Information Flow Property Preserving Transformation of UML Interaction Diagrams

Fredrik Seehusen and Ketil Stølen
SINTEF ICT / University of Oslo
PB 124 Blindern, 0373 Oslo, Norway
{fse, kst}@sintef.no

ABSTRACT

We present an approach for secure information flow property preserving refinement and transformation of UML inspired interaction diagrams. The approach is formally underpinned by trace-semantics. The semantics is sufficiently expressive to distinguish underspecification from explicit nondeterminism. A running example is used to introduce the approach and to demonstrate that it is of practical value.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Design, Languages, Security

Keywords

Model Driven Architecture, Information Flow Security, Refinement, Transformation, UML

1. INTRODUCTION

Security is an important attribute of many software systems. Nevertheless, careful engineering of security into overall design is often neglected. Security features are typically built into an application in an ad-hoc manner or are only integrated during the final phases of system development [24]. It is, however, a common view in the security field that security mechanisms should be taken into account and built into the system during early phases of system development at a higher level of abstraction than the level of implementation.

The Object Management Group (OMG) advocates a framework for system development, Model Driven Architecture (MDA) [27], that aims to raise the level of abstraction of the programming environment by supporting (1) a model-driven development process, (2) a clear separation of abstract, platform independent models (PIMs) and refined, platform dependent models (PSMs), and (3) transformations between PIMs and PSMs.

By integrating security into MDA, security documentation can be specified and analyzed at a high level of abstraction during system development, thereby reducing the need for ad hoc integration of security mechanisms after system implementation. Moreover, the advantage of analyzing PIMs rather than PSMs is that analysis is more feasible at high level of abstractions as opposed to levels closer to implementation which may include too much detail to make analysis practical. Also, it is well known that the earlier an error is discovered, the easier and cheaper it is to fix.

In order to contribute towards a formal foundation of security within the MDA framework, we restrict the notion of security to a property of *secure information flow*. Secure information flow properties in general, provide a way of specifying security requirements 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. There are numerous definitions of information flow properties [25]. The property that we consider is based on a definition given in [21]. Informally, the property holds if an observer communicating with a system, based on its observations and its knowledge of the system specification, cannot infer that another observer has communicated with the same system.

The relationship between information flow security and refinement has been researched for a fairly long time. In 1989 it was shown by Jacob [21] that the derivation of secure systems from security specifications can be practically infeasible. It has later been observed e.g. in [17, 22, 30], that the problem originates in the inability of most specification languages, UML 2.0 [13] included, to distinguish between underspecification (potential nondeterminism) and explicit nondeterminism[1]. An exception in this respect is STAIRS [14, 15, 16], an approach that provides a formal semantics for UML 2.0 inspired interaction diagrams where the distinction of potential and explicit nondeterminism is made. The approach presented in this paper extends the theory of STAIRS wrt. refinement and information flow. It also provides the first steps towards a formal foundation for model-driven security in the setting of UML interactions. In particular, the contributions of this paper are:

[1] Also termed *unpredictability* [22] and *probabilistic nondeterminism* [30].

- A formal definition of a notion of secure information flow in STAIRS. Thereby we provide a foundation for analysis of UML 2.0 interaction diagrams wrt. secure information flow. The only work that we are aware of that deals with information flow security in a proposed semantics of UML is [22], but only a simplified version of UML 1.5 [12] interaction diagrams is considered in this work.
- A formal definition of the refinement [20] that preserves our notion of secure information flow. The most notable recent theoretical works that we are aware of that address refinement and secure information flow are [5, 26]. In [5], conditions for *checking* that a given refinement is information flow preserving are presented, and [26] proposes a way of *modifying* refinement operations such that they remain secure information flow preserving. These approaches are clearly different from ours. We prove that *all* refinements preserve our notion of secure information flow, thus there is no need to check or modify given refinements as in [5, 26].
- A formal definition of a notion of *transformation* that preserves our notion of secure information flow. The notion of refinement is a relation between two specifications and it is in general not subject for automation. In contrast, a transformation takes a specification and delivers another specification as output in the sense of a compiler. There are no related works that we are aware of that address secure information flow with respect to this notion of transformation.
- A generalization of the notion of refinement into a more general notion of refinement modulo transformation. This more general concept may be understood as a kind of data refinement [19].

The paper is structured as follows: In Sect. 2 we use STAIRS to specify a PIM of an example system which is sufficiently realistic to suggest that our approach is of practical value. In Sect. 3 we define an information flow property in terms of the STAIRS semantics, and explain why our PIM is secure with respect to this property. Sect. 4 presents a formalization of refinement in the STAIRS semantics, and explains why the information flow property is preserved under refinement. In Sect. 5 we introduce a notion of transformation which is information flow property preserving. We also integrate this notion of transformation and the classical notion of refinement from STAIRS into a more general concept. In Sect. 6 we show how our PIM can be transformed to PSMs in such a way that the information flow property is preserved. Sect. 7 describes related work. Sect. 8 provides conclusions and suggests directions for future work. The appendix provides formal definitions of auxiliary operators used in this paper.

2. THE PROJECT MANAGEMENT SYSTEM

Consider a large software developing company that aims to develop a distributed system, the project management system (the PM system), in order to centralize all storage of software development projects. Software developers should be able to retrieve projects from a server to their local machines, edit or add files to the project, and upload any changes back to the server. Projects stored on the server should be versioned, and whenever a developer updates a project, only changes in the project with respect to the developer's local copy should be updated.

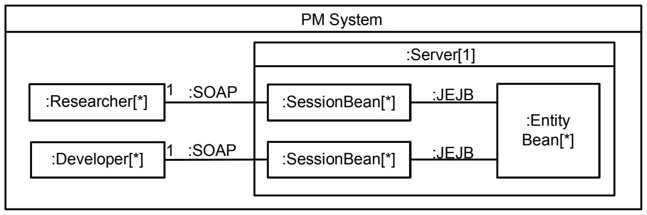

Figure 1: Overview of the PM System

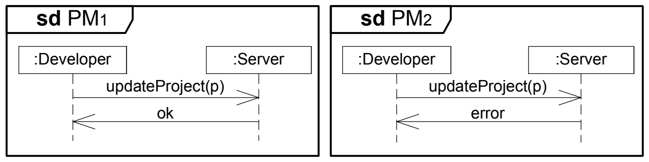

Figure 2: Simple interaction scenario of the PM system

Currently, the company has no unified development method, and developers working on different projects are to a large degree given flexibility in the method they choose to adopt. The company wants to assess the different methods in order to recommend improvements, and possibly to introduce a unified development process. This task is assigned to a group of researches. The researchers are to pick a set of sample projects, and assess each project thoroughly with respect to progress, quality of code etc. For convenience, the PM system should be augmented slightly such that the researchers will be able to retrieve projects from the server over the Internet on a regular basis. This additional functionality is not of high priority, thus it will be implemented with little resources. The researcher will not use the same versioning system that the developers are using, thus for simplicity, the whole project is copied to the researchers for every retrieval.

