow property adherence, as formalized by Theorem 3.

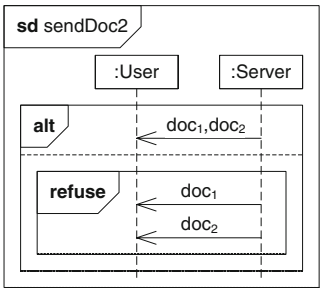


Fig. 16 Refined interaction

Example 5 We showed in Example 4 that the system specification S depicted in Fig. 15 adheres to the information flow property of non-inference. Assume, now, a specification S' in which *sendDoc* of Fig. 7 is replaced with *sendDoc2* of Fig. 16.

The refuse operator is introduced in STAIRS as a variant of the UML neg operator. Semantically, the difference from neg is that rather than defining the singleton set $\{\langle\rangle\}$ of the empty trace as positive, refuse yields the empty set $\emptyset$ as the set of positive traces. Introducing the refuse operator is motivated by the fact that the neg operator is ambiguously defined in the UML standard. See [32] for further details on this issue.

In *sendDoc2*, the alternative of sending the two requested documents as separate messages, which is positive in *sendDoc*, is specified as negative. It is easy to see that *sendDoc* $\rightsquigarrow$ *sendDoc2*. By the monotonicity results of Theorem 2, since *sendDoc* $\rightsquigarrow$ *sendDoc2*, then also $S \rightsquigarrow S'$.

By the same observations as in Example 4, it can be verified that for each positive trace of the specification, there exists an interaction obligation whose traces are low-level equivalent and have no high-level events. Again, the upper operand of the xalt operator ensures adherence to the property of non-inference.

5 Policies

During the last decade, policy based management of information systems has been subject to increased attention, and several frameworks (see e.g., [38]) have been introduced for the purpose of policy specification, analysis, and enforcement. At the same time, UML 2.1 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 of policies and for the analysis of policies in a UML setting.

In the following sections, we suggest how the UML in combination with STAIRS can be utilized for policy

specification. Furthermore, given a UML specification of a system for which a policy applies, we formally express what should be the relation between the policy specification and the system specification, i.e., we formally define the relation of policy adherence.

In Sect. 5.1, we introduce the syntax and semantics of our sequence diagram notation for policy specification. In Sect. 5.2, we define the notion of policy adherence for system specifications and show that policy adherence involves trace-set properties. In Sect. 5.3, preservation of adherence under refinement is addressed.

5.1 Specifying policies

A policy is defined as a set of rules governing the choices in the behavior of a system [37]. 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" [38]. This means that the potential behavior of the system generally spans wider than that which 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. In our approach, each rule is classified as either a *permission*, an *obligation*, or a *prohibition*. This classification is based on standard deontic logic [26], and several of the existing approaches to policy specification have language constructs of such a deontic type [1,17,37,40,43]. This categorization is furthermore used in the ISO/IEC standardized reference model for open, distributed processing [14].

In [39], we evaluate UML sequence diagrams as a notation for policy specification. It is argued that although the notation is to a large extent sufficiently expressive, it is not optimal for policy specification. The reason for this lies heavily in the fact that the UML has no constructs customized for expressing the deontic modalities. In this section, we extend the sequence diagram syntax with constructs suitable for policy specification, and we define their semantics.

In our notation, a policy rule is specified as a sequence diagram that consists of two parts; a triggering scenario and a deontic expression. The notation is exemplified with the permission rule *read* in Fig. 17. The diagram captures the rule, stating that *by the event of a valid login, the user is permitted to retrieve documents from the server, provided the user is a registered user.*

The triggering scenario is captured with the keyword trigger, and the scenario must be fulfilled for the rule to apply. This is exemplified by the diagram *userRegistered* followed by the message *login(id)* in Fig. 17. Observe that in *userRegistered*, there are two alternative ways for a user to be registered. Either the user registers on personal initiative by sending a message with a chosen user id, or the server creates an account and sends the user id to the user. With respect to the rule in question, the two alternatives are considered equivalent.

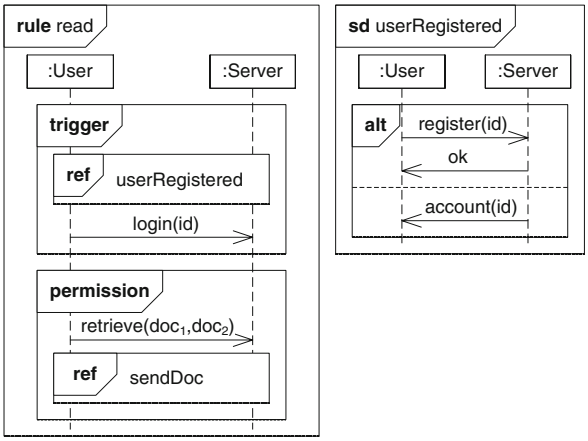


Fig. 17 Permission rule

The specification of a triggering scenario can be understood as a variant of a triggering event that is often used in policy specification [37]. By specifying the policy trigger as an interaction, the circumstances under which a rule applies can be expressed both as a simple message or as a more complex scenario that must be fulfilled.

Notice that we allow the trigger to be omitted in the specification of a policy rule. In that case, the rule applies under all circumstances and is referred to as a standing rule.

The behavior that is constrained by the rule is annotated with one of the keywords permission, obligation, or prohibition, where the keyword indicates the modality of the rule. In the example, the interaction describing document retrieval from the server by the user is specified as a permission. Two alternative ways of sending the document from the server to the user are specified as depicted in the referred diagram *sendDoc* of Fig. 7. Since the rule in Fig. 17 is a permission, the behavior specified in the body is defined as being permitted. By definition of a policy, a policy specification is given as a set of rules, each rule specified in the form shown in Fig. 17.

The proposed extension of the UML 2.1 sequence diagram notation to capture policies is modest and conservative, so people who are familiar with the UML should be able to understand and use the extended notation. Furthermore, all the constructs that are available in the UML for specification of sequence diagrams can be freely used in the specification of a policy rule.

