Towards Development of Secure Systems Using UMLsec

Jan Jürjens*

Computing Laboratory, University of Oxford, GB

Abstract. We show how UML (the industry standard in object-oriented modelling) can be used to express security requirements during system development. Using the extension mechanisms provided by UML, we incorporate standard concepts from formal methods regarding multi-level secure systems and security protocols. These definitions evaluate diagrams of various kinds and indicate possible vulnerabilities.
On the theoretical side, this work exemplifies use of the extension mechanisms of UML and of a (simplified) formal semantics for it. A more practical aim is to enable developers (that may not be security specialists) to make use of established knowledge on security engineering through the means of a widely used notation.

1 Introduction

There is presently an increased need to consider security aspects when developing systems (that may e.g. communicate over untrusted networks). This is not always met by adequate knowledge on the side of the developer. This is problematic since in practice, security is compromised most often not by breaking the dedicated mechanisms (such as encryption or security protocols), but by exploiting weaknesses in the way they are being used [And94]. Thus security mechanisms cannot be "blindly" inserted into a security-critical system, but the overall system development must take security aspects into account.

Object-oriented systems offer a very suitable framework for considering security due to their encapsulation and modularisation principles.

The Unified Modeling Language (UML) [RJB99] is an industry standard for specifying object-oriented software systems. Compared to other modelling languages, UML is very precisely defined.

The aim of this work is to use UML to encapsulate knowledge on prudent security engineering and thereby make it available to developers not specialised in security.

This work profits from a work towards a formal semantics for UML e. g. in [EFLR99,RACH00]. Its aim is to make formalism have an impact in the actual software development process, [Huß99,AR00].

* `http://www.jurjens.de/jan` – `jan@comlab.ox.ac.uk` - Supported by the Studienstiftung des deutschen Volkes and the Computing Laboratory.

H. Hussmann (Ed.): FASE 2001, LNCS 2029, pp. 187–200, 2001.

For space limitations, we present our results in the framework of a simplified fragment of UML (in its current version 1.3 [For99]), the extension to the whole of the actual UML is under development [Jür01b].

After presenting some background and related work in the following subsection, we summarise our use of UML in the next section. In the later sections we show how to use four central kinds of diagrams of UML to develop security-critical systems. We end with a conclusion and indicate future work.

1.1 Background and Related Work

A traditional way of ensuring security in computer systems is to design *multi-level secure* systems (LaPadula, Bell 1973). There are levels of sensitivity of data (usually *high* and *low*). To ensure confidentiality of data, one enforces the policy that information always flows up: *low* data may influence *high* data, but not vc. vs. (the opposite of this condition provides data *integrity*; thus also integrity is implicitly covered by our approach). To avoid *covert channels* one can use the notion of *noninterference* (Goguen, Meseguer 1982; cf. e. g. [Jür00]). Secure communication over untrusted networks requires specific mechanisms such as encryption and cryptographic protocols.

There has been much work on security using formal methods (e. g. [BAN89, RWW94,Low96,AJ00,Jür01c]), mostly about security protocols or secure information flow.

[And94] suggests to use software engineering techniques to ensure security. [DS00] proposes to "extend the syntax and semantics of standards such as UML to address security concerns". We are not aware of any published work towards this goal.

Our work depends on concepts from the considerable work towards providing a formal semantics for UML (e. g. [BGH$^+$98,EFLR99,GPP98,RCA00,RACH00, BD00,Öve00]). In particular the security notions concerning behaviour cannot be formulated at the level of abstract syntax, requiring a formal semantics to express and reason about the security requirements. As a formal semantics for UML is subject of ongoing research, we use a (simplified) semantics tailored to our needs for the time being. It would be preferable to employ a universally defined formal semantics, therefore our work may contribute to exemplify the need for a formal semantics for UML.

2 Developing Secure Systems with UML

UML consists of diagram types describing views on a system (an excellent introduction is [SP00]). We use an extension of UML (called UMLsec) to specify standard security requirements on security-critical systems. Our results may be used with any process; security-critical systems usually require an iterative approach (e. g. the "Spiral") [And94].

We concentrate on the most important kinds of diagrams for describing object-oriented software systems (following [RACH00]):

Class diagrams define the static structure of the system: classes with attributes and operations/signals and relationships between classes. We use them to ensure that exchange of data obeys security levels.

Statechart diagrams give the dynamic behaviour of an individual object: events may cause state in change or actions. We use them to prevent indirect information flow from high to low values within an object.

Interaction diagrams describe interaction between objects via message exchange. We use sequence diagrams to ensure correctness of security-critical interaction between objects (especially in distributed object systems).

Since security of a software system depends on the security of the underlying physical layer, we additionally use **deployment diagrams** to ensure that security requirements on communication are met by the physical layer.

To specify security levels we use the UML extension construct of a *tag* to indicate model elements with high security level.

