RISK-DRIVEN DEVELOPMENT OF SECURITY-CRITICAL SYSTEMS USING UMLSEC

Jan Jürjens
Software & Systems Engineering, Dep. of Informatics, TU München, Germany
http://www.jurjens.de/jan – juerjens@in.tum.de

Siv Hilde Houmb
Department of Computer and Information Science, NTNU, Norway
http://www.idi.ntnu.no/ sivhoumb – sivhoumb@idi.ntnu.no

Abstract Despite a growing awareness of security issues in distributed computing systems, most development processes used today still do not take security aspects into account. To address this problem we make use of a risk-driven approach to develop security-critical systems based on UMLsec, the extension of the Unified Modeling Language (UML) for secure systems development, the safety standard ICE 61508, and the concept of model-based risk assessment (MBRA). Security requirements are handled as an integrated part of the development and derived from enterprize information such as security policies, business goals, law and regulation as well as project specific security demands. These are then updated and refined in each iteration of the process and finally refined to security requirements at a technical level, which can be expressed using UMLsec, and analyzed mechanically using the tool-support for UMLsec by referring to a precise semantics of the used fragment of UML.

Keywords: Critical systems development, risk-driven development (RDD), model-based risk assessment (MBRA), model-driven development (MDD)

1. Introduction

Traditionally, in software development projects the focus is put on meeting the end-users' needs in terms of functionality. This has lead to rapidly developed systems with none or little attention to security, and many security-critical systems developed in practise turn out to be insecure. Part of the reason is that most often, security is not an integrated part of the system development process. While functional requirements

are carefully analyzed during system development, non-functional requirements, such as security requirements, are often considered only after the fact. In addition, in practice one has to worry about cost issues and try to achieve an adequate level of security under given time limits and financial constraints.

Lifecycle models and development processes are useful means of describing the various phases of a development project, from the conception of a system to its eventual decommissioning [Lev95]. Several standards exist to guide the development of critical systems, e.g. IEC 61508 [IEC] and the MIL-STD-882B standard [DoD84]. The Australian/New Zealand standard AS/NZS 4360:1999 Risk management [43699] is a general standard targeting risk management. The IST-project CORAS [COR02] is based on the concept of model based risk assessment (MBRA) and has developed an integrated system development and risk management process aiming at security-critical systems. The process is based on AS/NZS 4360, Rational Unified Process (RUP) [Kru99], and the Reference Model for Open Distributed Processes (RM-ODP) [Put00]. The focus is on handling security issues throughout the development process.

In our work we have adapted part of the lifecycle model of IEC 61508 and combined it with the risk management process of AS/NZS4360. Further, we base ourselves on the integrated process of CORAS to support specification of security requirements at an enterprize level, while we use a UML extension for secure systems development, UMLsec [Jür02; Jür03b], to specify security requirements at a technical level, which is then analyzed using tool-support for UMLsec.

This chapter is organized as following. Section 2 presents related work and put the work into context. Section 3 discusses distributed system security, while Section 4 provide a brief description of UMLsec. In Section 5 we discuss security evaluation of UML diagrams and presents the tool supporting security evaluation using UMLsec. Section 6 deals with risk-driven development and provide a brief description of IEC 61508, AS/NZS4360, and the integrated process of CORAS. In Section 7 we present the MBRA development process for security-critical systems, while Section 8 provides an example of how to specify and refine security requirements through the development using the MBRA process. In Section 9, we summarize the main contributions of the chapter.

2. Related Work

There exist a number of specialized risk assessment methodologies for the security domain. Within the domain of health care information systems the British Government's Central Computer and Telecommu-

nication Agency (CCTA) has developed CRAMM [BD92], CCTA risk analysis and management methodology. CRAMM aims at providing a structured and consistent approach to computer management of all systems. The UK National Health Service considers CRAMM to be the standard for risk analysis within systems supporting health care. However, CRAMM is intended for risk analysis of computerized systems in general.

Reactive System Design Support (RSDS) [LAC00] and Surety Analysis [WCF99] are methodologies integrating modelling and risk analysis methods. RSDS is an integrated modelling and risk analysis tool-supported methodology developed by King's College London and B-Core UK, ltd, while Surety Analysis is a method developed in Sandia National Laboratories, a governmental research organization in the U.S. and aims at the modelling and risk analysis of critical and complex systems. These approaches do not however put particular focus on specification, allocation, and verification of security requirements.

E.B. Fernandez and J.C. Hawkins present in [FH97] an extension of use cases and interaction diagrams to develop distributed system architecture requirements. Among other non-functional requirements they introduce questions for requirements elaboration, like system communication load, fault tolerance, safety, real-time deadlines, and security. However, this work is mainly focused on application examples for use cases in security-critical systems, not on giving a methodology for their development or a concept for their integration with domain models. More generally, there are further approaches to a rigorous development of critical systems based on UML, including [PO01; GFR02] (and other articles in [JCF+02]).

3. Distributed System Security

We explain a few important recurring security requirements of distributed object-oriented systems which are encapsulated in UML stereotypes and tags in the UMLsec profile by associating formalizations of these requirements (referring to the formal semantics) as constraints with the stereotypes. The formalizations are obtained following standard approaches to formal security analysis.

Fair exchange When trading goods electronically, the requirement fair exchange postulates that the trade is performed in a way that prevents both parties from cheating. If for example the buyer has to make a prepayment, he should be able to prove having made the payment and to reclaim the money if that good is subsequently not delivered.

Non-repudiation One way of providing fair exchange is by using the security requirement of *non-repudiation* of some action, which means that this action cannot subsequently be denied successfully. That is, the action is *provable*, usually wrt. some trusted third party.

Secure logging For fraud prevention in electronic business transactions, and in particular to ensure non-repudiation, one often makes use of auditing. Here the relevant security requirement represents that the auditing data is, at each point during the transaction of the system, consistent with the actual state of the transaction (to avoid the possibility of fraud by interrupting the transaction).

Guarded access One of the main security mechanisms is access control, which ensures that only legitimate parties have access to a security-relevant part of the system. Sometimes access control is enforced by *guards*.

Secure information flow Where trusted parts of a system interact with untrusted parts, one has to ensure that there is no indirect leakage of sensitive information from a trusted to an untrusted parties. The relevant formal security requirement on the flow of information in the system is called secure information flow. Trusted parts of a system are often marked as "high", untrusted parts as "low".

Secrecy and Integrity Two of the main data security requirements are *secrecy* (or *confidentiality*; meaning that some information can be *read* only by legitimate parties) and *integrity* (some information can be *modified* only by legitimate parties).

Secure communication link Sensitive communication between different parts of a system needs to be protected. The relevant requirement of a *secure communication link* is here assumed to provide secrecy and integrity for the data in transit.

For UMLsec, we give validation rules that evaluate a model with respect to listed security requirements. Many security requirements target the behavior of a system in interaction with its environment and potential adversaries. To verify these requirements, we use the formal semantics defined in Section 5.

4. UMLsec

We recall the fragment of UMLsec needed in our context. More details can be found in [Jür02; Jür03b]. UMLsec allows one to express security-

Stereotype	Base Class	Tags	Constraints	Description
secrecy	dependency			assumes secrecy
integrity	dependency			assumes integrity
critical	object, subsystem	secrecy, integrity		critical object
secure links	subsystem		dependency security matched by links	enforces secure communication links
secure dependency	subsystem		« call », « send » respect data security	structural interaction data security
data security	subsystem		provides data sec.	basic datasec requirements
fair exchange	subsystem	start,stop	after start eventually reach stop	enforce fair exchange

Figure 1. Some UMLsec stereotypes

related information within the diagrams in a UML system specification. The extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate security requirements and assumptions on the system environment; *constraints* give criteria that determine whether the requirements are met by the system design.