In this paper, we do not consider refinement of policy specifications, only refinement of system specifications expressed in STAIRS. We assume that the policy specification is the final, complete specifications to be enforced. In that case, all behavior that is not explicitly specified as positive is negative, i.e., there are no inconclusive traces. Since $(p \cup n) = \mathcal{T}$ in

the cases of complete specifications, it suffices to refer only to the set of positive traces p in the semantics of the policy specifications.

The semantics of a specification of a policy rule is captured by a tuple (dm, A, B), where $dm \in \{pe, ob, pr\}$ denotes the deontic modality, A denotes the traces representing the triggering scenario, and B denotes the traces representing the behavior. For the special case of a standing policy rule, i.e., a rule in which the trigger is omitted and that applies under all circumstances, the semantics A of the triggering scenario is the set of all traces $\mathcal{T}$. Since a policy specification is a set of policy rule specifications, the semantics of a policy specification is given by a set $P = \{r_1, \ldots, r_m\}$, where each r_i is a tuple capturing the semantics of a policy rule specification.

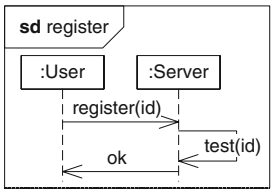
5.2 Policy adherence

In order to fully understand the meaning of a policy specification, it is crucial to understand what it means that a given system specification S adheres to a policy specification P. In the following, we define this adherence relation, denoted $P \rightarrow_a S$. We assume a STAIRS system specification with sequence diagrams and let $[\![S]\!]$ denote the STAIRS semantics of the specification, i.e., $[\![S]\!]$ is a set of interaction obligations.

If a policy rule specification is triggered at some point in a system run, the rule imposes a constraint on the continuation of the execution. An obligation requires that for all continuations, the behavior specified by the rule must be fulfilled, whereas a prohibition requires that none of the continuations may fulfill the behavior. Hence, adherence to obligations and prohibitions are trace properties since they can be falsified on a single trace. A permission, on the other hand, requires the existence of a continuation that fulfills the behavior. In order to falsify adherence to a permission, all continuations must be checked, so permissions represent trace-set properties. As we shall see, these trace properties and trace-set properties are generalized to properties of interaction obligations and sets of interaction obligations, respectively, in STAIRS system specifications.

Generally, a policy specification refers only to the system behavior of relevance for the purpose of the policy, which means that there may be scenarios described in the system specification that are not mentioned in the policy specification. For the same reason, scenarios of the system specification may be only partially described in the policy specification. Consider, for example, the diagram *userRegistered* of the trigger in Fig. 17. For a system specification to fulfill this scenario, there must exist a scenario in the system specification in which *userRegistered* is a subscenario. Assume that the diagram in Fig. 18 is the system specification of the registration of a new user. The user signs up with an id, typically a chosen username and password, after which the server tests the id, e.g., to ensure that the username is unique and that the password conforms to some rules. Since the trace of events representing *register* contains the trace of events representing one of the alternatives specified in *userRegistered*, *userRegistered* is a subscenario of *register*, i.e., *register* represents a fulfillment of *userRegistered*.

Fig. 18 User registration



The fulfillment of one scenario by another is captured by the sub-trace relation $\lhd$. The expression $t_1 \lhd t_2$ evaluates to true iff t_1 is contained in t_2. We say that t_1 is a sub-trace of t_2 and that t_2 is a super-trace of t_1. We have, for example, that

$$\langle !r, ?r, !o, ?o \rangle \lhd \langle !r, ?r, !c, ?c, !o, ?o \rangle$$

where the sub-trace is an element of the positive traces of *userRegistered*, and the super-trace is an element of the positive traces of *register*.[4]

We generalize $\lhd$ to handle trace sets in the first argument. For a trace t to fulfill a scenario represented by a trace set T, there must exist a trace $t' \in T$ such that $t' \lhd t$, denoted $T \lhd t$. Formally,

$$T \lhd t \stackrel{\text{def}}{=} \exists t' \in T : t' \lhd t$$

The negated relation $\not\lhd$ is defined by the following:

$$t_1 \not\lhd t_2 \stackrel{\text{def}}{=} \neg(t_1 \lhd t_2)$$
$$T \not\lhd t \stackrel{\text{def}}{=} \neg\exists t' \in T : t' \lhd t$$

For a policy rule (dm, A, B), we may now define what it means that an interaction obligation triggers the rule. Intuitively, an interaction obligation (p, n) triggers the given rule if some of the implementable traces $p \backslash n$ fulfill the triggering scenario A. This is formally captured by the following definition.

Definition 6 The rule (dm, A, B) is triggered by the interaction obligation (p, n) iff

$$\exists t \in (p \backslash n) : A \lhd t$$

Notice that since the semantics of the triggering scenario is $A = \mathcal{T}$ for standing rules, these rules are trivially triggered by all interaction obligations with a non-empty set of implementable traces.

[4] For a formal definition of the sub-trace relation $\lhd$, see Appendix A.3.

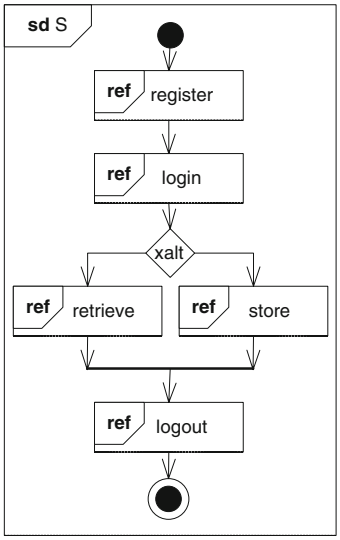


Fig. 19 System specification

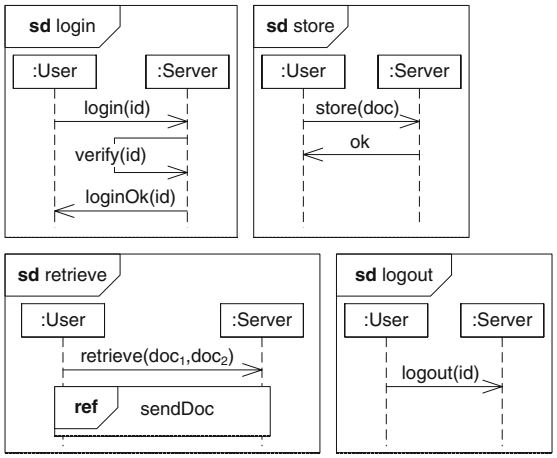


Fig. 20 System interactions

Example 6 The triggering of a rule is exemplified by the interaction overview diagram in Fig. 19 depicting a system specification. Semantically, the system specification S defines two interaction obligations:

$$\{o_1\} = [\![\ register\ ]\!] \gtrsim [\![\ login\ ]\!] \gtrsim [\![\ retrieve\ ]\!] \gtrsim [\![\ logout\ ]\!]$$

$$\{o_2\} = [\![\ register\ ]\!] \gtrsim [\![\ login\ ]\!] \gtrsim [\![\ store\ ]\!] \gtrsim [\![\ logout\ ]\!]$$

As explained above, *register* fulfills *userRegistered* of the rule *read* in Fig. 17. It is furthermore easy to see that *login* of Fig. 20 fulfills the remainder of the triggering scenario, so the composition $[\![\ register\ ]\!] \gtrsim [\![\ login\ ]\!]$ fulfills the triggering scenario of the permission rule. Hence, both o_1 and o_2 trigger the permission rule.

If a system specification S adheres to a permission rule (pe, A, B), whenever the rule is triggered by an interaction obligation $(p, n) \in [\![\ S\ ]\!]$, there must exist an interaction obligation $(p', n') \in [\![\ S\ ]\!]$ that triggers the same rule and also fulfills the permitted behavior. On the other hand, if S adheres to an obligation rule (ob, A, B), and the rule is triggered by an interaction obligation $(p, n) \in [\![\ S\ ]\!]$, (p, n) must fulfill the behavior. Finally, if S adheres to a prohibition rule (pr, A, B), and the rule is triggered by an interaction obligation $(p, n) \in [\![\ S\ ]\!]$, (p, n) must not fulfill the behavior.

As an example, consider again the permission rule *read* in Fig. 17 and the system specification given by the interaction overview diagram in Fig. 19 that triggers the rule. It is easy to see that interaction obligation o_1 fulfills the permitted behavior by the diagram *retrieve* in Fig. 20. As before, fulfillment of one scenario by another is defined by the sub-trace relation.

Formally, policy adherence is defined as follows.

Definition 7 Adherence to policy rule of system specification S.

$$\begin{aligned}
&-(pe, A, B) \rightarrow_a S \stackrel{\text{def}}{=} \\
&\quad (\exists (p_i, n_i) \in [\![\ S\ ]\!] : \exists t \in (p_i \backslash n_i) : A \lhd t) \Rightarrow \\
&\quad \exists (p_j, n_j) \in [\![\ S\ ]\!] : \forall t' \in (p_j \backslash n_j) : (A \gtrsim B) \lhd t' \\
&-(ob, A, B) \rightarrow_a S \stackrel{\text{def}}{=} \\
&\quad \forall (p, n) \in [\![\ S\ ]\!] : \forall t \in (p \backslash n) : A \lhd t \Rightarrow (A \gtrsim B) \lhd t \\
&-(pr, A, B) \rightarrow_a S \stackrel{\text{def}}{=} \\
&\quad \forall (p, n) \in [\![\ S\ ]\!] : \forall t \in (p \backslash n) : A \lhd t \Rightarrow (A \gtrsim B) \not\lhd t
\end{aligned}$$

With these definitions of adherence to policy rules, we define adherence to a policy specification as follows.

Definition 8 Adherence to policy specification P of system specification S.

$$P \rightarrow_a S \stackrel{\text{def}}{=} \forall r \in P : r \rightarrow_a S$$

5.3 Preservation of policy adherence under refinement

The property of adherence preserving refinement is desirable, since it implies that if adherence is verified at an abstract level, adherence is guaranteed for all refinements and need not be verified again.

In order to formally prove that policy adherence is preserved under refinement, we must prove preservation of adherence under refinement for each kind of policy rule.

Theorem 4 *Preservation of policy adherence under refinement.*

(a) $(pe, A, B) \rightarrow_a S_1 \wedge S_1 \rightsquigarrow S_2 \Rightarrow (pe, A, B) \rightarrow_a S_2$
(b) $(ob, A, B) \rightarrow_a S_1 \wedge S_1 \rightsquigarrow S_2 \Rightarrow (ob, A, B) \rightarrow_a S_2$
(c) $(pr, A, B) \rightarrow_a S_1 \wedge S_1 \rightsquigarrow S_2 \Rightarrow (pr, A, B) \rightarrow_a S_2$

Proof To prove (a), assume $(pe, A, B) \rightarrow_a S_1$ and $S_1 \rightsquigarrow S_2$. Prove $(pe, A, B) \rightarrow_a S_2$.

Let (p_i, n_i) be any element of $[\![S_2]\!]$ such that there exists $t \in (p_i \backslash n_i)$ and $A \lhd t$. The permission rule is then triggered by (p_i, n_i), and we need to prove that there is an element $(p_j, n_j) \in [\![S_2]\!]$ that triggers the same rule and that also fulfills the permitted behavior.

By Definition 3, $\exists (p'_i, n'_i) \in [\![S_1]\!] : (p'_i, n'_i) \rightsquigarrow (p_i, n_i)$. By Definition 2, $p_i \subseteq p'_i$ and $n'_i \subseteq n_i$. Hence, $t \in (p'_i \backslash n'_i)$, and (p'_i, n'_i) triggers the permission rule. Since $(pe, A, B) \rightarrow_a S_1$ by assumption, $\exists (p'_j, n'_j) \in [\![S_1]\!] : \forall t' \in (p'_j \backslash n'_j) : (A \gtrsim B) \lhd t'$.