To make the assessment as realistic as possible, the software developers of the company should not know which projects are sampled for the assessment. We ensure this by requiring that the developers should not, at any given moment in time, be able to deduce whether a researcher has retrieved a project from the server. This means that the developers are *ignorant* of the researchers.

The company decides that the server where all projects are stored shall expose two web-service endpoints, one for the developers and one for the researchers as illustrated in Fig. 1. Also, the server implementation will be based on J2EE technology, where so-called Java session beans are responsible for handling SOAP-message communication between the server and the clients, and the Java entity beans are responsible for handling persistence.

2.1 Capturing Behavior Using STAIRS

The interaction diagrams in Fig. 2 specify two simple interactions of the PM system in which a developer updates a project by sending an update message to the server, whereupon to server responds by returning an ok or an error message depending on whether or not the update was successful.

In the graphical diagrams, vertical dashed lines correspond to so-called *lifelines*, and the signals of messages correspond the to labels decorating the arrows between the lifelines.

Definition 1. A message is a triple (tr, re, si), consisting of a transmitter tr, a receiver re, and a signal si.

Example 1. The two messages in the left most diagram of Fig. 2 are represented by the triples (D, S, u) and (S, D, o). For conciseness we refer to the lifelines and the signals in graphical diagrams by their first letters if this can be done unambiguously.

In the trace-semantics of STAIRS [15], the transmission and the reception of a message m is represented by a *transmission event* (denoted $(!, m)$) and a *reception event* (denoted $(?, m)$), respectively. For conciseness, an event $(!, (tr, re, si))$ is usually denoted $!si$ if the transmitter and the receiver can be deduced from the context. The same convention applies for reception events.

A *trace* is a sequence of events representing a single run. A trace is causal and weakly sequenced. Causality means that a message must be transmitted before it is received. Weak sequencing [13] means that events are ordered by their vertical position with respect to each lifeline. For example, if two messages are transmitted from a lifeline l_1 to a lifeline l_2, then the first message is not necessarily received by l_2 before the second message is transmitted from l_1.

Definition 2. An event is a pair, (k, m), consisting of a kind $k \in \{!, ?\}$ and a message m. A trace is a sequence of events. We let $\mathcal{H}$ denote the set of all traces.

Example 2. The diagrams PM$_1$ and PM$_2$ in Fig. 2 describe the traces $\langle !u, ?u, !o, ?o\rangle$ and $\langle !u, ?u, !e, ?e\rangle$, respectively.

STAIRS distinguishes between *positive* and *negative* traces. Positive traces represent desired or acceptable behavior, while negative traces represent undesired or unacceptable behavior. The remaining traces are *inconclusive* meaning that their status is not decided or that they are irrelevant. The semantics of a diagram in STAIRS is set of pairs of positive and negative traces. Such pairs are referred to as *interaction obligations*. Each interaction obligation represents an explicit nondeterministic alternative. Underspecification (also referred to as potential nondeterminism) is represented by allowing the implementer to choose between different alternative behaviors within a single interaction obligation.

Definition 3. An interaction obligation is a pair, (p, n), where p is a set of positive traces and n is a set of negative traces. The semantics of a diagram d in STAIRS, written $[\![d]\!]$, is a set of interaction obligations.

Unless otherwise indicated, traces of interaction diagrams are interpreted as positive.

Example 3. Diagram PM$_1$ and PM$_2$ of Fig. 2 describe the interaction obligations $(\{\langle !u, ?u, !o, ?o\rangle\}, \emptyset)$ and $(\{\langle !u, ?u, !e, ?e\rangle\}, \emptyset)$, respectively. The set of negative traces in each interaction obligation is empty since traces of diagrams are interpreted as positive by default.

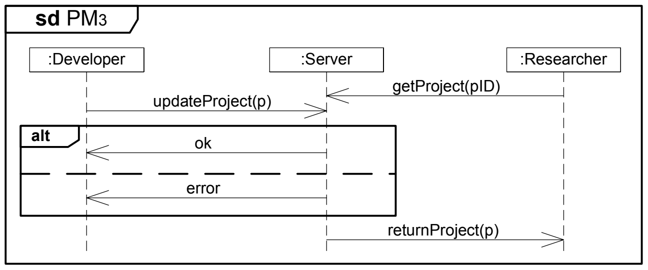

Figure 3: Potential nondeterministic alternatives

2.2 Potential Nondeterminism

The diagram in Fig. 3 specifies the interaction in which a developer tries to update a project before it is retrieved by a researcher who has requested the same project prior to the developer's update attempt. The updated project cannot be stored on the server while it is being read, but whether or not the update will have to be resubmitted (upon reception of an error message) by the developer is a postponed design decision.

In interaction diagrams as interpreted in STAIRS, underspecification in the form of alternative design decisions are explicitly expressed by the so-called **alt**-operator as illustrated in Fig. 3. Interactions that are separated by the **alt**-operator are interpreted as potential nondeterministic trace alternatives. In Fig. 3 the topmost operand of **alt** specifies the design alternative in which an ok message is transmitted back to the developer. The second alternative specifies the alternative where the developer receives an error message.

Definition 4. The **alt**-operator specifies potential nondeterminism by leaving the choice between its operands open. The semantics is the inner union of each point-wise selection of interaction obligations from its operands:

$$[\![\textbf{alt}[d_1, ..., d_n]]\!] \overset{\text{def}}{=} \{\biguplus\{o_1, ..., o_n\} \mid \forall i \in [1, n] : o_i \in [\![d_i]\!]\}$$

The inner union of interaction obligations is defined as:

$$\biguplus_{i\in[1,n]} (p_i, n_i) \overset{\text{def}}{=} (\bigcup_{i\in[1,n]} p_i, \bigcup_{i\in[1,n]} n_i)$$

Example 4. The diagram PM$_3$ in Fig. 3 describes an interaction obligation $(p, \emptyset)$ where p includes the traces $\langle !g, ?g, !u, ?u, !o, ?o, !r, ?r\rangle$ and $\langle !g, ?g, !u, ?u, !e, ?e, !r, ?r\rangle$ due to the **alt**-operator. Note that p will also contain other traces due to weak sequencing. E.g. p will also contain the trace $\langle !g, !u, ?g, ?u, !e, ?e, !r, ?r\rangle$, where both the messages get project ($!g$) and the update project ($!u$) are transmitted by the researcher and the developer, respectively, before the corresponding messages ($?g$ and $?u$) are received by the server.