Stereotypes define new types of modelling elements extending the semantics of existing types or classes in the UML metamodel. Their notation consists of the name of the stereotype written in double angle brackets « », attached to the extended model element. This model element is then interpreted according to the meaning ascribed to the stereotype.

One way of explicitly defining a property is by attaching a *tagged value* to a model element. A tagged value is a name-value pair, where the name is referred to as the *tag*. The corresponding notation is {*tag*=*value*} with the tag name tag and a corresponding *value* to be assigned to the tag.

Another way of adding information to a model element is by attaching *Constraints* to refine its semantics. Stereotypes can be used to attach tagged values and constraints as pseudo-attributes of the stereotyped model elements.

In Figure 1 we give the relevant fragment of the list of stereotypes from UMLsec, together with their tags and constraints.

We shortly explain the use of the stereotypes and tags given in Figure 1. More information can be found in [Jür02; Jür03b].

critical This stereotype labels objects that are critical in some way, which is specified in more detail using the corresponding tags. The tags are **{secrecy}** and **{integrity}**. The values of the first two are the names of expressions or variables (that is, attributes or mes-

sage arguments) of the current object the secrecy (resp. integrity) of which is supposed to be protected.

secure links This stereotype on subsystems containing deployment diagrams is used to ensure that security requirements on the communication are met by the physical layer.

secure dependency This stereotype on subsystems containing static structure diagrams ensures that the «call» and «send» dependencies between objects or subsystems respect the security requirements on the data that may be communicated across them, as given by the tags {secrecy} and {integrity} of the stereotype «critical».

fair exchange This stereotype of (instances of) subsystems has associated tags *start* and *stop* taking names of states as values. The associated constraint requires that, whenever a *start* state in the contained activity diagram is reached, then eventually a *stop* state will be reached.

5. Security evaluation of UML diagrams using formal semantics

For some of the constraints used to define the UMLsec extensions we need to refer to a precisely defined semantics of behavioral aspects, because verifying whether they hold for a given UML model may be mathematically non-trivial. Firstly, the semantics is used to define these constraints in a mathematically precise way. Secondly, in ongoing work, we are developing mechanical tool support for analyzing UML specifications (for example in [Sha03; Men], and a few other student projects). For this, a precise definition of the meaning of the specifications is necessary, and it is useful to formulate this as a formal model for future reference before coding it up. For security analysis, the security-relevant information from the security-oriented stereotypes is then incorporated.

Note that because of the complexities of the UML, it would take up too much space to recall our formal semantics here completely. Instead, we just define precisely and explain the interfaces of the semantics that we need here to define the UMLsec profile. More details on the formal semantics can be found in [Jür03b]. Our formal semantics of a simplified fragment of UML using Abstract State Machines (ASMs) includes the following kinds of diagrams:

Class diagrams define the static class structure of the system: classes with attributes, operations, and signals and relationships between classes. On the instance level, the corresponding diagrams are called *object diagrams*.

Statechart diagrams (or *state diagrams*) give the dynamic behavior of an individual object or component: events may cause a change in state or an execution of actions.

Sequence diagrams describe interaction between objects or system components via message exchange.

Activity diagrams specify the control flow between several components within the system, usually at a higher degree of abstraction than statecharts and sequence diagrams. They can be used to put objects or components in the context of overall system behavior or to explain use cases in more detail.

Deployment diagrams describe the physical layer on which the system is to be implemented.

Subsystems (a certain kind of *packages*) integrate the information between the different kinds of diagrams and between different parts of the system specification.

There is another kind of diagrams, the use case diagrams, which describe typical interactions between a user and a computer system. They are often used in an informal way for negotiation with a customer before a system is designed. We will not use it in the following. Additionally to sequence diagrams, there are *collaboration diagrams*, which present similar information. Also, there are *component diagrams*, presenting part of the information contained in deployment diagrams.

The used fragment of UML is simplified significantly to keep a formal treatment that is necessary for some of the more subtle security requirements feasible and to allow model-checking of UML specifications. Note also that in our approach we identify system objects with UML objects, which is suitable for our purposes. Also, as with practical all analysis methods, also in the real-time setting [Wat02], we are mainly concerned with instance-based models.

Although simplified, our choice of a subset of UML is reasonable for our needs, as we have demonstrated in several industrial case-studies (some of which are documented in [Jür03b]).

The formal semantics for subsystems incorporates the formal semantics of the diagrams contained in a subsystem. Although restricted in several ways (see [Jür03b]) any one time an object's behavior is represented by only one diagram in the formal semantics

- models actions and internal activities explicitly (rather than treating them as atomic given events), in particular the operations and the parameters employed in them,

- provides passing of messages with their parameters between objects or components specified in different diagrams, including a dispatching mechanism for events and the handling of actions, and thus
- allow in principle whole specification documents to be based on a formal foundation.

In particular, we can compose subsystems by including them into other subsystems. It prepares the ground for the tool-support based on this precise semantics.

Objects, and more generally system components, can communicate by exchanging messages. These consist of the message name, and possibly arguments to the message, which will be assumed to be elements of the set. Message names may be prefixed with object or subsystem instance names. Each object or component may receive messages received in an input queue and release messages to an output queue.

In our model, every object or subsystem instance O has associated multi-sets inQu_O and outQu_O (*event queues*). Then our formal semantics models sending a message $msg = op(exp_1, \ldots, exp_n) \in \mathbf{Events}$ from an object or subsystem instance S to an object or subsystem instance R as follows:

(1) S places the message $R.msg$ into its multi-set outQu_S.

(2) A scheduler distributes the messages from out-queues to the intended in-queues (while removing the message head); in particular, $R.msg$ is removed from outQu_S and msg added to inQu_R.

(3) R removes msg from its in-queue and processes its content.

In the case of operation calls, we also need to keep track of the sender to allow sending return signals. This way of modelling communication allows for a very flexible treatment; for example, we can modify the behavior of the scheduler to take account of knowledge on the underlying communication layer.

At the level of single objects, behavior is modelled using statecharts, or (in special cases such as protocols) possibly as using sequence diagrams. The internal activities contained at states of these statecharts can again be defined each as a statechart, or alternatively, they can be defined directly using ASMs.

Using subsystems, one can then define the behavior of a system component C by including the behavior of each of the objects or components directly contained in C, and by including an activity diagram that coordinates the respective activities of the various components and objects.

Thus for each object or component C of a given system, our semantics defines a function $[\![C]\!]()$ which

- takes a multi-set I of input messages and a component state S and
- outputs a set $[\![C]\!](I, S)$ of pairs (O, T) where O is a multi-set of output messages and T the new component state (it is a *set* of pairs because of the non-determinism that may arise)

together with an *initial state* S_0 of the component.

Specifically, the behavioral semantics $[\![D]\!]()$ of a statechart diagram D models the run-to-completion semantics of UML statecharts. As a special case, this gives us the semantics for activity diagrams. Any sequence diagram $\mathcal{S}$ gives us the behavior $[\![\mathcal{S}.C]\!]()$ of each contained component C.

Subsystems group together diagrams describing different parts of a system: a system component $\mathcal{C}$ given by a subsystem $\mathcal{S}$ may contain subcomponents $\mathcal{C}_1, \ldots, \mathcal{C}_n$. The behavioral interpretation $[\![\mathcal{S}]\!]()$ of $\mathcal{S}$ is defined as follows:

(1) It takes a multi-set of input events.

(2) The events are distributed from the input multi-set and the link queues connecting the subcomponents and given as arguments to the functions defining the behavior of the intended recipients in $\mathcal{S}$.

(3) The output messages from these functions are distributed to the link queues of the links connecting the sender of a message to the receiver, or given as the output from $[\![\mathcal{S}]\!]()$ when the receiver is not part of $\mathcal{S}$.

When performing security analysis, after the last step, the adversary model may modify the contents of the link queues in a certain way, which is explained in the next section.