By Definition 3, $\exists (p_j, n_j) \in [\![S_2]\!] : (p'_j, n'_j) \rightsquigarrow (p_j, n_j)$. By Definition 2, $p_j \subseteq p'_j$ and $n'_j \subseteq n_j$, i.e., $(p_j \backslash n_j) \subseteq (p'_j \backslash n'_j)$. Hence, $\forall t'' \in (p_j \backslash n_j) : (A \gtrsim B) \lhd t''$ as desired.

To prove (b), assume $(ob, A, B) \rightarrow_a S_1$ and $S_1 \rightsquigarrow S_2$. Prove $(ob, A, B) \rightarrow_a S_2$.

Let (p, n) be any element of $[\![S_2]\!]$ such that there exists $t \in (p \backslash n)$ and $A \lhd t$. The obligation rule is then triggered by (p, n), and we need to prove that $(A \gtrsim B) \lhd t$.

By Definition 3, $\exists (p', n') \in [\![S_1]\!] : (p', n') \rightsquigarrow (p, n)$. By Definition 2, $p \subseteq p'$ and $n' \subseteq n$. Hence, $t \in (p' \backslash n')$, and (p', n') triggers the obligation rule. By assumption of adherence, $(A \gtrsim B) \lhd t$ as desired.

To prove (c), assume $(pr, A, B) \rightarrow_a S_1$ and $S_1 \rightsquigarrow S_2$. Prove $(pr, A, B) \rightarrow_a S_2$.

Let (p, n) be any element of $[\![S_2]\!]$ such that there exists $t \in (p \backslash n)$ and $A \lhd t$. The prohibition rule is then triggered by (p, n), and we need to prove that $(A \gtrsim B) \not\lhd t$.

By Definition 3, $\exists (p', n') \in [\![S_1]\!] : (p', n') \rightsquigarrow (p, n)$. By Definition 2, $p \subseteq p'$ and $n' \subseteq n$. Hence, $t \in (p' \backslash n')$, and (p', n') triggers the prohibition rule. By assumption of adherence, $(A \gtrsim B) \not\lhd t$ as desired. □

Example 7 To illustrate the usefulness of Theorem 4, assume a specification S' that is identical to the specification of S depicted in Fig. 19, except that the diagram *sendDoc* of Fig. 7 is replaced with the diagram *sendDoc2* of Fig. 16.

As explained in Example 5, since *sendDoc* $\rightsquigarrow$ *sendDoc2*, then also $S \rightsquigarrow S'$. In this step of refinement, underspecification is reduced by moving from S to S', but since $P \rightarrow_a S$ and $S \rightsquigarrow S'$, we have $P \rightarrow_a S'$ by Theorem 4.

The property of refinement that every interaction obligation (p, n) at the abstract level is represented at the refined level by an interaction obligation (p', n') such that $(p, n) \rightsquigarrow (p', n')$ is crucial for the validity of Theorem 4; if an interaction obligation (p, n) ensures adherence to a permission rule at the abstract level, then all refinements of (p, n) will ensure the same adherence. For obligations and prohibitions, adherence is preserved since no interaction obligations can be added at the refined level that are not represented at the abstract level, thus ensuring that a policy breach is not introduced.

6 Related work

It is well known that trace-properties such as safety and liveness are preserved under the classical notion of refinement defined as reverse set inclusion. Trace-set properties, however, are generally not preserved under this notion of refinement, which has given rise to the so called refinement paradox [16]. As explained above, the STAIRS approach avoids this paradox by offering a semantics that is sufficiently rich to distinguish underspecification from inherent non-determinism. In the following, we relate STAIRS to other approaches to system specification and development with interactions.

The family of approaches we consider are those that have emerged from the ITU recommendation message sequence chart (MSC) [13], a notation most of which has been adopted and extended by the UML 2.1 sequence diagram notation. ITU has also provided a formal operational semantics of MSCs [12] based on work by Mauw and Reniers [23,24].

An MSC describes a scenario by showing how components or instances interact in a system by the exchange of messages. Messages in MSCs, are as for UML sequence diagrams, ordered by weak sequencing, which yields a partial ordering of events. Several operators are defined for the composition of MSCs, such as weak sequencing, parallel composition, and alternative executions. The explicit specification of scenarios as negative or forbidden is, however, not supported.

More important in our context is that the distinction between underspecification and inherent non-determinism is beyond the standard MSC language and its semantics [12, 13]. Furthermore, there is no notion of refinement defined for MSCs. This means that there is no support in MSCs for the capture of trace-set properties and their preservation under a stepwise development process.

In [18], Katoen and Lambert define a compositional denotational semantics for MSCs based on the notion of partial-order multi-sets. This denotational semantics complements the standardized operational semantics, and does not aim to introduce new expressiveness. Hence, there is no distinction between underspecification and inherent non-determinism. Refinement is also not addressed in [18].

In the work by Krüger [19], a variant of MSCs is given a formal semantics and provided a formal notion of refinement. Four different interpretations of MSCs are proposed; namely, existential, universal, exact, and negative. The existential interpretation requires the fulfillment of the MSC in question by at least one system execution; the universal interpretation requires the fulfillment of the MSC in all

executions; the exact interpretation is a strengthening of the universal interpretation by explicitly prohibiting behaviors other than the ones specified by the MSC in question; the negative interpretation requires that no execution is allowed to fulfill the MSC. Three notions of refinement are defined; namely, property, message and, structural. Property refinement corresponds to the classical notion of refinement as reverse set inclusion, i.e., the removal of underspecification by reducing the possible behavior of the system; message refinement is to substitute an interaction sequence for a specific message; structural refinement means to decompose an instance, or lifeline, with a set of instances.