2.3 Explicit Nondeterminism

In STAIRS we may use the **xalt**-operator to specify explicit nondeterminism. This is illustrated in the interaction diagram of Fig. 4. It states that any correct implementation must offer the choice between the operands PM$_1$, PM$_2$, and PM$_3$. Each alternative specified by an **xalt**-operator represents an explicit nondeterministic choice that must be preserved by any correct refinement of the diagram.

Definition 5. The **xalt**-operator specifies explicit nondeterminism by requiring all operands to be reflected in any

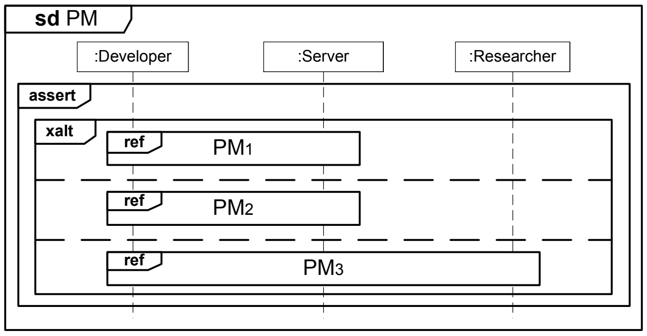

Figure 4: Explicit nondeterministic alternatives

proper implementation. The semantics is the union of the sets of interaction obligations characterized by the operands:

$$[\![\, \textsf{xalt}[d_1, ..., d_n] \,]\!] \stackrel{\mathsf{def}}{=} \bigcup_{i \in [1,n]} [\![\, d_i \,]\!]$$

Assume now for the sake of the running example, that the diagram in Fig. 4 describes all the relevant interactions of the PM system. We may then encapsulate the interactions of the diagram using the **assert**-operator (as shown in Fig. 4), which means that all traces that are not described within the scope of the assert, i.e. the inconclusive traces, are to be interpreted as negative. This is of course an incomplete specification, but interaction diagrams are often used to capture incomplete aspects of a system.

Definition 6. The **assert**-operator makes all inconclusive traces negative. The sets of positive and negative traces are left unchanged:

$$[\![\, \textsf{assert}\, d \,]\!] \stackrel{\mathsf{def}}{=} \{(p, n \cup (\mathcal{H} \setminus p)) \mid (p, n) \in [\![\, d \,]\!]\}$$

Example 5. The STAIRS-semantics interprets the diagram PM of Fig. 4 as $\{(p_1, \mathcal{H} \setminus p_1), (p_2, \mathcal{H} \setminus p_2), (p_3, \mathcal{H} \setminus p_3)\}$ where the trace sets p_1, p_2, and p_3 are described by the diagrams PM$_1$, PM$_2$, and PM$_3$, respectively.

A diagram is *well-defined* if its semantic interpretation does not contain an interaction obligation (p, n) such that $p \cap n \neq \emptyset$. We denote by $\mathcal{D}$ the set of all well-defined diagrams that can be constructed by the syntactic operators presented in this paper.

3. WHY OUR SPECIFICATION ENSURES IGNORANCE

A requirement of the PM system is that the developers should be ignorant of the researchers. In this section we formalize this notion. For this purpose, we distinguish between *observers* and the *system* under observation. Each observer is only able to observe its own interaction with the system, and we assume that each observer has complete knowledge of the diagram describing the system they interact with. Based on this knowledge, an observer can construct the set of all system behaviors which are *compatible* with a given observation, the so-called *inference set*, and try to deduce confidential information from this set. The secure information flow property that we formalize (the ignorance property) demands that an observer o cannot, based on the inferences it can make, deduce with certainty that another observer o' has communicated with the system under observation.

3.1 Systems, Behavior and Observations

In this section we define the notions of system, observer, behavior, and observation in terms of the STAIRS semantics.

Definition 7. Systems and observers are represented by sets of lifelines.

Example 6. In our running example, we require that developers should be ignorant of researchers. Thus developers and researchers are observers. With respect to the diagram PM (of Fig. 4), let D and R denote the observers $\{: \mathit{Developer}\}$ and $\{: \mathit{Researcher}\}$, respectively. Both the developer and the researcher communicate with the same server. Hence, the server is the system under observation. We let S denote the system $\{: \mathit{Server}\}$. This applies throughout the rest of our examples.

STAIRS does not prohibit an implementation of a diagram to produce traces that are not described by the diagram (the inconclusive traces). The set of traces allowed by an interaction obligation is therefore its set of non-negative traces (i.e. the union of its positive and inconclusive traces). Such traces are called *behavior traces*. The set of all behavior traces that are derived from exactly one interaction obligation is called a *behavior alternative*.

In general, an interaction diagram may describe the behavior of several systems or observers. We are usually interested in reasoning about the behavior of a specific system. A behavior alternative of a system s can be obtained from an interaction obligation a by restricting all behavior traces in a to the lifelines in s.

Definition 8. The behavior of a system s in diagram d, written $\nabla(d, s)$, is a set of behavior alternatives:

$$\nabla(d, s) \stackrel{\mathsf{def}}{=} \{\{\textsc{e}.s \circledS h \mid h \in \mathcal{H} \setminus n\} \mid (p, n) \in [\![\, d \,]\!]\}$$

Here $\textsc{e}.s$ yields the set of events that may occur on the lifelines in s. Furthermore, the function $A \circledS h$ yields the trace obtained from trace h by filtering away all events in h that are not in the set of events A[2].

Example 7. Let PM be the diagram specified in Fig. 4. The behavior of system S in PM is given by the behavior traces that can occur on the lifeline in S: $\nabla(\mathrm{PM}, S) = \{b_1, b_2, b_3\}$, where $b_1 = \{\langle ?u, !o \rangle\}$ (corresponding to PM$_1$ in Fig. 2), $b_2 = \{\langle ?u, !e \rangle\}$ (corresponding to PM$_2$ in Fig. 2), and $b_3 = \{\langle ?g, ?u, !o, !r \rangle, \langle ?g, ?u, !e, !r \rangle\}$ (corresponding to PM$_3$ in Fig. 3).

An observer may observe its own interaction with the system under observation. The set of *observation traces* that an observer can make of a system s is obtained from the behavior of s by filtering away the events that are not transmitted to or from the observer in question.

Definition 9. The set of all observation traces that an observer o can make of system s in diagram d, written $o \rhd (d, s)$, is defined:

$$o \rhd (d, s) \stackrel{\mathsf{def}}{=} \bigcup_{b \in \nabla(d,s)} \{\textsc{rt}.o \circledS h \mid h \in b\}$$

[2]Formal definitions of the non-standard operators used in this paper are given in the appendix.