5.1. Security analysis of UML diagrams

Our modular UML semantics allows a rather natural modelling of potential adversary behavior. We can model specific types of adversaries that can attack different parts of the system in a specified way. For example, an attacker of type *insider* may be able to intercept the communication links in a company-wide local area network. We model the actual behavior of the adversary by defining a class of ASMs that can access the communication links of the system in a specified way. To evaluate the security of the system with respect to the given type of adversary, we consider the joint execution of the system with any ASM

in this class. This way of reasoning allows an intuitive formulation of many security properties. Since the actual verification is rather indirect this way, we also give alternative intrinsic ways of defining security properties below, which are more manageable, and show that they are equivalent to the earlier ones.

Thus for a security analysis of a given UMLsec subsystem specification $\mathcal{S}$, we need to model potential adversary behavior. We model specific types of adversaries that can attack different parts of the system in a specified way. For this we assume a function $\mathsf{Threats}_A(s)$ which takes an *adversary type* A and a stereotype s and returns a subset of $\{\mathsf{delete}, \mathsf{read}, \mathsf{insert}, \mathsf{access}\}$ (*abstract threats*). These functions arise from the specification of the physical layer of the system under consideration using deployment diagrams, as explained in Sect. 4. For a link l in a deployment diagram in $\mathcal{S}$, we then define the set $\mathsf{threats}^{\mathcal{S}}_A(l)$ of *concrete threats* to be the smallest set satisfying the following conditions:

If each node n that l is contained in[1] carries a stereotype s_n with $\mathsf{access} \in \mathsf{Threats}_A(s_n)$ then:

- If l carries a stereotype s with $\mathsf{delete} \in \mathsf{Threats}_A(s)$ then $\mathsf{delete} \in \mathsf{threats}^{\mathcal{S}}_A(l)$.
- If l carries a stereotype s with $\mathsf{insert} \in \mathsf{Threats}_A(s)$ then $\mathsf{insert} \in \mathsf{threats}^{\mathcal{S}}_A(l)$.
- If l carries a stereotype s with $\mathsf{read} \in \mathsf{Threats}_A(s)$ then $\mathsf{read} \in \mathsf{threats}^{\mathcal{S}}_A(l)$.
- If l is connected to a node that carries a stereotype t with $\mathsf{access} \in \mathsf{Threats}_A(t)$ then $\{\mathsf{delete}, \mathsf{insert}, \mathsf{read}\} \subseteq \mathsf{threats}^{\mathcal{S}}_A(l)$.

The idea is that $\mathsf{threats}^{\mathcal{A}}_A(x)$ specifies the *threat scenario* against a component or link x in the ASM system $\mathcal{A}$ that is associated with an adversary type A. On the one hand, the threat scenario determines, which data the adversary can obtain by *accessing* components, on the other hand, it determines, which actions the adversary is permitted by the threat scenario to apply to the concerned links. delete means that the adversary may delete the messages in the corresponding link queue, read allows him to read the messages in the link queue, and insert allows him to insert messages in the link queue.

Then we model the actual behavior of an adversary of type A as a *type A adversary machine*. This is a state machine which has the following data:

[1]Note that nodes and subsystems may be nested one in another.

- a control state $\mathsf{control} \in \mathsf{State}$,
- a set of *current adversary knowledge* $\mathcal{K} \subseteq \mathbf{Exp}$, and
- for each possible control state $c \in \mathsf{State}$ and set of knowledge $K \subseteq \mathbf{Exp}$, we have
  - a set $\mathsf{Delete}_{c,K}$ which may contain the name of any link l with $\mathsf{delete} \in \mathsf{threats}_A^{\mathcal{S}}(l)$
  - a set $\mathsf{Insert}_{c,K}$ which may contain any pair (l, E) where l is the name of a link with $\mathsf{insert} \in \mathsf{threats}_A^{\mathcal{S}}(l)$, and $E \in \mathcal{K}$, and
  - a set $\mathsf{newState}_{c,k} \subseteq \mathsf{State}$ of states.

The machine is executed from a specified initial state $\mathsf{control} := \mathsf{control}_0$ with a specified *initial knowledge* $\mathcal{K} := \mathcal{K}_0$ iteratively, where each iteration proceeds according to the following steps:

(1) The contents of all link queues belonging to a link l with $\mathsf{read} \in \mathsf{threats}_A^{\mathcal{S}}(l)$ are added to $\mathcal{K}$.

(2) The content of any link queue belonging to a link $l \in \mathsf{Delete}_{\mathsf{control},\mathcal{K}}$ is mapped to $\emptyset$, .

(3) The content of any link queue belonging to a link l is enlarged with all expressions E where $(l, E) \in \mathsf{Insert}_{\mathsf{control},\mathcal{K}}$.

(4) The next control state is chosen non-deterministically from the set $\mathsf{newState}_{\mathsf{control},\mathcal{K}}$.

The set $\mathcal{K}_0$ of initial knowledge contains all data values v given in the UML specification under consideration for which each node n containing v carries a stereotype s_n with $\mathsf{access} \in \mathsf{Threats}_A(s_n)$. In a given situation, $\mathcal{K}_0$ may also be specified to contain additional data (for example, public encryption keys).

Note that an adversary A able to remove all values sent over the link l (that it, $\mathsf{delete}_l \in \mathsf{threats}_A^{\mathcal{S}}(l)$) may not be able to selectively remove a value e with known meaning from l: For example, the messages sent over the Internet within a virtual private network are encrypted. Thus, an adversary who is unable to break the encryption may be able to delete all messages undiscrimatorily, but not a single message whose meaning would be known to him.

To evaluate the security of the system with respect to the given type of adversary, we then define the *execution of the subsystem $\mathcal{S}$ in presence of an adversary of type A* to be the function $[\![\mathcal{S}]\!]_A()$ defined from $[\![\mathcal{S}]\!]()$

by applying the modifications from the adversary machine to the link queues as a fourth step in the definition of $[\![\mathcal{S}]\!]()$ as follows:

(4) The type A adversary machine is applied to the link queues as detailed above.

Thus after each iteration of the system execution, the adversary may non-deterministically change the contents of link queues in a way depending on the level of physical security as described in the deployment diagram (see Sect. 4).

There are results which simplify the analysis of the adversary behavior defined above, which are useful for developing mechanical tool support, for example to check whether the security properties secrecy and integrity (see below) are provided by a given specification. These are beyond the scope of the current paper and can be found in [Jür03b].

One possibility to specify security requirements is to define an idealized system model where the required security property evidently holds (for example, because all links and components are guaranteed to be secure by the physical layer specified in the deployment diagram), and to prove that the system model under consideration is behaviorally equivalent to the idealized one, using a notion of behavioral equivalence of UML models. This is explained in detail in [Jür03b].

In the following subsection, we consider alternative ways of specifying the important security properties secrecy and integrity which do not require one to explicitly construct such an idealized system and which are used in the remaining parts of this paper.

5.2. Important security properties

The formal definition of the two main security properties secrecy and integrity considered in this section follow the standard approach of [DY83] and are defined in an intuitive way by incorporating the attacker model.

Secrecy The formalization of secrecy used in the following relies on the idea that a process specification preserves the secrecy of a piece of data d if the process never sends out any information from which d could be derived, even in interaction with an adversary. More precisely, d is leaked if there is an adversary of the type arising from the given threat scenario that does not initially know d and an input sequence to the system such that after the execution of the system given the input in presence of the adversary, the adversary knows d (where "knowledge", "execution" etc. have to be formalized). Otherwise, d is said to be kept secret.

Thus we come to the following definition.

Definition 1 *We say that a subsystem* $\mathcal{S}$ preserves the secrecy *of an expression* E *from adversaries of type* A *if* E *never appears in the knowledge set* $\mathcal{K}$ *of* A *during execution of* $[\![\mathcal{S}]\!]_A()$.