We formulate our concepts at this point on the level of abstract syntax as far as possible (i. e. those concerning static aspects), given in set-theoretical terms for brevity. Behavioural aspects (modeled using statechart and sequence diagrams) can not be treated this way. In absence of a general formal semantics we use a specifically defined one which covers just the aspects needed and to the needed degree of detail, to follow space restrictions (a more complete account is to be found in [Jür01b]).

We stress that the aspects that are left out (such as association and generalisation in the case of class diagrams) can and should be used in the context of our usage of UML; they do not appear in our presentation simply because they are not needed to specify the considered properties (and for lack of space).

It should also be stressed that the formal semantics is used here only to convey our ideas on how to model security aspects of systems using UML and not to provide a semantics for general use. We will translate our definitions to a generally defined formal semantics when available.

3 Classes and State

We give the abstract syntax for class models. Besides the usual information on attributes and operations/signals we also have interfaces. Additionally, the tag {high} is used to mark data items (attributes, arguments of operations or signals or return values of operations) of the corresponding security level (absence of this tag is interpreted as the level low).[1] In the concrete syntax, these tags are written behind the data types. Following [RACH00] we assume that the attributes are fully encapsulated by the operations, i. e. an attribute of a class can be read and updated only by the class operations.

[1] More precisely, UML provides tag-value pairs. Here we use the convention that where the values are supposed to be boolean values, they need not be written (then presence of the label denotes the value true, and absence denotes false).

For brevity, our abstract syntax only gives the aspects we use to specify our security condition. [EFLR99] has a more complete account.

An *attribute specification* $A = (\mathsf{att\_name}, \mathsf{att\_type}, \mathsf{init\_value}, \mathsf{att\_tags})$ is given by a name att_name, a type att_tags, an initial value init_value[2] and a set of tags att_tags (here we only care if it contains the tag $\{\mathsf{high}\}$ or not, determining the security level of the attribute).

An *operation specification* $O = (\mathsf{op\_name}, \mathsf{Arguments}, \mathsf{op\_type}, \mathsf{re\_tags})$ is given by a name op_name, a set of Arguments, the type op_type of the return value and a set re_tags of tags denoting the security level of the return value. The set of arguments may be empty and the return type may be the empty type $\emptyset$ denoting absence of a return value. An *argument* $A = (\mathsf{arg\_name}, \mathsf{arg\_type}, \mathsf{arg\_tags})$ is given by its name arg_name, its type arg_type and a set arg_tags denoting the security level of the argument.

A *signal specification* is like an operation specification, except that there is no return type and no corresponding set of tags.

An *interface* $I = (\mathsf{int\_name}, \mathsf{Operations}, \mathsf{Signals})$ is given by a name int_name and sets of operation names Operations and signal names Signals giving the operations and signals that can be called or sent through it.

A *class model* $C = (\mathsf{class\_name}, \mathsf{AttSpecs}, \mathsf{OpSpecs}, \mathsf{SigSpecs}, \mathsf{Interfaces}, \mathsf{State})$ is given by a name class_name, a set of attribute specifications AttSpecs, a set of operation specifications OpSpecs, a set of signal specifications SigSpecs, a set of class interfaces Interfaces and a statechart diagram State giving the object behaviour. We require that in the set of attribute specifications, the attribute names are mutually distinct, so that an attribute is uniquely specified by its name and the name of its class (and similarly for operation and signal specifications and class interfaces).

3.1 Statechart Diagrams

We fix a set Var of (typed) variables $x, z, y, \ldots$ used in statechart diagrams. At this point, we only consider simple states with one thread of execution, because the formal semantics of more complex features raises some questions [RACH00].

We define the notion of a statechart diagram for a given class model C: A *statechart diagram* $S = (\mathsf{States}, \mathsf{init\_state}, \mathsf{Transitions})$ is given by a set of States (that includes the initial state init_state) and a set of Transitions. (On the level of concrete syntax, the initial state is the state with an in-going transition from the start marker.)

A *statechart transition* $t = (\mathsf{source}, \mathsf{event}, \mathsf{guard}, \mathsf{Actions}, \mathsf{target})$ has a source state, an event, a guard, a list of Actions and a target state. Here an *event* is the name of an operation or signal with a list of distinct variables as arguments that is assumed to be well-typed (e. g. $\mathsf{op}(x, y, z)$). Let the set Assignments consist of all partial functions that assign to each variable and each attribute of the class C a value of its type (partiality arises from the fact that variables may be undefined). A *guard* is a function $g : \mathsf{Assigments} \to \mathsf{Bool}$ evaluating each

[2] We assume that these are specified in the class rather than by the creating object.

assignment to a boolean value. (On the level of UML's concrete syntax, it is left open how to write such guards; often it is done in OCL.) An *action* can be either to assign a value v to an attribute a (written $a := v$), to call an operation op resp. to send a signal sig with values $v_1, \ldots, v_n$ (written $\mathsf{op}(v_1, \ldots, v_n)$ resp. $\mathsf{sig}(v_1, \ldots, v_n)$), or to return values $v_1, \ldots, v_n$ as a response to an earlier call of the operation op (written $\mathsf{return}_{\mathsf{op}}(v_1, \ldots, v_n)$). In each case, the values can be constants, variables or attributes (and need to be well-typed). In the case of *output actions* (calling an operation or sending a signal) we include the types of the arguments (and possibly of the return value) together with any security tags. (This is not necessary in the usual UML syntax; we need this extra information later.)