RT.o yields the set of events that can be transmitted to or from the observer o.

The definition (Def. 9) implies that an observer may observe liveness properties (such as termination) because neither the set of behavior traces nor the set of observations traces have to be prefix-closed [3].

Example 8. Based on Fig. 4, we have that $D \rhd (\mathrm{PM}, S)$ yields the observation set $\{\langle ?u, !o\rangle, \langle ?u, !e\rangle\}$, and $R \rhd (\mathrm{PM}, S)$ yields the set $\{\langle ?g, !r\rangle\}$.

3.2 Inference and Ignorance

We assume that observers have full access to the specification of the system, i.e. the observers have complete knowledge of the behavior alternatives of the system under observation. An observer may then, based on a behavior alternative b, construct an *inference set* consisting of all traces in b that are compatible with a given observation h.

Definition 10. The *inference set* that observer o can construct from behavior alternative b, based on observation trace h, written $o \lhd_h b$, is defined:

$$o \lhd_h b \overset{\mathsf{def}}{=} \{h^b \in b \,|\, h = (\mathrm{RT}.o \circledS h^b)\}$$

Example 9. Continuing the previous examples, let h_1 and h_2 be observation traces $\langle ?u, !o\rangle$ and $\langle ?u, !e\rangle$ of observer D, respectively, and let behavior alternative b be the set $\{\langle ?g, ?u, !o, !r\rangle, \langle ?g, ?u, !e, !r\rangle\}$. We have that observer D, based on h_1 can infer the following set of behavior traces from b: $D \lhd_{h_1} b = \{\langle ?g, ?u, !o, !r\rangle\}$. Similarly we have that $D \lhd_{h_2} b = \{\langle ?g, ?u, !e, !r\rangle\}$.

If an inference set that an observer o can construct has a trace that *compromise* another observer o', meaning that the trace contains an event that is sent to or received from o', then o may deduce that o' has done something. Conversely, if none of the traces in the inference set compromise o', then o cannot deduce that o' has done something. We say that a behavior alternative from which such an inference set can be constructed *hides* o'.

Definition 11. Behavior alternative b hides observer o' from observer o wrt. observation h, written $o \wr_{b,h} o'$, iff

$$\forall h^b \in o \lhd_h b : \mathrm{RT}.o' \circledS h^b = \langle\rangle$$

Example 10. Consider diagram PM$_3$ (Fig. 3). Let $b \in \nabla(\mathrm{PM}_3, S)$ and $h \in D \rhd (\mathrm{PM}_3, S)$, such that behavior alternative b equals $\{\langle ?g, ?u, !o, !r\rangle, \langle ?g, ?u, !e, !r\rangle\}$ and observation $h = \langle ?u, !o\rangle$. The set of traces that D can infer from b based on h equals $D \lhd_h b = \{h^b\}$ (where $h^b = \langle ?g, ?u, !o, !r\rangle$). Since the trace h^b contains events that compromises observer R ($\mathrm{RT}.R \circledS h^b = \langle ?g, !r\rangle$), we have that b does not hide R from D wrt. h, i.e. $D \wr_{b,h} R$ yields false.

Observer o is ignorant of observer o' if o cannot infer with certainty that o' has communicated with the system under observation. More precisely, this means that if o, based on an observation h and a behavior alternative b_1, is able to infer a trace that compromises o', then there must exist another behavior alternative b_2 that hides o' from o wrt. the same observation h.

Definition 12. An observer o is ignorant of observer o' wrt. diagram-system pair S_d, written $o \wr^{S_d} o'$, iff

$$\forall h \in o \rhd S_d : \forall b_1 \in \nabla S_d : \neg(o \wr_{b_1,h} o') \Rightarrow$$
$$\exists b_2 \in \nabla S_d : o \wr_{b_2,h} o' \wedge b_2 = o \lhd_h b_2$$

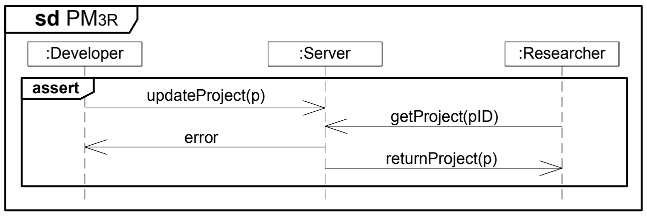

Figure 5: Refinement of diagram PM$_3$

The last condition ($b_2 = o \lhd_h b_2$) ensures that the ignorance property is preserved during refinement. This will be explained in Sect 4.

Example 11. D is ignorant of R wrt. S in diagram PM (Fig. 4). To see this, note that for each observation trace h that D can make of S, we have that for all traces that compromise R that D can infer based on h, there exists a behavior alternative that hides R from D wrt. h. E.g. as described in the previous example, D can infer the compromising trace $\langle ?g, ?u, !o, !r\rangle$ based on observation $h = \langle ?u, !o\rangle$. However, there also exists a behavior alternative $b \in \nabla(\mathrm{PM}, S)$ that hides R wrt. h. Specifically, $b = \{\langle ?u, !o\rangle\}$ (corresponding to the interaction obligation described by diagram PM$_1$ in Fig. 2). We have that D, based on h, can infer the trace $\langle ?u, !o\rangle$ from b. Since this trace does not compromise R ($\mathrm{RT}.R \circledS \langle ?u, !o\rangle = \langle\rangle$), all traces in b (since there is only one trace in b) hide R from D wrt. h.

4. WHY THE IGNORANCE PROPERTY IS PRESERVED BY REFINEMENT

Refinement means to add information to a specification such that the specification becomes more complete. This may be achieved by categorizing inconclusive traces as either positive or negative, or by reducing the set of positive traces. Negative traces always remain negative. We first define refinement of interaction obligations formally. This definition is then lifted to interaction diagrams.

Definition 13. An interaction obligation (p_c, n_c) is a refinement of an interaction obligation (p_a, n_a), written $(p_a, n_a) \rightsquigarrow (p_c, n_c)$, iff

$$n_a \subseteq n_c \wedge p_a \subseteq p_c \cup n_c$$

Definition 14. An interaction diagram d_c is a refinement of an interaction diagram d_a, written $d_a \rightsquigarrow d_c$, iff

$$\forall a \in [\![d_a]\!] : \exists c \in [\![d_c]\!] : a \rightsquigarrow c \quad \wedge$$
$$\forall c \in [\![d_c]\!] : \exists a \in [\![d_a]\!] : a \rightsquigarrow c$$

Example 12. The diagram PM$_{3R}$ (Fig. 5) is a refinement of diagram PM$_3$ (in Fig. 3). That is, $[\![\mathrm{PM}_3]\!] = \{(p_a, \emptyset)\}$ and $[\![\mathrm{PM}_{3R}]\!] = \{(p_c, \mathcal{H} \backslash p_c)\}$ such that $p_c \subset p_a$. PM$_3$ differs from PM$_{3R}$ in that some positive traces in PM$_3$ and all inconclusive traces have been made negative in PM$_{3R}$.