The interesting interpretation in our context is the existential scenario as it may capture trace-set properties. As shown in [19], a system that fulfills an MSC specification under the universal, exact, or negative interpretation also fulfills specifications that are property refinements of the initial specification. This is, however, not the case for the existential interpretation, which means that trace-set properties are not preserved under refinement.

Broy [3] proposes an interpretation of MSCs that differs significantly from the standardized approach. Rather than understanding an MSC as a description of the entire system as a whole, MSCs are in his work perceived as descriptions of properties of each system component (instance or lifeline in MSC and UML terms, respectively). In the semantic model, MSCs are described in terms of logical propositions that characterize stream-processing functions. A stream is basically a sequence of events, and a component is described by a function from input streams to output streams. If an MSC is not specified for a given input stream, the behavior is inconclusive. This interpretation deviates from the standard interpretation of both MSCs and UML sequence diagrams. The intention in [3], however, is merely to treat MSCs as a general idea of a system description in which components interact in a concurrent and distributed setting.

The paper discusses how MSCs should be utilized in the development process and is supported by a formal notion of refinement. Refinement is defined in the style of Focus [4] on the basis of the input/output function of a single component and with respect to a given input stream. Basically, a component is a refinement of another component if for a given input stream, the set of output streams generated by the former is a subset of the set of output streams generated by the latter. Hence, the concrete component fulfills all the requirements that are fulfilled by the abstract component. The approach in [3] does not address trace-set properties.

Uchitel et al. [42] present a technique for generating Modal Transition Systems (MTSs) from a specification given as a combination of system properties and scenario specifications. System properties, such as safety or liveness, are universal as they impose requirements on all system runs, and are in [42] specified in Fluent Linear Temporal Logic. Scenario specifications are existential as they provide examples of intended system behavior, and are specified in the form of MSCs. An MTS is a behavior model with the expressiveness to distinguish between required, possible, and proscribed behavior. The method for generating MTSs from properties and scenarios ensures that all system behaviors satisfy the properties while the system may potentially fulfill all the scenarios. Furthermore, both composition and refinement of behavior models preserve the original properties and scenarios. With this approach, MSCs can be utilized to capture trace-set properties and preserve these under refinement. However, as opposed to e.g., STAIRS, the approach in [42] is not a pure interaction based development process, and the use of MSCs is restricted to the existential interpretation. Moreover, reasoning about system behavior and refinement is based on the MTS behavior models and cannot be easily conducted at the syntactic level.

Grosu and Smolka [6] address the problem of equipping UML 2.0 sequence diagrams with a formal semantics that allows compositional refinement. In their approach, positive and negative interactions are interpreted as liveness and safety properties, respectively. This may obviously be a useful interpretation of sequence diagrams, but is much stronger than the traditional interpretation in which sequence diagrams are used to illustrate example runs. Refinement is defined as the traditional reverse trace set inclusion and is compositional. There is, however, no support for the preservation of trace-set properties under this notion of refinement.

Störrle [41] defines a trace semantics for UML 2.0 interactions where a sequence diagram is captured by a pair of a set of positive traces and a set of negative traces. The approach provides no support for distinguishing between underspecification and inherent non-determinism. Rather, positive traces are interpreted as necessary, i.e., must be possible in an implementation, whereas negative traces are interpreted as forbidden, i.e., must not be possible in an implementation. A notion of refinement is introduced in which previously inconclusive traces may be redefined as positive or negative. With this approach, there is no guarantee that adherence to trace-set properties is preserved under refinement.

Live sequence charts (LSCs) [5,8] are an extension of MSCs that particularly address the issue of expressing liveness properties. LSCs support the specification of two types of diagrams; namely, existential and universal. An existential diagram describes an example scenario that must be satisfied by at least one system run, whereas a universal diagram describes a scenario that must be satisfied by all system runs. Universal charts can furthermore be specified as conditional scenarios by the specification of a prechart that, if successfully executed by a system run, requires the fulfillment of the scenario described in the chart body.

The universal/existential distinction is a distinction between mandatory and provisional behavior, respectively.

Such a distinction is also made between elements of a single LSC by characterizing these as hot or cold, where a hot element is mandatory and a cold element is provisional. LSCs furthermore have the expressiveness to specify forbidden scenarios by placing a hot condition that evaluates to false immediately after the relevant scenario. The condition construct of LSCs and MSCs corresponds to UML 2.1 state invariants and is a condition that must evaluate to true when a given state is active. If and when the system fulfills the given scenario, it is then required to satisfy the false condition, which is impossible.

LSCs seem to have the expressiveness to ensure adherence to trace-set properties by the use of existential diagrams; obviously, falsification of system satisfaction of requirements expressed by an existential diagram cannot be done on a single trace. However, a system development in LSCs is intended to undergo a shift from an existential view in the initial phases to a universal view in later stages as knowledge of the system evolves. Such a development process with LSCs will generally not preserve trace-set properties. Moving from an existential view to a universal view can be understood as a form of refinement, but LSCs are not supported by a formal notion of refinement.

The semantics of LSCs is that of partial ordering of events defined for MSCs [13]. Importantly, however, the semantics of LSCs defines the beginning of precharts and the beginning of main charts as synchronization points, meaning that all lifelines enter the prechart simultaneously and that the main chart is entered only after all lifelines have completed their respective activities in the prechart. This yields a strong sequencing between the prechart and main chart, a property of LSCs that may seem unfortunate in some situations, specifically for distributed systems where system entities behave locally and interact with other entities asynchronously. A further disadvantage with respect to system development and analysis with LSCs is the lack of composition constructs for combining LSCs. A specification is instead given as a set of LSCs, and a system satisfies a specification by satisfying each of its individual LSCs.

Modal sequence diagrams (MSDs) [7] are defined as a UML 2.0 profile. The notation is an extension of the UML sequence diagram notation based on the universal/existential distinction of LSCs. The main motivation for the development of the MSD language are the problematic definitions of the assert and negate constructs of UML sequence diagrams. The authors observe that the UML 2.0 specification is contradictory in the definition of these constructs, and also claim that the UML trace semantics of valid and invalid traces is inadequate for properly supporting an effective use of the constructs.