This definition is especially convenient to verify if one can give an upper bound for the set of knowledge $\mathcal{K}$, which is often possible when the security-relevant part of the specification of the system $\mathcal{S}$ is given as a sequence of command schemata of the form *await event e – check condition g – output event e'* (for example when using UML sequence diagrams or statecharts for specifying security protocols, see Sect. 4).

Examples.

- The system that sends the expression $\{m\}_K :: K \in \mathbf{Exp}$ over an unprotected Internet link does not preserve the secrecy of m or K against attackers eavesdropping on the Internet, but the system that sends $\{m\}_K$ (and nothing else) does, assuming that it preserves the secrecy of K against attackers eavesdropping on the Internet.

- The system that receives a key K encrypted with its public key over a dedicated communication link and sends back $\{m\}_K$ over the link does not preserve the secrecy of m against attackers eavesdropping on and inserting messages on the link, but does so against attackers that cannot insert messages on the link.

Integrity The property integrity can be formalized similarly: If during the execution of the considered system, a system variable gets assigned a value initially only known to the adversary, then the adversary must have caused this variable to contain the value. In that sense the integrity of the variable is violated. (Note that with this definition, integrity is also viewed as violated if the adversary as an honest participant in the interaction is able to change the value, so the definition may have to be adapted in certain circumstances; this is, however, typical for formalizations of security properties.) Thus we say that a system preserves the integrity of a variable v if there is no adversary A such that at some point during the execution of the system with A, v has a value i_0 that is initially known only to A.

Definition 2 *We say that a subsystem* $\mathcal{S}$ preserves the integrity *of an attribute* a *from adversaries of type* A *with initial knowledge* $\mathcal{K}_0$ *if during*

execution of $[\![\mathcal{S}]\!]_A()$*, the attribute* a *never takes on a value appearing in* $\mathcal{K}_0$ *but not in the specification* $\mathcal{S}$*.*

The idea of this definition is that $\mathcal{S}$ preserves the integrity of a if no adversary can make a take on a value initially only known to him, in interaction with $\mathcal{A}$. Intuitively, it is the "opposite" of secrecy, in the sense that secrecy prevents the flow of information from protected sources to untrusted recipients, while integrity prevents the flow of information in the other direction. Again, it is a relatively simple definition, which may however not prevent implicit flows of information.

5.3. Tool support

Security validation in our approach is performed through mechanical analysis that validates the fulfilment of the constraints of the security requirements, as those associated with the stereotypes defined in Section 4. A first version has been demonstrated at [Jür03a]. The tool works with UML 1.4 models, which can be stored in a XMI 1.2 (XML Metadata Interchange) format by a number of existing UML design tools. To avoid processing UML models directly on the XMI level, the MDR (Meta-Data Repository, http://mdr.netbeans.org) is used, which allows one to operate directly on the UML concept level (as used by e.g. the UML CASE tool Poseidon, http://www.gentleware.com). The MDR library implements repository for any model described by a modelling language compliant to the MOF (Meta Object Facility).

Figure 2 illustrates the functionality of the tool. The developer creates a model and stores it in the UML 1.4 / XMI 1.2 file format. The file is imported by the tool into the internal MDR repository. The tool accesses the model through the JMI interfaces generated by the MDR library. The checker parses the model and checks the constraints associated with the stereotype. The results are delivered as a text report for the developer describing found problems, and a modified UML model, where the stereotypes whose constraints are violated are highlighted.

6. Risk-Driven Development

In the following, we give a brief introduction to the lifecycle of IEC 61508, the principle of AS/NZS4360, and the work of CORAS which we base our MBRA development approach on. Risk-driven development is risk-driven in that it focusses on assessing risks and proposing treatments throughout a set of activities. We assume that functional requirements are handled as part of the development and focus on security requirements and the allocation of security requirements in this section.

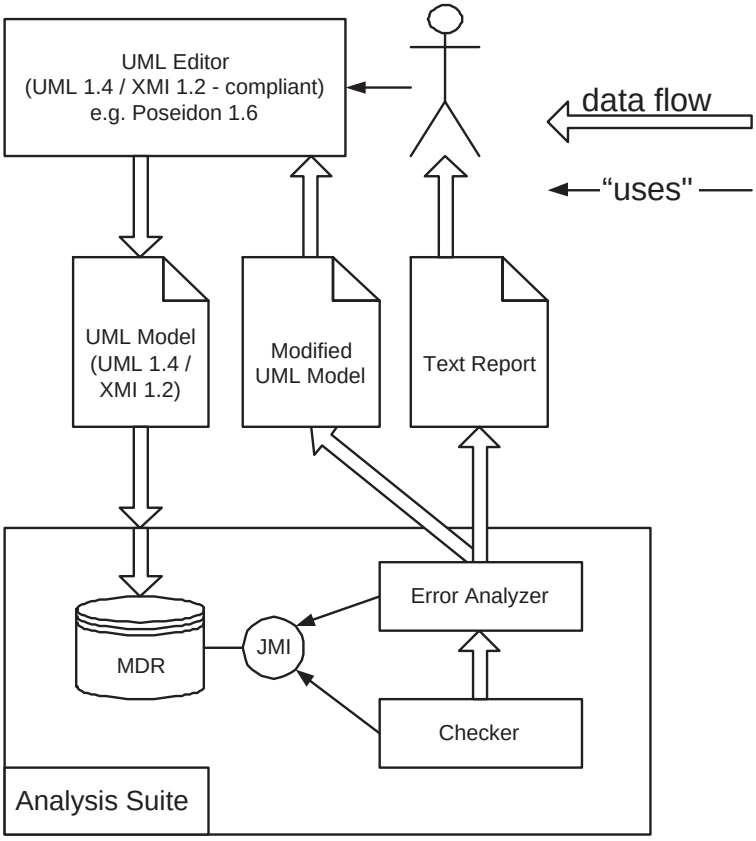

Figure 2. The UMLsec analysis tool

6.1. IEC 61508

The IEC standard IEC 61508 (Functional safety of electrical/electronic/ programmable electronic safety-related systems) [IEC] covers important aspects that need to be addressed when electrical, electronic, and programmable devices are used in connection with safety functions. The strategy of the standard is to derive safety requirements from a hazard and risk analysis and to design the system to meet those safety requirements, taking all possible causes of failure into account. The essence is that all activities relating to functional safety are managed in a planned and methodical way, with each phase having defined inputs and outputs [Bro00]. The standard considers all phases in a safety lifecycle, from initial concept, through design, implementation, operation and maintenance to decommissioning. Figure 3 depicts the lifecycle model of IEC 61508.

IEC 61508 applies to any safety-related software in which is implemented as the aforesaid solutions. This includes: (a) software that is part of a safety-related system; (b) software that is used to develop a safety-related system; and (c) the operating system, system software, communication software, human computer interface (HCI) functions, utilities, and software engineering tools used with (a) or (b).

The process consists of the following phases:

(1) **Concept:** An understanding of the system and its environment is developed.

(2) **Overall scope definition:** The boundaries of the system and its environment are determined, and the scope of the hazard and risk analysis is specified.

(3) **Hazard and risk analysis:** Hazards and hazardous events of the system, the event sequences leading to the hazardous events, and the risks associated with the hazardous events are determined.

(4) **Overall safety requirements:** The specification for the overall safety requirements is developed in order to achieve the required functional safety.

(5) **Safety requirements allocation:** The safety functions contained in the overall safety requirements specification are allocated to the safety-related system, and a safety integrity level is allocated to each safety function.