Interpreting statechart diagrams. To define our security condition on statechart diagrams we need to interpret them as state machines. Due to space constraints we can only sketch this interpretation, a detailed account is in [Jür01b].

Suppose we are given a class model C with a statechart diagram S. We define the associated state machine $M_S \overset{\text{def}}{=} (\mathcal{S}, i, \mathcal{T})$ to consist of

- a set of states $\mathcal{S} \overset{\text{def}}{=} \mathsf{States} \times \mathsf{Assignments}$,
- an initial state $i \overset{\text{def}}{=} (\mathsf{init\_state}, \mathsf{init\_as})$ (where the initial assignment $\mathsf{init\_as}$ maps each attribute to its initial value and is undefined on the set of variables) and
- a set $\mathcal{T}$ of state machine transitions $t = (s, e, A, s')$ (for states s, s', a trigger e and a list of actions A) defined as follows. A trigger is the name of an operation or signal with a list of (suitably typed) values as arguments.

Firstly, for any assignment function ϕ, a list of variable or attribute names $\boldsymbol{x} = (x_1, \ldots, x_n)$ and a list of values $\boldsymbol{v} = (v_1, \ldots, v_n)$ of the corresponding types, we define $\phi[\boldsymbol{x} \mapsto \boldsymbol{v}]$ by $\phi[\boldsymbol{x} \mapsto \boldsymbol{v}](z) \overset{\text{def}}{=} \phi(z)$ for an attribute or variable z not in $\boldsymbol{x}$ and $\phi[\boldsymbol{x} \mapsto \boldsymbol{v}](x_i) \overset{\text{def}}{=} v_i$ for $i = 1, \ldots, n$.

Suppose we are given a transition $t = (\mathsf{source}, \mathsf{event}, \mathsf{guard}, \mathsf{Actions}, \mathsf{target}) \in \mathsf{Transitions}$ in the statechart diagram (where the event has the list of variables $\boldsymbol{x} = (x_1, \ldots, x_n)$), a corresponding list of values $\boldsymbol{v} = (v_1, \ldots, v_n)$ and an assignment ϕ such that $\mathsf{guard}(\phi[\boldsymbol{x} \mapsto \boldsymbol{v}]) = \mathsf{true}$. Denote the list of attributes that are assigned new values in the list of $\mathsf{Actions}$ by $\boldsymbol{a}$ and the corresponding list of values by $\boldsymbol{w}$. We define the state machine transition $t_{\boldsymbol{v},\phi} \overset{\text{def}}{=} ((\mathsf{source}, \phi), \mathsf{event}(\boldsymbol{v}), \mathsf{Actions}', (\mathsf{target}, \phi[\boldsymbol{x} \mapsto \boldsymbol{v}][\boldsymbol{a} \mapsto \boldsymbol{w}]))$ where $\mathsf{event}(\boldsymbol{v})$ is the trigger obtained from event by substituting the values $\boldsymbol{v}$ for the variables $\boldsymbol{x}$ and $\mathsf{Actions}'$ is the list of actions obtained from $\mathsf{Actions}$ by removing the attribute assignments and by instantiating any variables or attributes appearing as parameters using the assignment $(\phi[\boldsymbol{x} \mapsto \boldsymbol{v}])[\boldsymbol{a} \mapsto \boldsymbol{w}]$. Then we define $\mathcal{T} \overset{\text{def}}{=} \{t_{\boldsymbol{v},\phi} : \mathsf{guard}(\phi[\boldsymbol{x} \mapsto \boldsymbol{v}]) = \mathsf{true}\}$.

Assume for the moment that the state machine under consideration is deterministic in the sense that at any state, any given trigger can fire at most one transition. Any state s of the state machine defines a function $[s]$ from sequences of triggers to sequences of actions as follows. For a given sequence of triggers $\boldsymbol{e} = (e_1, \ldots, e_n)$ we obtain the actions performed by an object in reaction to these triggers as follows: If there exists a transition $s \overset{e_1;a_1,\ldots,a_m}{\longrightarrow} s'$ for some list of actions $a_1, \ldots, a_m$ (which is the only transition triggered by e_1 by determinism), then define $[s](\boldsymbol{e}) \overset{\text{def}}{=} (a_1, \ldots, a_m).[\![s]\!]((e_2, \ldots, e_n))$ (where . denotes concatenation of tuples), otherwise define $[s](\boldsymbol{e}) \overset{\text{def}}{=} [\![s]\!]((e_2, \ldots, e_n))$ (here we follow the UML convention that events with unspecified reaction are ignored).