THEOREM 1. *The ignorance property is preserved by refinement, formally:*

$$o \wr^{(d_a,s)} o' \wedge d_a \rightsquigarrow d_c \Rightarrow o \wr^{(d_c,s)} o'$$

The proof[3] of the theorem relies on the fact that the ignorance property is defined in terms of behavior alternatives as opposed to individual traces which is how secure information flow properties are usually defined in the literature [25]. That is, instead of demanding that there exists a trace that fulfills a certain criterion, we demand that there exists a behavior alternative such that *all* its traces fulfills a certain criterion. In Def. 12, this criterion is expressed by the condition $o \wr_{b_2,h} o'$ which requires that none of the traces in b_2 that are compatible with observation h must compromise observer o'. The condition $b_2 = o \lhd_h b_2$ demands that all traces in b_2 must be compatible with observation h. This ensures that all traces in b_2 hide o' from o. Since *all* (not just some) traces in b_2 must fulfill this requirement, the criterion holds regardless of how b_2 is refined. Note that we do not have to worry about empty behavior alternatives, since no well-defined diagram can yield such a behavior alternative given the syntactic constructs presented in this paper.

5. IGNORANCE PRESERVING TRANSFORMATION

The notion of refinement addressed above is a binary relation on specifications that formalizes the process of stepwise development by removal of underspecification. This process depends on human intuition and is in general not subject for automation. Hence, we introduce a notion of transformation that syntactically is a function mapping a specification to another specification (e.g. a PIM to a PSM) in the same manner as a compiler maps a program to machine code. A transformation should be understood a compiler program that takes a specification as input and delivers a new specification as output. Semantically, we represent a syntactic transformation by a set of functions mapping traces over abstract events to traces over concrete events. The reason why a single syntactic transformation is represented by a set of functions is that transformations often introduce a finer granularity in the sense that one abstract trace may correspond to a set of concrete traces. Alternatively, we could have used a function mapping traces to sets of traces.

5.1 Transformations from a Semantic Perspective

We denote by $\mathcal{H}_a$ a set of traces over an abstract set of events, and by $\mathcal{H}_c$ a set of traces over a concrete set of events. Hence, contrary to earlier, we now have an abstract universe $\mathcal{H}_a$ and a concrete universe $\mathcal{H}_c$. Transformations are semantically represented by a set of functions

$$F \subseteq \mathcal{H}_a \rightarrow \mathcal{H}_c$$

that maps abstract traces to concrete traces.

In order to ensure that transformations preserve our security property, we impose two requirements on each function $f \in F$.

First, we require f to be transmitter and receiver preserving in the following sense:

$$f(\mathrm{E}_{l_1,l_2} \circledS h_a) = \mathrm{E}_{l_1,l_2} \circledS f(h_a) \quad (1)$$

Here h_a is universally quantified over $\mathcal{H}_a$. Similarly, l_1 and l_2 range over the set of all lifelines. The function E_{l_1,l_2} yields the set of events that can occur on lifeline l_1 whose message is sent from l_1 to l_2 or from l_2 to l_1.

[3]Detailed proofs of all lemmas and theorems presented in this paper are given in [31].

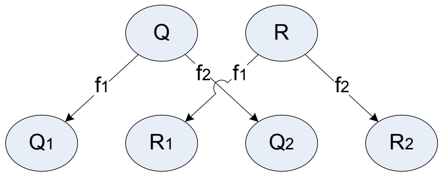

Figure 6: Example of a transformation

Condition (1) is just states that an abstract trace whose events all have the same transmitter and receiver, say t and r, must by transformed to a concrete trace whose events also have the same transmitter and receiver t and r. Note that condition (1) may be weakened; e.g. by allowing lifelines to be decomposed into a set of lifelines.

The second condition requires each $f \in F$ to be *homomorphic* wrt. concatenation of traces:

$$f(h_1) \frown f(h_2) = f(h_1 \frown h_2) \quad (2)$$

The traces h_1 and h_2 are universally quantified over $\mathcal{H}_a$. Condition (2) essentially states that f is a function of *events* rather than trace histories.

Note that requirements (1) and (2) ensure that one abstract observation is transformed to one and only one concrete observation. We let $\mathcal{F}$ denote the set of all functions that abide to requirements (1) and (2) above.

5.2 Ignorance Preserving Transformation of Interaction Diagrams

In the following, d_a and d_c are interactions diagrams with traces in $\mathcal{H}_a$ and $\mathcal{H}_c$, respectively. Moreover, T is a (syntactic) transformation on interaction diagrams whose semantics $[\![T]\!] \subseteq \mathcal{F}$ is a transformation as defined above.

We first define what it means to translate an interaction obligation. Then we lift this notion of translation to interactions diagrams.

Definition 15. Interaction obligation (p_c, n_c) is a translation of interaction obligation (p_a, n_a) with respect to function $f \in \mathcal{F}$, written $(p_a, n_a) \hookrightarrow_f (p_c, n_c)$, iff

$$\mathcal{H}_c \setminus n_c = \{f(h_a) \,|\, h_a \in (\mathcal{H}_a \setminus n_a)\} \wedge p_c = \{f(h) \,|\, h \in p_a\}$$

The first condition states that each non-negative trace of the concrete interaction obligation must be a translation of a non-negative trace of the abstract obligation. The second condition ensures that a positive trace in the abstract obligation is transformed to a positive (not inconclusive) trace in the concrete obligation.

Definition 16. An interaction diagram d_c is a translation of an interaction diagram d_a with respect to transformation T, written, $d_a \hookrightarrow_T d_c$, iff

$$[\![d_c]\!] = \{c \,|\, a \hookrightarrow_f c \wedge a \in [\![d_a]\!] \wedge f \in [\![T]\!]\}$$

If $F_1, F_2 \subseteq \mathcal{F}$ then $F_1 \circ F_2$ is understood as the set of functions obtained by functional point-to-point composition of all functions in F_1 and F_2. That is,

$$F_1 \circ F_2 \overset{\mathsf{def}}{=} \{f_1 \circ f_2 \,| f_1 \in F_1 \wedge f_2 \in F_2\}$$

If T_1 and T_2 are syntactic transformations, we write $T_1 \circ T_2$ instead of $[\![T_1]\!] \circ [\![T_2]\!]$.