The semantics for MSDs is basically the same as for LSCs. The main difference is that the LSC prechart construct is left out. Instead, a more general approach is adopted in which cold fragments inside universal diagrams serve the purpose of a prechart. A cold fragment is not required to be satisfied by all runs, but if it is satisfied, it requires the satisfaction of the subsequent hot fragment.

With respect to capturing trace-set properties and preserving these under refinement, the situation is the same as for LSCs. Existential diagrams may be utilized to ensure adherence to trace-set properties. However, there is no formal notion of refinement guaranteeing preservation of adherence. The existential view is understood as a form of underspecification suitable for the early phases. A gradual shift from existential to universal diagrams reduces the underspecification and restricts the set of system models that satisfy the specification. It is in [7] indicated that existential diagrams should be kept in the specification as the development process evolves and that universal diagrams are gradually added. With such an approach, trace-set properties might be preserved.

Triggered message sequence charts (TMSCs) [36] is an approach in the family of MSCs that is related to STAIRS in several ways. The development of TMSCs is motivated by the fact that MSCs do not have the expressiveness to define conditional scenarios, i.e., that one interaction (the triggering scenario) requires the execution of another (the action scenario), that MSCs do not formally define a notion of refinement, and that MSCs lack structuring mechanisms that properly define ways of grouping scenarios together.

The triggering scenarios of TMSCs are closely related to the precharts of LSCs. An important semantic difference, however, is that whereas LSCs are synchronized at the beginning of precharts and main charts, TMSCs are based on weak sequencing in the spirit of MSCs.

TMSCs have composition operators for sequential composition, recursion (similar to loop), and parallel composition. The most interesting composition operators in the context of this paper, however, are delayed choice and internal choice. Both operators take a set of diagrams as operands and define a choice between these. Internal choice is an operator in the spirit of the STAIRS alt operator and defines underspecification. An implementation that can execute only one of the operands is a correct implementation. Delayed choice, on the other hand, is an operator somewhat related to the STAIRS xalt operator; a correct implementation must provide all the choices defined by the specification. However, as opposed to inherent non-determinism captured by the xalt operator, a delayed choice is not made until a given execution forces it to be made.

It is observed in [36] that the simple trace set semantics of MSCs does not have the expressiveness to distinguish between optional and required behavior, which means that a heterogeneous mix of these in the specification is not supported. Such a mix is supported in STAIRS; syntactically by the alt operator for underspecification and the xalt

operator for inherent non-determinism, semantically by the set of interaction obligations.

The semantics of TMSCs is defined in terms of so-called acceptance trees. Acceptance trees record the traces that are defined by a specification, but also distinguish between required and optional behavior. Importantly, the semantics of TMSCs supports a notion of refinement that preserves the required behavior and gives freedom with respect to optional behavior.

The authors stress that refinement should preserve properties such as safety and liveness. These properties are trace properties, and the issue of capturing trace-set properties and preserving these under refinement is not discussed. However, by the semantics of the delayed choice operator and the fact that this type of choice is preserved under refinement, TMSCs seem to have the expressiveness to capture and preserve trace-set properties.

Like the refinement relations of STAIRS, refinement of TMSCs is compositional, i.e., composition of specifications is monotonic with respect to refinement.

An important difference between STAIRS and TMSCs is that the latter do not support the specification of negative behavior. This also means that with TMSCs there is no notion of inconclusive behavior; a system specification defines a set of valid traces, and all other traces are invalid. The refinement relation is based on set inclusion, so refinement basically means that behavior that was previously defined as positive is redefined as negative. TMSCs allow the specification of partial scenarios in the sense that interactions may continue beyond what is specified in a given TMSC. This means that diagrams may be incomplete and that the traces are extensible through refinement. Semantically, partial scenarios mean that what is left unspecified by a scenario imposes no constraints on behavior, i.e., that all behavior beyond what is specified is legal.

Distinguishing between positive, negative, and inconclusive behavior as in STAIRS and UML 2.1 provides useful expressiveness and is, in our opinion, a more intuitive interpretation of partial or incomplete scenarios than what is provided for TMSCs. The lack of a precise definition in the UML 2.1 specification of what is the meaning of a valid trace motivated the work on MSDs. For both LSCs and MSDs, the precise meaning of a specification is given by defining what it means that a system model satisfies a specification. In STAIRS, this is correspondingly defined by formalizing the relation of compliance [34]. However, in the process of system development, it is as important to understand the meaning of the distinction between valid and invalid behavior when moving from one specification to a more concrete specification by refinement. This is precisely defined in STAIRS by the formal notion of refinement.

We conclude this section by relating our customized UML sequence diagram notation for policy specification presented in Sect. 5 to other work. The new features of this notation with respect to both UML sequence diagrams and MSCs are basically the triggering construct and the deontic modalities. Triggering scenarios for MSCs are supported in the work by Krüger, and also for LSCs and TMSCs. In the former two, the main motivation for specifying triggering scenarios is that they facilitate the capturing of liveness properties. In contrast to our approach, the composition of the triggering scenario and the triggered scenario is that of strong rather than weak sequencing in both Krüger's work and for LSCs. Furthermore, Krüger proposes no constructs for explicitly characterizing scenarios as obligated, permitted, or prohibited, but rather operates with different interpretations of MSC specifications. In LSCs, scenarios are existential, universal, or forbidden, which corresponds closely to the permission, obligation, and prohibition modalities, respectively. There is, however, no explicit construct for specifying the latter. Instead, a hot false condition must be placed immediately after the relevant scenario. As for TMSCs, there is no support for distinguishing between obligated, permitted, and negative behavior; all traces are characterized as either valid or invalid.

7 Conclusions

The expressiveness of STAIRS allowing inherent non-determinism to be distinguished from underspecification facilitates the specification of trace-set properties and the preservation of such properties under refinement. This paper demonstrates the potential of STAIRS in this respect within the areas of information flow security and policy specification.