Thus any deterministic state machine S gives a function $[\![S]\!]$ from sequences of triggers to sequences of actions per $[\![S]\!](\boldsymbol{e}) \overset{\text{def}}{=} [\mathsf{init\_state}_S](\boldsymbol{e})$. We generalise this to *nondeterministic* state machines S by defining $[\![S]\!]$ to be the set of functions $[\![T]\!]$ for all (deterministic) state machines T constructed by choosing one transition for each trigger in each state in S (where such a transition exists).

Note that we follow [RACH00] in considering sequences of events rather than multisets (*queues* in UML terminology) to ensure preservation of event orderings *within* a state machine, but we can easily derive the case for multisets by considering all possible sequences that give rise to a given multiset. – Note that in the *interaction* between state machines, preservation of message ordering can not be assumed.

We define the *low view* $\mathcal{L}(\boldsymbol{e})$ of a sequence $\boldsymbol{e}$ of triggers (resp. actions) to be the sequence obtained from $\boldsymbol{e}$ by substituting all arguments of types marked $\{\mathsf{high}\}$ by the symbol $\square$.

For example, the low view of the action sequence $(\mathsf{op}_1(2,5,7), \mathsf{op}_2(4,3))$, with operation specifications

$$(\mathsf{op}_1, ((a_1, \mathsf{int}, \emptyset), (a_2, \mathsf{int}, \mathsf{high}), (a_3, \mathsf{int}, \emptyset)), \mathsf{int}, \{\mathsf{high})\} \text{ and}$$
$$(\mathsf{op}_2, ((a_3, \mathsf{int}, \{\mathsf{high}\}), (a_4, \mathsf{int}, \{\mathsf{high}\})), \mathsf{int}, \emptyset)$$

is $(\mathsf{op}_1(2, \square, 7), \mathsf{op}_2(\square, \square))$.

Definition 1 *An object* preserves security *if in its statechart diagram S, no* low *output value depend on* $\{\mathsf{high}\}$ *input values, i. e. if for all functions $h \in [\![M_S]\!]$ (where M_S is the associated state machine) and all trigger sequences $\boldsymbol{e}, \boldsymbol{f}$, $\mathcal{L}(\boldsymbol{e}) = \mathcal{L}(\boldsymbol{f})$ implies $\mathcal{L}(h(\boldsymbol{e})) = \mathcal{L}(h(\boldsymbol{f}))$.*

This is a simple generalisation of the notion of *noninterference* [GM82] to the nondeterministic case. It is possible to refine this notion to allow *encrypted* low data to depend on high data (e. g. to encrypt high data and then send it out on an untrusted network); we cannot give the details here for lack of space.

Example: Entry in multi-level database. The object in Figure 1 does not preserve security (the operation $\mathsf{rx}()$ leaks information on the account balance).

3.2 Class Diagrams

A *class diagram* $D = (\mathsf{Cls}, \mathsf{Dependencies})$ is given by a set Cls of class models and a set of $\mathsf{Dependencies}$. A *dependency* is a tuple $(\mathsf{client}, \mathsf{supplier},$

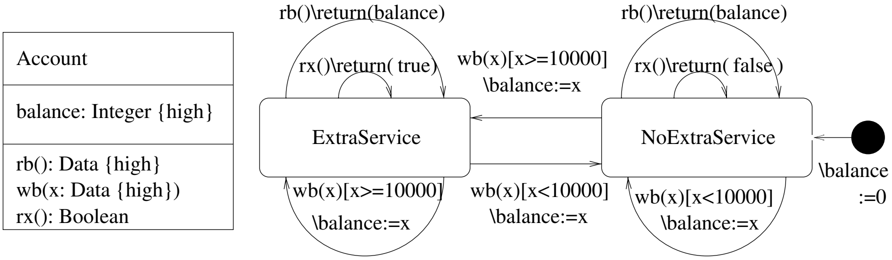


Fig. 1. Multi-level database

interface, stereotype) consisting of class names client and supplier (signifying that client depends on supplier), an interface name interface (giving the interface of the class supplier through which client accesses supplier; if the access is direct this field contains the client name) and a stereotype which for our present purposes can be either «send» or «call». We require that the names of the class models are mutually distinct.

Definition 2 *A class diagram D gives* secure dependency *if whenever there is a dependency with stereotype* «send» *or* «call» *from a client A to an interface I of a supplier B then for each operation or signal o specified in I the following holds:*

- *the security levels on the argument values of o in the statechart diagram of A agree with those of o in the class diagram of B and*
- *the security levels of the return value of o (when o is an operation) in the class diagram of B agree with those of o in the statechart diagram of A.*

If the statechart diagram is not specified for a certain class model, the security levels can be given in a special model element newly introduced here for this purpose, the *output action list*, which specifies the operations an object can call and the signals an object can send.[3] An output action list is a classifier given by a rectangle with the keyword «output actions», carrying the name of the corresponding class and with the operations and signals (with their types and security levels) listed in a compartment of the rectangle. It is similar but dual to an interface.

Example. In Figure 2, the random() operation of the server does not provide the security level required by the client for the seed.