(6) **Overall operation and maintenance planning:** A plan is developed for operating and maintaining the system, and the required

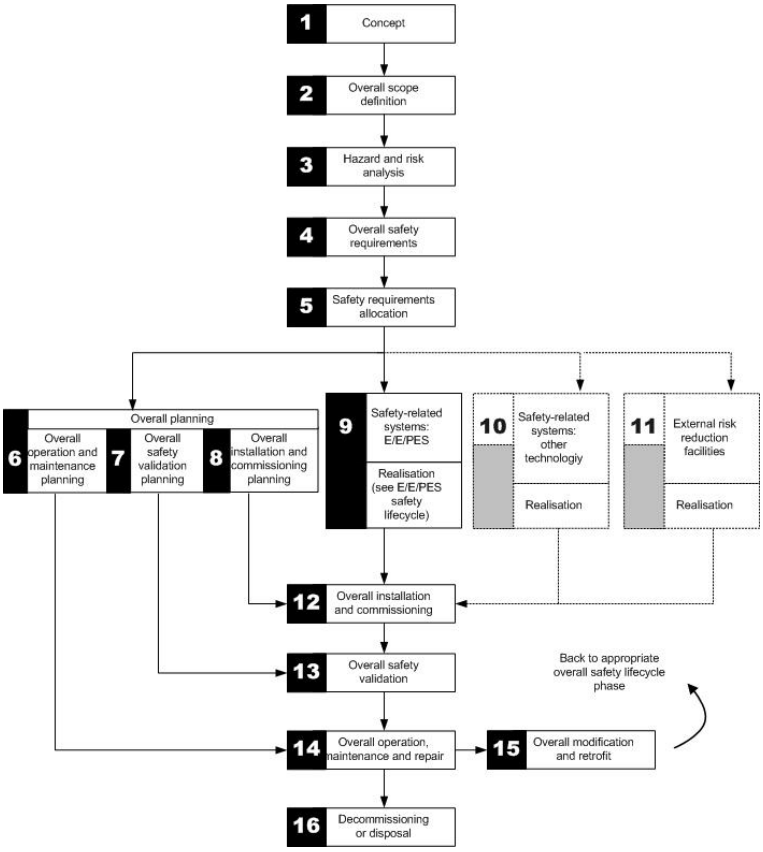

Figure 3. Overall safety lifecycle of IEC 61508

functional safety is ensured to be maintained during operation and maintenance.

(7) **Overall safety validation planning:** A plan for the overall safety validation of the system is developed.

(8) **Overall installation and commissioning planning:** Plans, ensuring that the required functional safety are achieved, are developed for the installation and commissioning of the system.

(9) **Safety-related systems:** The Electrical, Electronic and Programmable Electronic Systems (E/E/PES) safety-related system is created conforming to the safety requirements specification.

(10) **Safety-related systems (other technology):** Safety-related systems based on other technology are created to meet the requirements specified for such systems (outside scope of the standard).

(11) **External risk reduction facilities:** External risk reduction facilities are created to meet the requirements specified for such facilities (outside scope of the standard).

(12) **Overall installation and commissioning:** The Electrical, Electronic and Programmable Electronic Systems (E/E/PES) safety-related system is installed and commissioned.

(13) **Overall safety validation:** The Electrical, Electronic and Programmable Electronic Systems (E/E/PES) safety-related system is validated to meet the overall safety requirements specification.

(14) **Overall operation, maintenance and repair:** The system is operated, maintained and repaired in order to ensure that the required functional safety is maintained.

(15) **Overall modification and retrofit:** The functional safety of the system is ensured to be appropriate both during and after modification and retrofit.

(16) **Decommissioning or disposal:** The functional safety of the system is ensured to be appropriate during and after decommissioning or disposing of the system.

6.2. Model-based risk assessment: The CORAS approach

Model-based risk assessment (MBRA) has been a research topic since the early 80-ies [KM87; GO84] and builds on the concept of applying system modelling when specifying and describing the systems to be assessed as an integrated part of the risk assessment. The CORAS framework is based on the concept of MBRA and employs modelling methodology for three main purposes: (1) To describe the target of evaluation at the right level of abstraction, (2) As a medium for communication and interaction between different groups of stakeholders involved in a risk assessment, and (3) To document risk assessment results and the assumptions on which these results depend.

Figure 4 outlines the sub-processes and activities contained in the CORAS risk management process, which is a refinement of AS/NZS 4360:1999. Further information on the CORAS risk management process can be found in [HdBLS02].

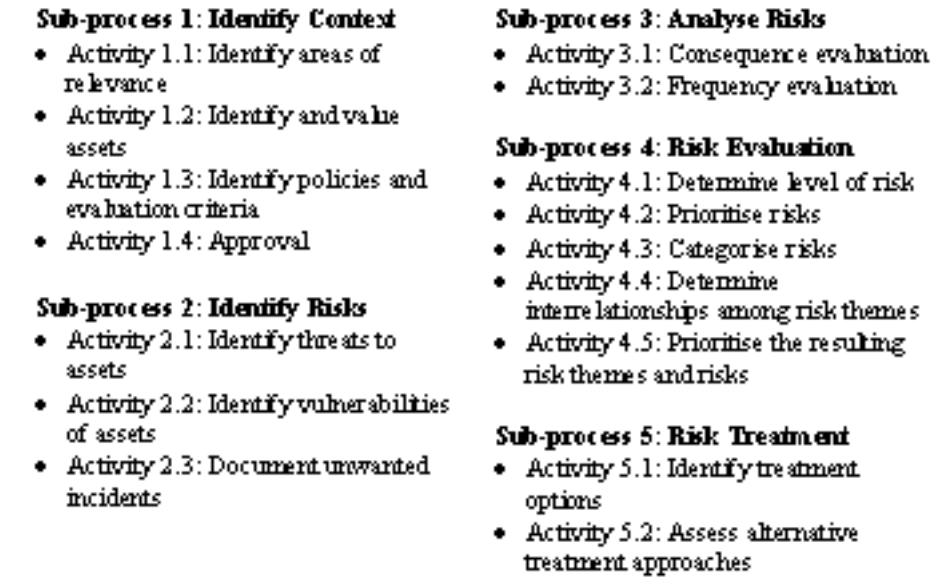

Figure 4. Sub-processes and activities in the CORAS risk management process [HdBLS02]

The integrated system development and risk management process of CORAS is based on the CORAS risk management process, the Reference Model - Open Distributed Processing (RM-ODP), and the Rational Unified Process (RUP). RUP structures system development according to four phases: (1) Inception, (2) Elaboration, (3) Construction, and (4) Transition. As illustrated in Figure 5, these two processes are combined in order to address security throughout the development. In each iteration in the development one assesses a particular part of the system or the whole system at a particular viewpoint according to RM-ODP. For each of the iterations, treatments are evaluated and proposed according to a cost-benefit strategy.

7. MBRA Development Process for Security-Critical Systems

In system development, one usually distinguishes between three levels of abstractions: the requirement specification, the design specification, and the implementation. The design specification is thus a refinement of the requirement specification, and the implementation is a refinement of the design specification.

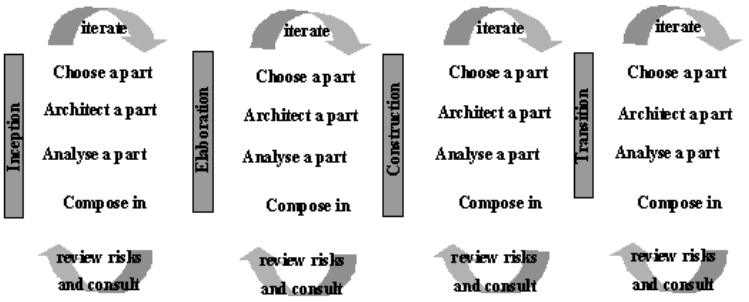

Figure 5. The integrated system development and risk management process

In the MBRA development process for security-critical systems (see Figure 6), we make use of the engineering and technical experience gained when developing safety critical systems within the process industry. The development process is based on the concept of handling safety requirements in IEC 61508 and the idea of using models both to document the system and as input to risk management from the CORAS integrated risk management and system development process. The process is both stepwise iterative and incremental. For each iteration more information is added and increasingly detailed versions of the system are constructed through subsequent iterations.