LEMMA 1. $\hookrightarrow$ *is transitive, formally:*

$$d_1 \hookrightarrow_{T_1} d_2 \wedge d_2 \hookrightarrow_{T_2} d_3 \Rightarrow d_1 \hookrightarrow_{T_2 \circ T_1} d_3$$

THEOREM 2. *The ignorance property is preserved by transformation, formally*

$$o \wr^{(d_a,s)} o' \wedge d_a \hookrightarrow_T d_c \Rightarrow o \wr^{(d_c,s)} o'$$

The proof relies on two facts. First, all transformations are interpreted by a set of functions where each function by definition abides to conditions (1) and (2) that together ensure that one observation at the abstract level is transformed to one and only one observation at the concrete level. Second, each function that interprets a transformation is by Def. 16 applied to interaction obligations in the manner illustrated in Fig 6. That is, each function is applied to all interaction obligations in a diagram and each interaction obligation on the concrete level is a translation of one and only one interaction obligation on the abstract level. This ensures that additional granularity introduced at the concrete level is in the form of explicit nondeterminism as opposed to potential nondeterminism.

The notions of refinement and transformation may be combined into a more general notion of refinement.

Definition 17. Two interaction diagrams d_a and d_c are in a refinement relation wrt. transformation T, written $d_a \rightsquigarrow_T d_c$, iff

$$\exists d \in \mathcal{D} : d_a \hookrightarrow_T d \wedge d \rightsquigarrow d_c$$

LEMMA 2. $\rightsquigarrow$ *is transitive modulo translation.*

$$d_1 \rightsquigarrow_{T_1} d_2 \wedge d_2 \rightsquigarrow_{T_2} d_3 \Rightarrow d_1 \rightsquigarrow_{T_2 \circ T_1} d_3$$

The results on preservation of the ignorance property carry over to the general notion of refinement.

COROLLARY 1. *The ignorance property is preserved by refinement modulo transformation, formally:*

$$o \wr^{(d_a,s)} o' \wedge d_a \rightsquigarrow_T d_c \Rightarrow o \wr^{(d_c,s)} o'$$

The proof is straightforward given Theorem 1 and 2.

6. TRANSFORMING THE PROJECT MANAGEMENT SYSTEM

In Sect. 5 we showed that our notion of transformation preserves the ignorance property. The purpose of this section is to show that our notion of transformation is not so strong that it cannot be used to semantically characterize transformations of practical value.

In our running example, the developers and the researchers of the PM system communicate with the server by SOAP messages. SOAP, however, is typically bound to HTTP which again typically runs over TCP, etc. In the following we demonstrate how the PM system based on SOAP can be transformed into a system based on TCP in such a way that the ignorance property is guaranteed to be preserved even when developers are able to observe TCP messages as opposed to SOAP messages. In MDA terminology, this can be seen as a transformation from a PIM to a PSM, where the platform is understood as a protocol.

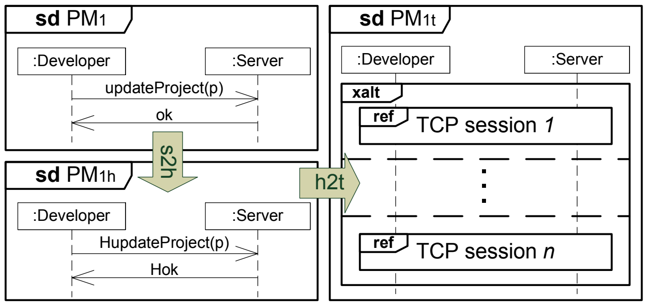

Figure 7: Example transformation

6.1 SOAP to TCP

In practice, a transformation from SOAP to TCP typically works as follows: First each SOAP request and response message is encapsulated by a HTTP request and response header, respectively, then a new so-called TCP session is created for each HTTP request-response pair. That is, for each HTTP request-response pair, (1) a connection is established between the two sides of communication, (2) both the request and response are segmented, encapsulated by TCP frames and transmitted, (3) finally the connection is explicitly terminated. The TCP protocol must, among other things, handle message overtaking and message disappearance. Hence, a HTTP request-response pair may be translated into one of several potential TCP session alternatives.

Let $d_s \hookrightarrow_{s2h} d_h \hookrightarrow_{h2t} d_t$ be two transformations that translate diagrams at the SOAP level to diagrams at the HTTP and TCP levels. $s2h$ may be interpreted by the set $[\![s2h]\!]$ consisting of one bijective function that substitutes the signal si of each event in d_s with a signal that represents its encapsulated HTTP equivalent, e.g. by concatenating the letter "H" and si. $h2t$ can be interpreted as the set $[\![h2t]\!]$ such that each function in $[\![h2t]\!]$ translates a HTTP request message and a HTTP response message into a TCP session alternative. The fact that the functions in $[\![s2h]\!]$ and $[\![h2t]\!]$ satisfy requirements (1) and (2) is almost immediate. (1) is satisfied because the transformation preserves the transmitters and receivers of events. (2) is satisfied because $s2h$ and $h2t$ can be expressed as a function of single events (as opposed to traces). Since translations are transitive by Lemma 2, $d_s \hookrightarrow_{s2h} d_h \hookrightarrow_{h2t} d_t$ is ignorance preserving by Theorem 2.

Example 13. The two left most diagrams in Fig. 7 illustrate the case where $s2h$ is applied to diagram PM$_1$ (also specified in Fig. 2). Here, the SOAP level message names $update(p)$ and ok have been transformed to the HTTP level message names $Hupdate(p)$ and Hok in diagram PM$_{1h}$. Diagrams PM$_{1h}$ and PM$_{1t}$ illustrate the application of $h2t$ where the HTTP request and response messages are translated in a number of different TCP session alternatives. These alternatives are composed by the **xalt**, since by our definition of transformation, each function in $[\![h2t]\!]$ yields an explicit nondeterministic alternative.