[3] In future work we will examine the possibility to employ UML-RT capsules in this context.

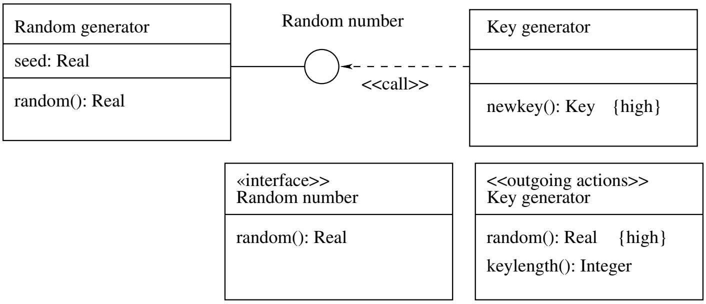


Fig. 2. Key generator

4 Interaction

We would like to model security-aspects in particular of distributed objects systems where messages are sent over networks (e. g. the Internet) where they can be intercepted, modified or deleted. Here the security-critical part of the interaction between objects is the exchange of encryption keys etc. by means of cryptographic protocols. Since it is not always possible to use protocols off-the-shelf, but they have to be adjusted to the specific application domain (cf. the example below), and because vulnerabilities often arise at the boundary between a protocol and the rest of the system [Aba00], we would like to specify cryptographic protocols within the general framework of a security-enhanced UML, which we do using using sequence diagrams.

We concentrate on one kind of interaction diagram, the sequence diagrams (collaboration diagrams are very similar). We first give an abstract syntax of aspects of sequence diagrams that allow specification of security-critical interaction between objects (specifically employing cryptography). We proceed to give a formal semantics tailored to our purposes. Again, a more general account can be found elsewhere [IT95]. We concentrate on concurrent objects that each have their own lifelines and exchange messages asynchronously (by sending signals). As pointed out in [BGH$^+$98], sequential systems are a special case of concurrent ones, and synchronous communication (i. e. operation calls) can be modeled using handshake.

A *sequence diagram* $S = (\mathsf{Obj}, \mathsf{MsgSpecs})$ is given by a list Obj of pairs ($\mathsf{obj\_name}$, $\mathsf{cls\_name}$) (where $\mathsf{obj\_name}$ may be an empty string) and a list of message specifications $\mathsf{MsgSpecs}$. A *message specification* $m = (\mathsf{sender}, \mathsf{receiver}, \mathsf{guard}, \mathsf{signal}(\mathsf{args}))$ consists of the names of the sender and the $\mathsf{receiver}$, a guard, the name of the signal sent and a list of arguments args. We assume that the messages can be ordered linearly, following the view that two events never occur at exactly the same time [RJB99, p.440]; the more general case follows by using a non-deterministic scheduler. As above, a *guard* is a function $g : \mathsf{Assigments} \to \mathsf{Bool}$ evaluating each assignment to a boolean value (where an assignment is any partial function $f : \mathbf{Var} \rightharpoonup \mathbf{Exp}$ similar to above). Here we assume the set $\mathbf{Var}$ to consist of the variables $\mathsf{msg}_{(m,n)}$ representing the n^{th} argument of the signal with name m received most recently.

We define the data type **Exp** of cryptographic messages that can be exchanged during the interaction. We assume a set $\mathcal{D}$ of basic data values. The set **Exp** contains the expressions defined inductively by the grammar

$E ::=$	expression
d	data value ($d \in \mathcal{D}$)
K	key ($K \in \mathbf{Keys}$)
x	variable ($x \in \mathbf{Var}$)
$E_1 :: E_2$	concatenation
$\{E\}_e$	encryption ($e \in \mathbf{Keys} \cup \mathbf{Var}$)
$\mathcal{D}ec_e(E)$	decryption ($e \in \mathbf{Keys} \cup \mathbf{Var}$)

Here we consider asymmetric encryption, so the set **Keys** is the disjoint union of the sets of private and public keys, where the private keys corresponding to a public key K is denoted K^{-1}. We assume $\mathcal{D}ec_{K^{-1}}(\{E\}_K) = E$, and for the following example (for RSA signing) also $\{\mathcal{D}ec_{K^{-1}}(E)\}_K = E$ (for all $E \in \mathbf{Exp}, K, K^{-1} \in \mathbf{Keys}$), and that no other equations except those implied by these hold. We write **Msg** for the set of *messages* of the form $\mathsf{sig}(E_1, \ldots, E_n)$ where sig is a signal with n arguments of type **Exp**.