STAIRS formalizes the trace semantics that is only informally described in the UML standard. STAIRS furthermore offers a rich set of refinement relations [34] of which only one (restricted limited) has been addressed in this paper. The refinement relations are supported by theoretical results indicating their suitability to formalize modular, incremental system development.

STAIRS also offers an operational semantics that has been proved to be sound and complete with respect to the denotational semantics [21]. The Escalator tool [20] makes use of this semantics to offer automatic verification and testing of refinements.

Furthermore, there are extensions of STAIRS to deal with (hard) real-time [10], as well as probability [30] and soft real-time [29]

Acknowledgments The research on which this paper reports has been partly funded by the Research Council of Norway through the projects ENFORCE (164382/V30) and DIGIT (180052/S10), and partly by the European Commission through the S3MS project (Contract no. 27004) under the IST Sixth Framework Programme. We are grateful to Mass

Soldal Lund, Atle Refsdal, and Ragnhild Kobro Runde for valuable comments and suggestions.

Appendix A: Formal definitions

A.1 Operations on sequences

The following operations on sequences are adapted from [4].

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 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:

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

The prefix relation on sequences,

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

is formally defined as follows:

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

The complementary relation is defined by the following:

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

The truncation operator

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

is used to truncate a sequence at a given length.

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

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

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

is used to filter away elements. $A ⓢ s$ 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.

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

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

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

The filtering operator ⓣ is defined for pairs of sequences:

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

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

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

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 ⓣ is now defined by the following conditional equations.

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

A.2 Weak sequencing

For traces t_1 and t_2, which are sequences of events, weak sequencing is defined by the function

$$\_ \gtrsim \_ \in \mathcal{T} \times \mathcal{T} \to \mathbb{P}(\mathcal{T})$$

Formally, weak sequencing is defined as follows.

$$t_1 \gtrsim t_2 \stackrel{\text{def}}{=} \{t \in \mathcal{T} \mid \forall l \in \mathcal{L} : e.l ⓢ t = e.l ⓢ t_1 \frown e.l ⓢ t_2\}$$

where $e.l$ denotes the set of events that may take place on the lifeline l. Formally,

$$\begin{aligned} e.l \stackrel{\text{def}}{=} \{ & (k, (co, tr, re)) \in \mathcal{E} \mid \\ & (k = ! \wedge tr = l) \ \vee (k = ? \wedge re = l)\} \end{aligned}$$

A.3 Sub-trace relation

The sub-trace relation $\lhd$ on sequences,

$$\_ \lhd \_ \in E^{\omega} \times E^{\omega} \to \mathbb{B}\text{ool}$$

is formally defined as follows.

$$s_1 \lhd s_2 \stackrel{\text{def}}{=} \exists s \in \{1, 2\}^{\infty} : \pi_2((\{1\} \times E) ⓣ (s, s_2)) = s_1$$

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 s_2 are filtered away.