7. RELATED WORK

The work that is most related to ours is the work of Jürjens, largely summarized in the book [22]. Not only does he give a formal semantics for a fragment of UML, but he also identifies conditions under which a secure information flow property (among other properties) is preserved under refinement. There are appreciable differences between our work and his. First, he formalizes a fragment of UML in which only a simplified version of sequence diagrams of UML 1.5 is included. In other words, while his work is based on formalized parts of UML that we do not consider, our work is based on STAIRS which provides a more in depth formalization of UML 2.0 interactions than what is considered in Jürjens work. Second, his semantics is based on so-called UML machines which are a kind of state machines. The semantics of STAIRS is not based on state machines. Third, while Jürjens distinguishes between potential and explicit nondeterminism in order to define a refinement preserving property of confidentiality and integrity, he does not rely on this distinction in the definition of his information flow property. That is, a secure information flow property for UML machines is formalized such that each behavior refinement to a *deterministic* UML machine satisfies this property, i.e. he effectively closes the property under refinement without considering explicit nondeterminism. He himself notes this is "rather strong" and it is in fact exactly this kind of definition that we have been trying to avoid. Our experience suggests that information flow properties closed under refinement tend to be too strong for practical use if the distinction between potential and explicit nondeterminism is not taken into consideration. For example, if we had based our running example on a definition of ignorance similar to that which is given in [21] (where explicit nondeterminism is not considered), and then closed this property under refinement without considering explicit nondeterminism, then the property would have been so strong that the example would not have made any sense. Finally, Jürjens does not address MDA or transformations.

Other efforts that are comparable to ours on a general level in that they address security in an MDA-setting and/or model-based security, can be roughly classified into (1) access control related works: [2, 4, 7, 8, 9, 18, 23, 24, 28, 29], (2) secure database development [10], and (3) specification of high-level security requirements [1, 6, 11, 32]. Although all these works are comparable to ours on a general level, they are in fact quite different in that they do not deal with a formalized notion of refinement, nor do they address information flow security.

There are a number of more theoretical papers related to information flow security and refinement. The work of Jacob [21] is related to ours in that our formalization of *ignorance* is based on Jacob's formalization of this property. Jacob's property is, however, not preserved under refinement. In fact, he was the first that we are aware of to show that the derivation of secure systems from security specifications can be practically infeasible. This became known as the *refinement paradox*. It has later been observed that this "paradox" is a manifestation of failing to clearly distinguish between potential and explicit nondeterminism. As far as we are aware of, this observation was first made in [30].

Two notable recent theoretic works that define refinement operators that preserve information flow properties under refinement are [5, 26]. Both papers investigate a number of information flow properties, but their approaches differ from ours. Specifically, [5] presents sufficient conditions with which to check that a given refinement (defined in terms of simulation in a processes calculus) preserves information flow properties. Similarly, [26] presents refinement operators that can be used to check or modify refinements such that security is preserved. Both papers differ from ours in that we address secure information flow preservation in a formalism that distinguishes between potential and explicit nondeterminism. By taking this distinction into consideration in our definition of refinement and the ignorance property, we show that *all* refinements preserve ignorance, thus there is no need to check each specific refinement.

The work of Heisel. et. al. [17] is similar to ours in that they distinguish between potential and explicit nondeterminism. The main differences between their work and ours are: (1) They work in a probabilistic setting in a different formalism than ours and they do not consider UML interaction diagrams. (2) They consider a notion of confidentiality based on low-level indistiguishability. This notion of security is strictly speaking not a secure information flow *property*. (3) Their approach is different from ours in that they build the condition of security preservation into their notion of confidentiality preserving refinement. Wrt. refinement, we take the dual approach of strengthening the notion of security, i.e. by strengthening the ignorance property wrt. its original proposal [21].

8. CONCLUSIONS AND FUTURE WORK

On a general level, we have argued that by integrating security into MDA, security documentation can be specified and analyzed at a high level of abstraction during early phases of system development, thereby reducing the need of ad hoc integration of security mechanisms at the level of implementation. Moreover, analysis is more feasible at high levels of abstraction than at levels closer to implementation because specifications at these levels may be too detailed to make analysis practical.

Our main objective has been to contribute towards a formal foundation for this by presenting an approach where high-level UML inspired interaction diagrams can be analyzed with respect to a secure information flow property, and transformed, if desired, down to the level of sequences of bits in such a way that the property is preserved.

Our approach is based on STAIRS [16], and the contributions of this paper are specifically (1) the formalization of the ignorance property (Sect. 3), (2) the formalization of the notion of refinement that preserves the ignorance property (Sect. 4), and (3) the formalization of ignorance preserving transformations and the general notion of refinement (Sect. 5). We are not aware of any other work that exploits the distinction of potential and explicit nondeterminism in order to define a notion refinement in such a way that all refinements preserve a secure information flow property. Moreover, we are not aware of any work that addresses secure information flow and a notion of transformation that can be used to describe communication at different levels of abstraction in the protocol hierarchy.

A natural direction of future work is to generalize our results to other information flow properties. We are also planning to consider syntactic transformations, and to implement a computerized tool that will automate the security analysis of transformations.

9. ACKNOWLEDGMENTS

The research on which this paper reports has been funded by the Research Council of Norway through the project SECURIS (152839/220). We would like to thank Fabio Massacci and Jan Øyvind Aagedal for useful comments and suggestions.

10. REFERENCES

[1] H. Abie, D. B. Aredo, T. Kristoffersen, S. Mazaher, and T. Raguin. Integrating a security requirement language with UML. In *UML 2004 - The Unified Modelling Language: Modelling Languages and Applications*, volume 3273 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2004.

[2] K. Alghathbar and D. Wijesekera. Consistent and complete access control policies in use cases. In *UML 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference*, volume 2863 of *Lecture Notes in Computer Science*, pages 373–387. Springer, 2003.

[3] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[4] D. Basin, J. Doser, and T. Lodderstedt. Model driven security for process-oriented systems. In *SACMAT 2003, 8th ACM Symposium on Access Control Models and Technologies*, pages 100–109. ACM Press, 2003.

[5] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Refinement operators and information flow security. In *1st International Conference on Software Engineering and Formal Methods (SEFM 2003)*, pages 44–53. IEEE Computer Society Press, 2003.

[6] R. Breu, M. Hafner, B. Weber, and A. Novak. Model driven security for inter-organizational workflows in e-government. In *E-Government: Towards Electronic Democracy, International Conference, TCGOV 2005*, volume 3416 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2005.

[7] C. C. Burt, B. R. Bryant, R. R. Raje, A. M. Olson, and M. Auguston. Model driven security: unification of authorization models for fine-grain access control. In *7th International Enterprise Distributed Object Computing Conference (EDOC 2003)*, pages 159–173. IEEE Computer Society Press, 2003.

[8] T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl. MAC and UML for secure software design. In *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 75–85. ACM Press, 2004.

[9] P. Epstein and R. Sandhu. Towards a UML based approach to role engineering. In *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*, pages 135–143. ACM Press, 1999.

[10] E. Fernández-Medina and M. Piattini. Extending OCL for secure database development. In *UML 2004 - The Unified Modelling Language: Modelling Languages and Applications. 7th International Conference*, volume 3273 of *Lecture Notes in Computer Science*, pages 380–394. Springer, 2004.