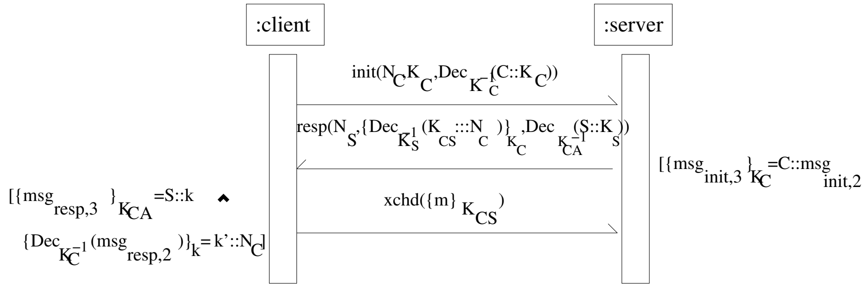

:client
:server
$\mathrm{init}(N_C, K_C, \mathrm{Dec}_{K_C^{-1}}(C::K_C))$
$\mathrm{resp}(N_S, \{\mathrm{Dec}_{K_S^{-1}}(K_{CS}:::N_C)\}_{K_C}, \mathrm{Dec}_{K_{CA}^{-1}}(S::K_S))$
$[\{msg_{init,3}\}_{K_C} = C::msg_{init,2}$
$[\{msg_{resp,3}\}_{K_{CA}} = S::k$ ∧
$\{\mathrm{Dec}_{K_C^{-1}}(msg_{resp,2})\}_k = k'::N_C]$
$\mathrm{xchd}(\{m\}_{K_{CS}})$

Fig. 3. Variant of TLS

Example: Proposed variant of TLS. The protocol in Figure 3 has been proposed in [APS99] as a variant of the handshake protocol of TLS (the successor of the Internet protocol SSL) to satisfy certain performance constraints (for more details cf. [Jür01c]).

4.1 Interpreting Sequence Diagrams

To specify security properties we give a formal interpretation of sequence diagrams in the specification framework Focus [BS00] which was suggested in [BGH+98] for a formal semantics of UML and was used e. g. in [Jür01c,AJ00] to reason about security. In Focus, one can model concurrently executing systems interacting by transmitting sequences of data values over unidirectional

FIFO communication channels. Communication is asynchronous in the sense that transmission of a value cannot be prevented by the receiver. Focus uses streams and stream-processing functions defined in the following.

For a set C we write $\mathbf{Stream}_C \overset{\text{def}}{=} (\mathbf{Msg}^{\omega})^C$ for the set of C-indexed tuples of (finite or infinite) sequences of messages, the (untimed) *streams.* A function $f : \mathbf{Stream}_I \to \mathcal{P}(\mathbf{Stream}_O)$ from streams to sets of streams represents a reactive system that receives an input stream $\boldsymbol{s}$ on the input channels in I and nondeterministically sends out an output stream (from the set of possible output streams $f(\boldsymbol{s})$) on the output channels in O.

The composition $f_1 \otimes f_2 : \mathbf{Stream}_I \to \mathcal{P}(\mathbf{Stream}_O)$ of two functions $f_i : \mathbf{Stream}_{I_i} \to \mathcal{P}(\mathbf{Stream}_{O_i})$ $(i = 1, 2)$ (with $O_1 \cap O_2 = \emptyset$, $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$ and $O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$), linking input and output channels, is defined by $f_1 \otimes f_2(\boldsymbol{s}) \overset{\text{def}}{=} \{\boldsymbol{t}\lfloor_O : \boldsymbol{t} \in \mathbf{Stream}_{I \cup O} \wedge \boldsymbol{t}\lfloor_I = \boldsymbol{s}\lfloor_I \wedge \forall i.\boldsymbol{t}\lfloor_{O_i} \in f_i(\boldsymbol{s}\lfloor_{I_i})\}$.

Interpretation. Again we can only sketch the interpretation, details are in [Jür01b]. Suppose we have a sequence diagram $S = (\mathsf{Objects}, \mathsf{Messages})$. For every $O \in \mathsf{Object}$ we will define a function $\llbracket O \rrbracket : \mathbf{Stream}_{\{\mathsf{in}_O\}} \to \mathcal{P}(\mathbf{Stream}_{\{\mathsf{out}_O\}})$: Let $\mathsf{MsgSpecs}_O$ be the sublist of $\mathsf{MsgSpecs}$ of message specifications with O as sender. Any $\boldsymbol{s} \in \mathbf{Stream}_{\{\mathsf{in}_O\}}$ defines an assignment $\mathsf{ass}(\boldsymbol{s})$ by sending the variable $\mathsf{msg}_{(m,n)}$ to the n^{th} argument of the signal named m received most recently. Let $\llbracket O \rrbracket(\boldsymbol{s})$ be the singleton set consisting of the sequence of those messages $\mathsf{signal}(\mathsf{arguments})$ from $\mathsf{MsgSpecs}_O$ for which $g(\mathsf{ass}(\boldsymbol{s})) = \mathsf{true}$ for the corresponding guard g.

4.2 Secrecy

We say that a stream-processing function $f : \mathbf{Stream}_I \to \mathcal{P}(\mathbf{Stream}_O)$ *may eventually output* an expression $E \in \mathbf{Exp}$ if there exist streams $\boldsymbol{s} \in \mathbf{Stream}_I$ and $\boldsymbol{t} \in f(\boldsymbol{s})$, a channel $c \in O$ and an index $j \in \mathbb{N}$ such that E is an argument of the message $(\boldsymbol{t}(c))_j$.