References

1. Aagedal, J.O., Milošević, Z.: ODP enterprise language: UML perspective. In: Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99), pp. 60–71. IEEE Computer Society (1999)
2. Alpern, B., Schneider, F.B.: Defining liveness. Inform. Process. Lett. **21**(4), 181–185 (1985)
3. Broy, M.: A semantic and methodological essence of message sequence charts. Sci. Computer Program. **54**(2–3), 213–256 (2005)
4. Broy, M., Stølen, K.: Specification and development of interactive systems. FOCUS on Streams, Interface, and Refinement. Springer, Berlin (2001)
5. Damm, W., Harel, D.: LSCs: Breathing life into message sequence charts. Formal Methods Syst. Des. **19**(1), 45–80 (2001)
6. Grosu, R., Smolka, S.A.: Safety-liveness semantics for UML 2.0 sequence diagrams. In: Proceedings of Applications of Concurrency to System Design (ACSD'05), pp. 6–14. IEEE Computer Society (2005)
7. Harel, D., Maoz, S.: Assert and negate revisited: modal semantics for UML sequence diagrams. Softw. Syst. Model. **7**(2), 237–252 (2008)
8. Harel, D., Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer, Berlin (2003)
9. Haugen, O., Husa, K.E., Runde, R.K., Stølen, K.: STAIRS towards formal design with sequence diagrams. Softw. Syst. Model. **4**, 355–367 (2005)
10. Haugen, O., Husa, K.E., Runde, R.K., Stølen, K.: Why timed sequence diagrams require three-event semantics. In: Scenarios: models, transformations and tools, vol. 3466 of LNCS, pp. 1–25. Springer, Berlin (2005)
11. Hoare, C.A.R.: Communicating Sequential Processes. Series in computer science. Prentice-Hall, Englewood Cliffs, NJ (1985)
12. International Telecommunication Union. Recommendation Z.120 Annex B—Semantics of Message Sequence Chart (MSC) (1998)
13. International Telecommunication Union. Recommendation Z.120—Message Sequence Chart (MSC) (2004)
14. ISO/IEC. FCD 15414, Information Technology—Open Distributed Processing—Reference Model—Enterprise Viewpoint (2000)
15. Jacob, J.: On the derivation of secure components. In: Proceedings of the IEEE Symposium on Security and Privacy (SP'89), pp. 242–247. IEEE Computer Society (1989)
16. Jürjens, J.: Secrecy-preserving refinement. In: Proceedings of Formal Methods Europe (FME'01), vol. 2021 of LNCS, pp. 135–152. Springer, Berlin (2001)
17. Kagal, L., Finin, T., Joshi, A.: A policy language for a pervasive computing environment. In: Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY'03), pp. 63–74. IEEE Computer Society (2003)
18. Katoen, J.-P., Lambert, L.: Pomsets for message sequence charts. In: Formale Beschreibungstechniken für verteilte Systeme, pp. 197–208. Shaker, Germany (1998)
19. Krüger, I.H.: Distributed System Design with Message Sequence Charts. PhD thesis, Institut für Informatik, Ludwig-Maximilians-Universität München (2000)
20. Lund, M.S.: Operational analysis of sequence diagram specifications. PhD thesis, University of Oslo (2008)
21. Lund, M.S., Stølen, K.: A fully general operational semantics for UML 2.0 sequence diagrams with potential and mandatory choice. In: Proceedings of the 14th International Symposium on Formal Methods (FM'06), number 4085 in LNCS, pp. 380–395. Springer, Berlin (2006)
22. Mantel, H.: Possibilistic definitions of security—an assembly kit. In: Proceedings of IEEE Compuer Security Foundations Workshop (CSFW'00), pp. 185–199. IEEE Computer Society (2000)
23. Mauw, S., Reniers, M.A.: High-level message sequence charts. In: Proceedings of the 8th SDL Forum, pp. 291–306. Elsevier, Amsterdam (1997)
24. Mauw, S., Reniers, M.A.: Operational semantics for MSC'96. Computer Netw. ISDN Syst. **31**(17), 1785–1799 (1999)
25. McLean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, pp. 79–93. IEEE Computer Society (1994)
26. McNamara, P.: Deontic logic. In: Gabbay, D.M., Woods, J. (eds) Logic and the Modalities in the Twentieth Century, vol. 7 of Handbook of the History of Logic, pp. 197–288. Elsevier, Amsterdam (2006)
27. Object Management Group. Unified Modeling Language: Superstructure, version 2.1.1 (2007)
28. O'Halloran, C.: A calculus of information flow. In: Proceedings of European Symposium on Research in Computer Security (ESORICS'90), pp. 147–159. AFCET (1990)
29. Refsdal, A., Husa, K.E., Stølen, K.: Specification and refinement of soft real-time requirements using sequence diagrams. In: Proceedings of the 3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05), vol. 3829 of LNCS, pp. 32–48. Springer, Berlin (2005)
30. Refsdal, A., Runde, R.K., Stølen, K.: Underspecification, inherent nondeterminism and probability in sequence diagrams. In: Proceedings of the 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'06), vol. 4037 of LNCS, pp, 138–155. Springer, Berlin (2006)
31. Roscoe, A.: CSP and determinism in security modelling. In: Proceedings of IEEE Symposium on Security and Privacy (SP'95), pp. 114–127. IEEE Computer Society (1995)
32. Runde, R.K., Haugen, O., Stølen, K.: How to transform UML neg into a useful construct. In: Proceedings of Norsk Informatikkonferanse, pp. 55–66. Tapir, Trondheim (2005)
33. Runde, R.K., Haugen, O., Stølen, K.: Refining UML interactions with underspecification and nondeterminism. Nordic J. Comput. **12**(2), 157–188 (2005)
34. Runde, R.K., Refsdal, A., Stølen, K.: Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice. Part 1. Underspecification and inherent nondeterminism. Technical Report, vol. 346. Department of Informatics, University of Oslo (2007)
35. Schneider, F.B.: Enforceable security policies. ACM Trans. Inform. Syst. Security **3**(1), 30–50 (2000)
36. Sengupta, B., Cleaveland, R.: Triggered message sequence charts. IEEE Trans. Softw. Eng. **32**(8), 587–607 (2006)
37. Sloman, M.: Policy driven management for distributed systems. Netw. Syst. Manage. **2**(4), 333–360 (1994)
38. Sloman, M., Lupu, E.: Security and management policy specification. IEEE Netw. **16**(2), 10–19 (2002)
39. Solhaug, B., Elgesem, D., Stølen, K.: Specifying policies using UML sequence diagrams – An evaluation based on a case study. In: Proceedings of the 8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07), pp. 19–28. IEEE Computer Society (2007)
40. Steen, M., Derrick, J.: Formalising ODP enterprise policies. In: Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99), pp. 84–93. IEEE Computer Society (1999)
41. Störrle, H.: Trace semantics of interactions in UML 2.0. Technical Report TR 0403, University of Munich (2004)
42. Uchitel, S., Brunet, G., Chechik, M.: Behaviour model synthesis from properties and scenarios. In: Proceedings of the 29th International Conference in Software Engineering (ISCE'07), pp. 34–43. IEEE Computer Society (2007)

43. Wies, R.: Policy definition and classification: Aspects, criteria, and examples. In: Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operation and Management (1994)
44. Zakinthinos, A., Lee, E.S.: A general theory of security properties. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, pp. 94–102. IEEE Computer Society (1997)

Author biographies

Fredrik Seehusen holds a Master's degree in Computer Science from the University of Oslo. He is currently employed as a research scientist at SINTEF Information and Communication Technology, and affiliated with the University of Oslo where he is studying for his PhD. His field of interest includes security and modeling languages.

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

Ketil Stølen is Chief Scientist and Group Leader at SINTEF. Since 1998 he is Professor in computer science at the University of Oslo. He has broad experience from basic research (4 years at Manchester University; 5 years at Munich University of Technology; 10 years at the University of Oslo) as well as applied research (1 year at the Norwegian Defense Research Establishment; 3 years at the OECD Halden Reactor Project; 8 years at SINTEF). Stølen did his PhD "Development of Parallel Programs on Shared Data-structures" at Manchester University on a personal fellowship granted by the Norwegian Research Council for Science and the Humanities. At Munich University of Technology his research focused on the theory of refinement and rules for compositional and modular system development; in particular, together with Manfred Broy, he designed the Focus method as documented in the Focus book published in 2001. At the OECD Halden Reactor Project he was responsible for software development projects involving the use of state-of-the-art CASE-tool technology for object-oriented modeling. He led several research activities concerned with the modeling and dependability-analysis of safety-critical systems. He has broad experience from research projects, nationally as well as internationally, and from the management of research projects. From 1992–1996 he was project-leader under Sonderforschungsbereich 342 "Methodik des Entwurfs verteilter Systeme" at Munich University of Technology. From 2001–2003 he was the technical manager of the EU-project CORAS which had 11 partners and a total budget of more than 5 million EURO. He is currently managing several major Norwegian research projects focusing on issues related to modeling, security, privacy and trust.