[11] P. Giorgini, F. Massacci, and J. Mylopoulos. Requirement engineering meets security: a case study on modelling secure electronic transactions by VISA and mastercard. In *Conceptual Modeling - ER 2003, 22nd International Conference on Conceptual Modeling*, volume 2813 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2003.

[12] O. M. Group. *OMG Unified Modeling Language Specification v.1.5.* Version 1.5. OMG Document formal/03-03-01, 2003.

[13] O. M. Group. *UML 2.0 Superstructure Specification.* OMG Adopted Sepcification ptc/03-08-02, 2003.

[14] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Research Report 309, Department of Informatics, University of Oslo, 2004.

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

[16] Ø. Haugen and K. Stølen. STAIRS – steps to analyse interactions with refinement semantics. In *Sixth International Conference on UML (UML'2003)*, Lecture Notes in Computer Science, pages 388–402. Springer, 2003.

[17] M. Heisel, A. Pfitzmann, and T. Santen. Confidentiality-preserving refinement. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001)*, pages 295–306. IEEE Computer Society Press, 2001.

[18] R. Heldal and F. Hultin. Bridging model-based and language-based security. In *Computer Security - ESORICS 2003, 8th European Symposium on Research in Computer Security*, volume 2808 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2003.

[19] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[20] C. A. R. Hoare. *Communicating Sequential Processes.* Series in computer science. Prentice-Hall, 1985.

[21] J. Jacob. On the derivation of secure components. In *Security and Privacy*, pages 242–247. IEEE Computer Society Press, 1989.

[22] J. Jürjens. *Secure systems development with UML.* Springer, 2005.

[23] M. Koch and F. Parisi-Presicce. Formal access control analysis in the software development process. In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 67–76. ACM Press, 2003.

[24] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML 2002 - The Unified Modeling Language: 5th International Conference*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer, 2002.

[25] H. Mantel. Possibilistic definitions of security - an assembly kit. In *IEEE Compuer Security Foundations Workshop (CSFW'00)*, pages 185–199. IEEE Computer Society Press, 2000.

[26] H. Mantel. Preserving information flow properties under refinement. In *IEEE Symposium on Security and Privacy*, pages 78–91. IEEE Computer Society Press, 2001.

[27] O. M. G. A. B. ORMSC. *Model Driven Architecture (MDA).* Document number ormsc/2001-07-01, 2001.

[28] A. Poniszewska-Maranda, G. Goncalves, and F. Hemery. Representation of extended RBAC model using UML language. In *SOFSEM 2005: Theory and Practice of Computer Science, 31st Conference on Current Trends in Theory and Practice of Computer Science*, volume 3381 of *Lecture Notes in Computer Science*, pages 413–417. Springer, 2005.

[29] I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to visualize role-based access control constraints. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 115–124. ACM Press, 2004.

[30] A. Roscoe. CSP and determinism in security modelling. In *IEEE Symposium on Security and Privacy*, pages 114–127. IEEE Computer Society Press, 1995.

[31] F. Seehusen and K. Stølen. Information flow property preserving transformation of UML interaction diagrams. Technical report STF90 A06030, SINTEF ICT, 2006.

[32] J. L. Vivas, J. A. Montenegro, and J. Lopez. Towards a business process-driven framework for security engineering with the UML. In *Information Security, 6th International Conference, ISC 2003*, volume 2851 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2003.

APPENDIX

A. SET NOTATION

Standard set-operations are given below.

Set			*Meaning*
$\mathbb{P}(A)$	$\overset{\text{def}}{=}$	$\{X \mid X \subseteq A\}$	The power set of A.
A^ω	$\overset{\text{def}}{=}$		The set of all sequences over the set A.

A list of set-conventions is given below.

Set			*Meaning*
$\mathcal{L}$	$\overset{\text{def}}{=}$		The set of all lifelines.
$\mathcal{SI}$	$\overset{\text{def}}{=}$		The set of all signals.
$\mathcal{M}$	$\overset{\text{def}}{=}$	$\mathcal{L} \times \mathcal{L} \times \mathcal{SI}$	The set of all messages.
$\mathcal{K}$	$\overset{\text{def}}{=}$	$\{!, ?\}$	The set of all kinds.
$\mathcal{E}$	$\overset{\text{def}}{=}$	$\mathcal{K} \times \mathcal{M}$	The set of all events.
$\mathcal{H}$	$\overset{\text{def}}{=}$	$\mathcal{E}^\omega$	The trace universe.
$\mathcal{S}$	$\overset{\text{def}}{=}$	$\mathbb{P}(\mathcal{L})$	The set of all systems.

B. AUXILIARY OPERATORS

Formal definitions of non-standard operators are given below.

Function	$\text{K}.\_ \in \mathcal{E} \to \mathcal{K}$
Def.	$\text{K}.e \overset{\text{def}}{=}\, !$ if $e \in \{!\} \times \mathcal{M}$. ? otherwise.
Meaning	The kind (output or input) of event e.
Function	$\text{TR}.\_ \in \mathcal{E} \to \mathcal{L}$
Def.	$\text{TR}.e \overset{\text{def}}{=} t$, where $e \in \{t\} \times \mathcal{L} \times \mathcal{SI}$
Meaning	The transmitter of event e.
Function	$\text{RE}.\_ \in \mathcal{E} \to \mathcal{L}$
Def.	$\text{RE}.e \overset{\text{def}}{=} r$, where $e \in \mathcal{L} \times \{r\} \times \mathcal{SI}$
Meaning	The transmitter of event e.
Function	$\text{E}.\_ \in \mathcal{S} \to \mathbb{P}(\mathcal{E})$
Def.	$\text{E}.s \overset{\text{def}}{=} \{e \in \mathcal{E} \mid (\text{K}.e = ! \wedge \text{TR}.e \in s) \vee (\text{K}.e = ? \wedge \text{RE}.e \in s)\}$
Meaning	The set of events that can occur on the lifelines of system s.
Function	$\text{RT}.\_ \in \mathcal{S} \to \mathbb{P}(\mathcal{E})$
Def.	$\text{RT}.s \overset{\text{def}}{=} \{e \in \mathcal{E} \mid \text{TR}.e \in s \vee \text{RE}.e \in s\}$
Meaning	The set of events whose message can be sent to or from the lifelines in s.
Function	$\text{E}_{\_,\_} \in \mathcal{L} \times \mathcal{L} \to \mathbb{P}(\mathcal{E})$
Def.	$\text{E}_{l_1,l_2} \overset{\text{def}}{=} \text{E}.\{l_1\} \cap \text{RT}.\{l_2\}$
Meaning	The set of events on lifeline l_1 whose message can be sent from l_1 to l_2 or from l_2 to l_1.