The following definition uses the notion of an *adversary* from [Jür01c], which is a specific kind of stream-processing function that captures the capabilities of an adversary intercepting the communication link between the distributed objects (the exact definition of "adversary" and "without access" have to be left out here for space limitations but can be found in [Jür01c]).

Definition 3 *We say that a system modeled by a sequence diagram S* leaks *a secret $d \in \mathcal{D} \cup \mathbf{Keys}$ if there is an adversary A without access to d such that $\llbracket S \rrbracket \otimes \llbracket A \rrbracket$ may eventually output d. Otherwise we say that S* preserves the secrecy *of d.*

S preserves the secrecy of d if no adversary can find out d in interaction with the system modeled by S, following the approach of Dolev and Yao (1983), cf. [Aba00,Jür01c].

Example (continued). It has been shown in [Jür01c] that the proposed variant of the TLS handshake protocol given above does not preserve the secrecy of d.

5 Physical View

We give the abstract syntax of (aspects of) deployment models, extended with the tag {high} used to ensure that security requirements on communication links between different components are met by the physical layer.

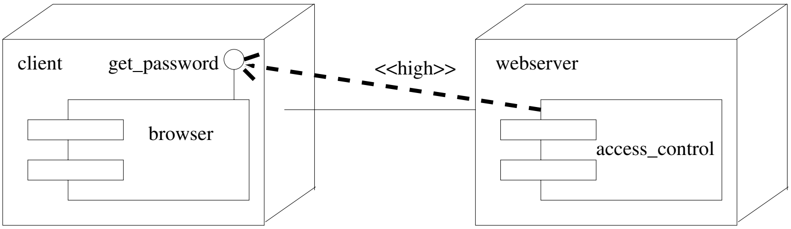
A *component* $C = (\mathsf{comp\_name}, \mathsf{Interfaces})$ is specified by its name comp_name and a (possibly empty) set of Interfaces.

A *node* $N = (\mathsf{node\_name}, \mathsf{Components})$ is given by a name node_name and a set of contained Components.

A *deployment diagram* $D = (\mathsf{Nodes}, \mathsf{Links}, \mathsf{Dependencies})$ is given by a set of Nodes, a set of Links and a set of Dependencies. A *link* $l = (\mathsf{nds}, \mathsf{target})$ is given by a two-element set nds of nodes being linked and a set of tags (that may or may not contain the tag {high} indicating the corresponding security level of the link). Here a *dependency* is a tuple $(\mathsf{client}, \mathsf{supplier}, \mathsf{interface}, \mathsf{tags})$ consisting of component names client and supplier, an interface name interface and a set of tags giving the security level of the dependency. We assume that for every dependency $D = (C, S, I, t)$ there is exactly one link $L_D = (N, t')$ such that $N = \{C, S\}$ for the set of linked nodes.

Definition 4 *A deployment diagram provides* communication security *if for each dependency D that is tagged* {high}*, the corresponding link L_D is also tagged* {high}*.*

Example



This model does not provide communication security, because the communication link between web-server and client does not provide the needed security level.

6 Conclusion and Future Work

The aim of this work is to use UML to encapsulate knowledge on prudent security engineering and to make it available to developers not specialised in security by highlighting aspects of a system design that could give rise to vulnerabilities.

Concentrating on the kinds of diagrams most important for our purpose we showed how UML, by using its extension mechanisms, can be used to express standard concepts from formal methods regarding multi-level secure systems and security protocols. These definitions evaluate diagrams of various kinds and indicate possible weaknesses.

In absence of a general formal semantics we use a specifically defined one which covers the aspects needed and to the required degree of detail. It is not a replacement for a general formal semantics but rather exemplifies the need for one.

In further work [Jür01a], we have used UMLsec to reason about audit-security in a smart-card based payment scheme.

In future work, we will consider the remaining kinds of diagrams and move closer to the unabridged standard of UML by bringing in more detail. In particular, we will explore the use of profiles or prefaces [CKM$^+$99]. Also, we will try to link the various views on a system given by different diagrams to gain additional information on the system that may be security-relevant. To increase usability, we will consider how to automatically derive security annotations from usual modelling information.

To enable tool support, we aim to find efficiently checkable conditions equivalent to the ones given here and to give proof-techniques.

Further case-studies are planned, e. g. regarding development of multilateral security platforms such as [PSW$^+$98].

Acknowledgements. This idea for this work arose when doing security consulting for a project lead by S. Nagaraswany during a research visit with M. Abadi at Bell Labs (Lucent Tech.), Palo Alto, whose hospitality is gratefully acknowledged. This work was presented at the summer school "Foundations of Security Analysis and Design 2000" (Bertinoro), the Computing Laboratory at the University of Oxford, the Department of Computer Science at the TU München, and the Department of Computer Science at the TU Dresden. Comments from S. Abramsky, C. Bolton, M. Broy, J. Davis, G. Lowe, H. Muccini, G. Wimmel, the anonymous referees, and especially A. Pfitzmann, B. Rumpe and P. Stevens are gratefully acknowledged.