The first two phases concern the specification of concepts and overall scope definition of the system. A system description and a functional requirements proposition are the results of these two phases. This information is further used as input to the preliminary hazard analysis (PHA). By performing a PHA early in the development process, the most obvious and conspicuous potential hazards can be identified and handled more easily and at a lower cost. Furthermore, the PHA aids in the elicitation of security requirements for the system, which is the fourth phase in the development process. Based on the security policy for the involved organizations, security requirements are specified first on the enterprize level and then refined into more technical specifications using UMLsec. Phase 4 targets the identification of security threats using e.g. Security-HazOp [Vog01]. Security threats are then analyzed in terms of finding the frequency of occurrence and potential impacts of the threats. Based on the results from risk analysis, risks are evaluated and either accepted or not accepted. Unacceptable risks are treated by refining or by specifying new security requirements or by introducing safeguards before the risk management step is iterated until no unacceptable risks remain. When the required security level is achieved, the

system is implemented and tested. If the implemented version is not approved, the process is reiterated from the risk management step. The whole process is iterated from phase 1 whenever updating the system description or the functional requirements.

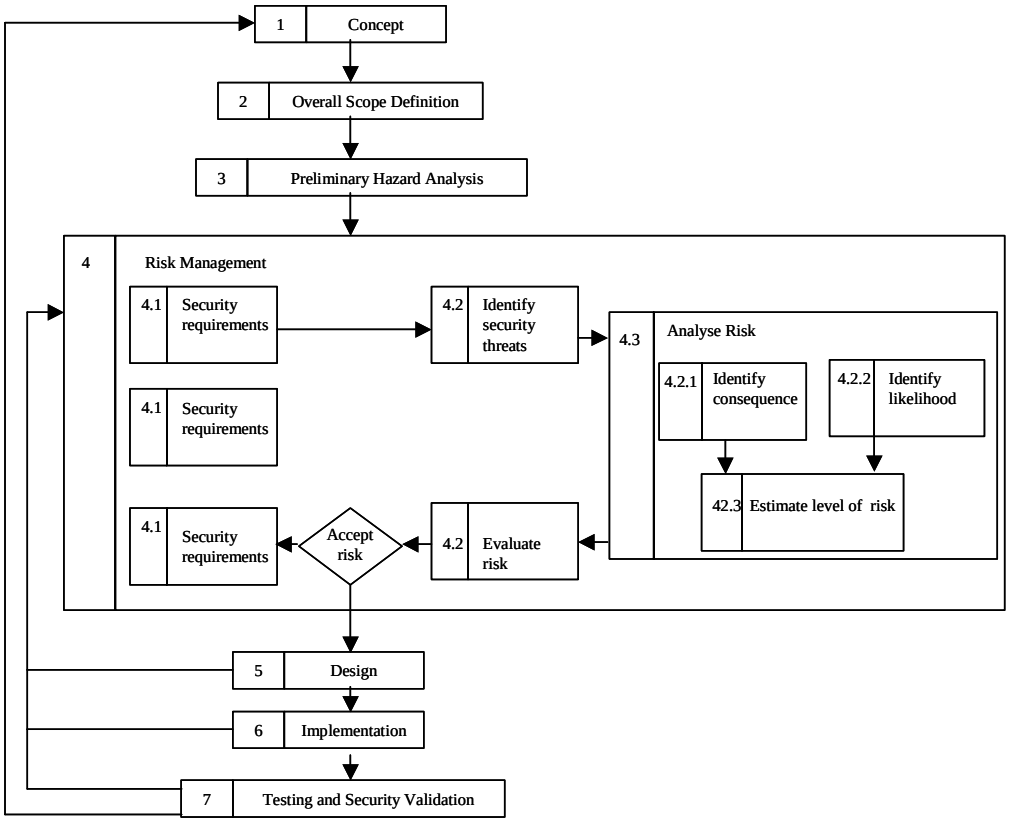

Figure 6. Development process for security-critical systems

8. Development of a AIBO-Lego Mindstorm Prototype Using the Approach

In this Section, we will illustrate the use and applicability of the risk-driven development process for security-critical systems using a AIBO-Lego Mindstorm prototype system. The system is used as a medium of teaching the effect of handling security and safety as an integrated part of development and to test the applicability of techniques and ap-

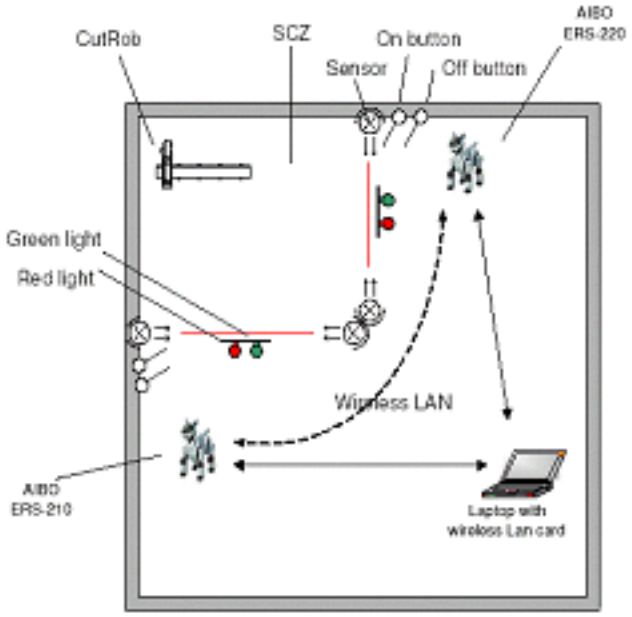

Figure 7. Illustration of the prototype system

proaches for development of security-critical systems at the Norwegian University of Science and Technology (NTNU), Norway. The prototype was developed as part of a Master Thesis at NTNU [Sør02] and consists of an prototypical industrial robot and a computerized control and monitoring system. The control system is implemented using Lego Mindstorm and the monitoring system is implemented using Sony AIBO robots as the monitoring system and a PC-controller (portable computer with software) representing the control system. AIBO and PC-controller communicate using WLAN and TCP/IP as depicted in Figure 7.

8.1. Concept and overall scope definition

Concept and overall scope definition constitute the first two phases of the process. The main objective of these two phases is to define the main objective and purpose of the system. The main objective of the Lego-AIBO system is to develop a prototype to investigate the relationship between security threats and safety consequences in a safety critical system that make use of computerized monitoring and control systems. However, in this context we will only look into the security aspects of the computerized monitoring and control system. The main objective

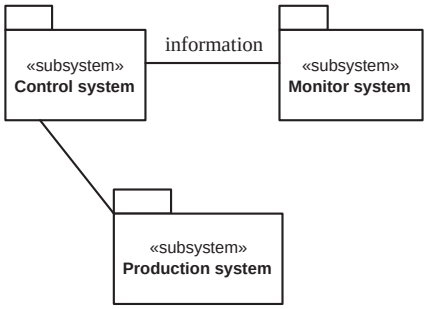

Figure 8. The main components of the AIBO-Lego prototype

of these two systems is to monitor all access to the safety zone of the system and prevent unauthorized access to the zone.

The AIBO-Lego prototype system consists of three components, the monitoring system, the control system, and the production system as depicted in Figure 8. The control system receives information from the AIBO (monitoring system), processes this information, and sends instructions to the production system based on the information provided by the AIBO. The main functionality of the interface between the monitoring and the control system, represented by the AIBO and the PC-controller, is to send and receive information as illustrated in Figure 9.

8.2. Preliminary hazard analysis (PHA)

When the purpose and scope of the system has been established, a preliminary hazard analysis is performed. In this phase, we use Security-HazOp as described in [GWJ01] to identify overall security threats to the system. However, due to space restrictions we will only focus on security threats related to the communication between the monitoring

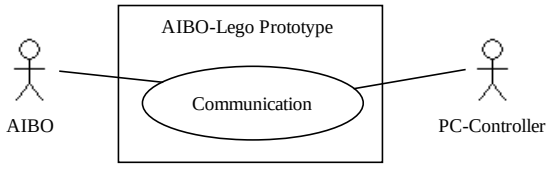

Figure 9. Overview of the main functionality between the AIBO and PC-controller

Pre-Guideword	Attribute	Components	Post-Guideword
Deliberate	Disclosure	AIBO	Insider
Unintentional	Manipulation	PC-controller	Outsider
	Denial	Communication	Technical failure
	Fabrication		Virus
	Delay		Physical blocking

Figure 10. Combination of guidewords used for PHA

and control system. The reader is referred to Chapter 9 in [Sør02] for more information on the result of the PHA.

Security-HazOp is an adaptation of the safety analysis method HazOp (Hazard and Operability Analysis) for Security-Critical systems. Security-HazOp make use of the negation of the security attributes as part of the guidewords. Guidewords, in HazOp, is used to guide the brainstorming process when identifying security threats. The reader is referred to [Lev95] for more information on HazOp. Security-HazOp is performed as a brainstorming session using different combinations of sentences of the form:

Pre-Guideword Attribute *of* **Component** *due to* **Post-Guideword**.

Figure 10 depicts the combination of guidewords used for PHA. Pre-guideword denotes whether the attack is intentional or not, while attribute is the negation of the security attributes secrecy, integrity, and availability. Components denotes the components that are analyzed and the post-guideword relate to the threat agent who is responsible for the attack.

As input to PHA in a risk-driven development, we use UML diagrams describing the main functionality of the system. These diagrams are called PHA input diagrams. Figure 11 provide an example of a PHA input diagram. PHA input diagrams could be any type of UML diagram, however, since we are mainly concerned with information flow and behavior in Security-Critical systems, one usually uses one or several of the UML behavioral diagrams. Figure 11 depicts a PHA input diagram modelled as a UML sequence diagram. The diagram specifies the main interface between the control and monitoring system in the AIBO-Lego prototype.

When using UML models as input to PHA or other risk analysis methods one goes through each diagram using a set of guidelines. These guidelines specify two things: Firstly, the information provided by the specific UML diagram that should be used as input, and secondly, how

to use the information as input to risk analysis methods. The risk analysis methods supported are HazOp (Hazard and Operability analysis), FME(C)A (Failure Mode, Effect, and Criticality Analysis, and FTA (Fault Three Analysis). Currently, all UML 1.4 diagrams are supported by the guidelines (will be updated to support UML 2.0 when finalized). As an example we will describe the guideline for using UML sequence diagrams as input to HazOp. The reader is referred to [GSD+03] for more information on the guidelines.

HazOp is organized as structured brainstorming using a group of experts. The brainstorming meetings consist of a set of experts, a risk analysis leader, and a risk analysis secretary. The risk analysis leader goes through the set of guidelines as already explained, while the secretary records the result from the brainstorming during the meeting. The result from the brainstorming is recorded in a HazOp table, as illustrated in Figure 12. The columns *Pre-Guideword*, *Attribute*, and *Post-Guideword* are the same as described in Figure 10. The column *ID* is used to assign a unique id to the threat scenario, while the column *Asset* denotes the information from the UML diagram being analyzed. For the guideline for use of UML sequence diagrams as input to HazOp, assets are represented as either messages or objects. Generally, an asset is something of value to one or more stakeholders and can be anything from a particular piece of information to a physical computer or other equipment. Assets are typically derived from requirement specifications. For more information on how to identify assets, see [GSD+03]. The column *Component* denotes the part of the system the asset is part of or connected to. In the case of the example, we are looking at the communication between the AIBO and the PC-Controller. The column *Threat* describes the event that may happen. In the example, the threat is derived from combining pre-guideword, attribute, asset, and components,

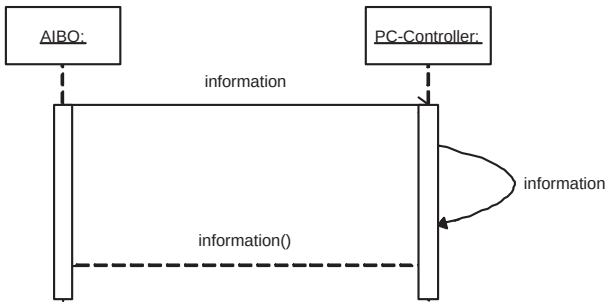

Figure 11. PHA input diagram as UML sequence diagram

for example *deliberate manipulation of information on communication channel*, which gives the threat *Incorrect, but valid information*. The column *Threat scenario* describes who or what causes the threat to occur and the column *Unwanted incident* describes what happens if the threat occurs. In the example death or severe damage to personnel is the unwanted incident of the threat that incorrect, but valid information is sent on the communication channel because an outsider has altered the information.

We use the UML sequence diagram in Figure 11 as the PHA input diagram. PHA input diagrams specify both the structural and behavioral aspects of a system, and one typically makes use of a set of UML diagrams as PHA input diagrams in order to cover both aspects during risk analysis. Sequence diagrams describe the behavior of the system.

Figure 12 provides an example of a PHA with Figure 11 as PHA input diagram.

ID	Pre-Guideword	Asset	Attribute	Component	Post-Guideword	Threat	Threat scenario	Unwanted incident
1a	Deliberate	Information	Manipulation	Communication	Outsider	Incorrect, but valid information	Outsider alters information	Death or severe damage to personnel

Figure 12. Example of use of guideline for use of UML sequence diagram as input to Security-HazOp

The main result from PHA is a list of security threats which are then used as input to a security requirement specification and allocation, which is the next phase in the development process. In this context, we focus on the security attribute integrity and the security threats related to breach of integrity. We look into the communication between the AIBO robot and the PC controller where any alteration, being either accidental or intentional, may lead to unauthorized access to the system, which might lead to death or serious damage to either unauthorized or authorized personnel. Since we are dealing with a distributed object–oriented system we need to make use of secure communication link between the monitoring and control system (see Section 3) to ensure integrity for information in transit. This can be ensured by encrypting the communication link, which is WLAN using TCP/IP as the communication protocol, between the AIBO and the PC controller.

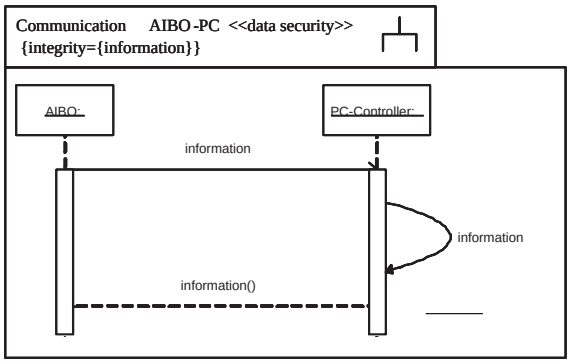

Figure 13. Security requirement for integrity preservation of the communication between AIBO and PC-controller

The treatment option is transformed into security requirements in the next phase of the development process, which is the risk management and specification of security requirements phase.

8.3. Risk management and specification of security requirements

Risk management concerns the following activities:

- specifying security requirements addressing security threats from PHA,
- performing risk identification to reveal unsolved security issues, and
- analyzing and proposing treatments for the unsolved issues evaluated as not acceptable.

In our example, the PHA sketched in the previous section identified the need to preserve the integrity of the communication between AIBO and the PC-controller. In this phase in the development, we specify the security requirements using UMLsec. We make use of the UMLsec stereotype « data security » and the {integrity} as defined in Section 4 to fulfill the demand on preserving integrity of data in transit. Figure 13 depict the specification of the security requirement integrity preservation specifying the communication as « data security » and specifying the data in need of protection using the {integrity}.