References

[Aba00] M. Abadi. Security protocols and their properties. In F.L. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation*, pages 39–60. IOS Press, 2000. 20th Int. Summer School, Marktoberdorf, Germany.

[AJ00] M. Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation, 2000. submitted.

[And94] R. Anderson. Why cryptosystems fail. *Communications of the ACM*, 37(11):32–40, November 1994.

[APS99] V. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How much does it really cost ? In *Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.

[AR00] E. Astesiano and G. Reggio. Formalism and method, 2000. to appear in Theoretical Computer Science.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989.

[BD00] C. Bolton and J. Davies. Using relational and behavioural semantics in the verification of object models. In C. Talcott and S. Smith, editors, *Proceedings of FMOODS*. Kluwer, 2000.

[BGH$^+$98] R. Breu, R. Grosu, F. Huber, B. Rumpe, and W. Schwerin. Systems, views and models of UML. In M. Schader and A. Korthaus, editors, *The Unified Modeling Language, Technical Aspects and Applications*, pages 93–109. Physica Verlag, Heidelberg, 1998.

[BS00] M. Broy and K. Stølen. *Specification and Development of Interactive Systems*. Springer, 2000. (to be published).

[CKM$^+$99] S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, and A. Wills. Defining UML family members using prefaces. In Ch. Mingins and B. Meyer, editors, *TOOLS'99 Pacific*. IEEE Computer Society, 1999.

[DS00] P. Devanbu and S. Stubblebine. Software engineering for security: a roadmap. In *The Future of Software Engineering*, 2000. Special Volume (ICSE 2000).

[EFLR99] A. Evans, R. France, K. Lano, and B. Rumpe. The UML as a formal modeling notation. In J. Bezivin and P.-A. Muller, editors, *The Unified Modeling Language - Workshop UML'98: Beyond the Notation*, LNCS. Springer, 1999.

[For99] UML Revision Task Force. OMG UML Specification 1.3. Available at http : //www.omg.org/uml, 1999.

[GM82] J. Goguen and J. Meseguer. Security policies and security models. In *Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, 1982.

[GPP98] M. Gogolla and F. Parisi-Presicce. State diagrams in UML: A formal semantics using graph transformations. In M. Broy, D. Coleman, T. Maibaum, and B. Rumpe, editors, *PSMT'98*. TU München, TUM-I9803, 1998.

[Huß99] H. Hußmann. Formale Beschreibungstechniken und praktische Softwaretechnik – eine unglückliche Verbindung ? In K. Spies and B. Schätz, editors, *Formale Beschreibungstechniken '99*, pages 1–6. Herbert Utz Verlag, 1999.

[IT95] ITU-T. Z.120 B – Message Sequence Chart Algebraic Semantics. ITU-T, Geneva, 1995.

[Jür00] Jan Jürjens. Secure information flow for concurrent processes. In C. Palamidessi, editor, *CONCUR 2000 (11th International Conference on Concurrency Theory)*, volume 1877 of *LNCS*, pages 395–409, Pennsylvania, 2000. Springer.

[Jür01a] Jan Jürjens. Object-oriented modelling of audit security – a smart-card case study. 2001. submitted.

[Jür01b] Jan Jürjens. *Principles of Secure Systems Design*. PhD thesis, Oxford University Computing Laboratory, 2001. in preparation.

[Jür01c] Jan Jürjens. Secrecy-preserving refinement. In J. Fiadeiro and P. Zave, editors, *Formal Methods Europe*, LNCS. Springer, 2001. to be published.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR. In Margaria and Steffen, editors, *TACAS*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.

[Öve00] G. Övergaard. Formal specification of object-oriented meta-modelling. In *FASE2000*, volume 1783 of *LNCS*. Springer, 2000.

[PSW$^+$98] A. Pfitzmann, A. Schill, A. Westfeld, G. Wicke, G. Wolf, and J. Zöllner. A Java-based distributed platform for multilateral security. In *IFIP/GI Working Conference "Trends in Electronic Commerce"*, volume 1402 of *LNCS*, pages 52–64. Springer, 1998.

[RACH00] G. Reggio, E. Astesiano, C. Choppy, and H. Hußmann. Analysing UML active classes and associated state machines – A lightweight formal approach. In *FASE2000*, volume 1783 of *LNCS*. Springer, 2000.

[RCA00] G. Reggio, M. Cerioli, and E. Astesiano. An algebraic semantics of UML supporting its multiview approach. In D. Heylen, A. Nijholt, and G. Scollo, editors, *AMiLP 2000*, 2000.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[RWW94] A. Roscoe, J. Woodcock, and L. Wulf. Non-interference through determinism. In *ESORICS 94*, volume 875 of *LNCS*. Springer, 1994.

[SP00] P. Stevens and R. Pooley. *Using UML*. Addison-Wesley, 2000.