As defined in Sect. 4 and Sect. 5, for an adversary type A and a stereotype s, we have $\mathsf{Threats}_A(s) \in \{\mathsf{delete}, \mathsf{read}, \mathsf{insert}, \mathsf{access}\}$, which are the

actions that adversaries are capable of with respect to physical links or nodes stereotyped with s. Specifying the security requirement for preservation of integrity is done using the UMLsec stereotype « data security » in connection with the {integrity} on the transport layer of the model, and the stereotype « secure links » on the physical layer. The constraint on the communication links between AIBO and PC-controller is that for each dependency d with stereotype « integrity » between components on nodes $n \neq m$, we have a communication link l between n and m with stereotype t such that $\mathsf{insert} \notin \mathsf{Threats}_A(s)$.

In the next phase of the development process, the security requirements are addressed and allocated through treatment options. This is further implemented and validated during the testing and security validation phase of the development.

8.4. Design and implementation

Design in this context relates to the allocation of security requirements, while implementation relates to the actual implementation of the requirements according to the design specification.

During the PHA and the risk management and specification of security requirements, we identified the need to preserve the integrity of the communication between the AIBO, representing the monitoring system, and the PC-controller, representing the control system. The communication link between the AIBO and the PC-controller is a WLAN connection, which is not encrypted by default. We address this requirement by making use of encryption according to the encryption protocol depicted in Figure 14.

We thus decide to create a secure channel for the sensitive data that has to be sent over the untrusted networks, by making use of cryptography. As usual, we first exchange symmetric session keys for this purpose. Let us assume that, for technical reasons, we decide not to use a standard and well-examined protocol such as SSL but instead a customized key exchange protocol such as the one in Fig. 14. The goal is to exchange a secret session key K, using previously exchanged public keys K_C and K_S, which is then used to sign the data s which should satisfy integrity before transmission. Here $\{M\}_K$ is the encryption of the message M with the key K, $\mathcal{S}ign_K(M)$ is the signature of the message M with K, and :: denotes concatenation.

One can now again use stereotypes to include important security requirements on the data that is involved. Here, the stereotype « critical data » labels classes containing sensitive data and has the associated tags {secrecy}, {integrity}, and {fresh} to denote the respec-

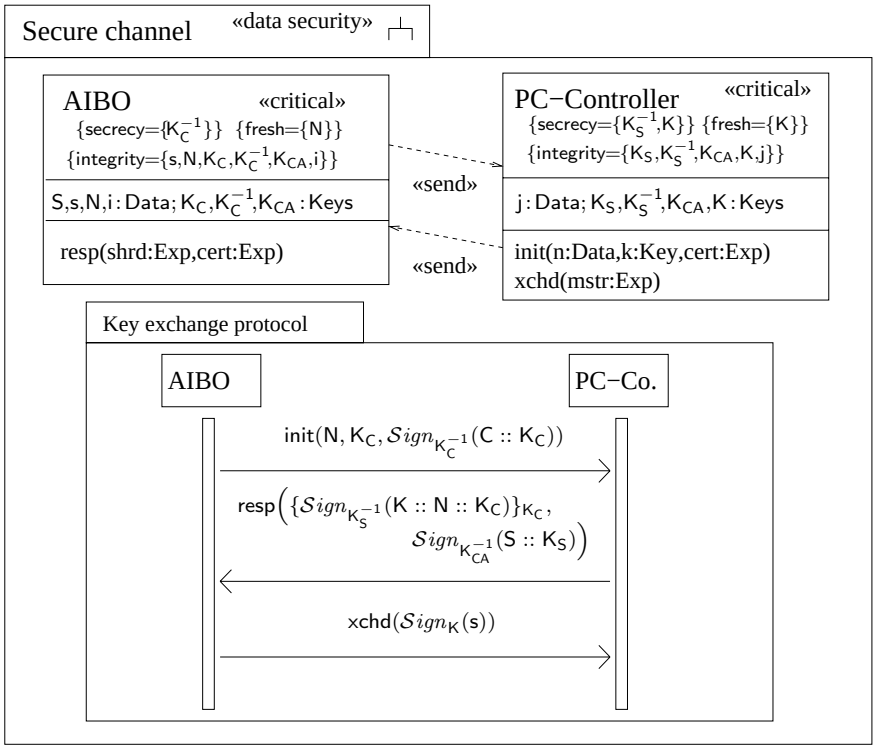

Figure 14. Key exchange protocol

tive security requirements on the data. The constraint associated with «data security» then requires that these requirements are met relative to the given adversary model. We assume that the standard adversary is not able to break the encryption used in the protocol, but can exploit any design flaws that may exist in the protocol, for example by attempting so-called "man-in-the-middle" attacks (this is made precise for a universal adversary model in Sect. 5.1). Technically, the constraint then enforces that there are no successful attacks of that kind. Note that it is highly non-trivial to see whether the constraint holds for a given protocol. However, using well-established concepts from formal methods applied to computer security in the context of UMLsec, it is possible to verify this automatically.

We refer to [Sør02] for further details on these two phases in the development process.

8.5. Testing and security validation

Testing and security validation target both the testing of functional requirements and the validation of the fulfillment of security requirement. In this context, we refer to other sources for testing strategies such as [Pat00] and will only discuss and illustrate how to perform the security validation.

Security requirements are specified using UMLsec and the security validation is performed using the tool support for UMLsec as described in Section 5.3.

9. Conclusion

Traditionally, software development processes do not offer particular support for handling security requirements. In most cases, security issues are only considered after the fact, which is both costly and resource demanding. Security should therefore be handled as an integrated part of system development. The focus is on providing an adequate level of security given resource bounds on time and money.

We have presented a MBRA development process for security-critical systems based on the safety standard IEC 61508 and integrated system development and risk management process of CORAS. The process consists of seven phases:

(1) Concept,

(2) Scope definition,

(3) Preliminary Hazard Analysis,

(4) Risk Management,

(5) Design,

(6) Implementation, and

(7) Testing and security validation.

The main aim is to use models not only to specify and document the system, but also as input into the PHA and risk management.

In our approach, models are used for five purposes:

(1) precise specification of non-functional requirements,

(2) as a medium to communicate non-functional requirements,

(3) to describe the target of assessment,

(4) as a medium to communicate risk assessment results, and

(5) to document risk assessment results.

Furthermore, models are also used for security validation using tool support for UMLsec. The main purpose of this is to validate that the implementation fulfills the security requirements.

The process is illustrated using an AIBO-Lego Mindstorm prototype system, where the focus is on the computerized part of the system and how security threats may affect the safety of the system. However, the process is designed for security-critical system in general and target both small web applications as well as large scale production systems.

Acknowledgments

The work is based on the results from the IST-project CORAS and the work done by the 11 partners in this project and the Master Thesis of Karine Sorby, NTNU, Norway.

References

AS/NZS 4360:1999. Risk management. Standards Australia, Strathfield, 1999.

B. Barber and J. Davey. The use of the ccta risk analysis and management methodology cramm in health information systems. In K.C. Lun, P. Degoulet, T.E. Piemme, and O. Rienhoff, editors, *MEDINFO 92*, pages 1589–1593, Amsterdam, 1992. North Holland Publishing Co.

S. Brown. Overview of iec 61508: design of electrical/electronic/programmable electronic safety-related systems. *Computing Control Engineering Journal*, 11:6–12, February 2000.

CORAS, The CORAS Intregrated Platform. Poster at the CORAS public workshop during ICT-2002, 2002.

DoD. Military standard: System safety program requirements. Standard MIL-STD-882B, Department of Defense, Washington DC 20301, USA, 30 March 1984.

D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

E.B. Fernandez and J.C. Hawkins. Determining role rights from use cases. In *Workshop on Role-Based Access Control*, pages 121–125. ACM, 1997.

G. Georg, R. France, and I. Ray. An aspect-based approach to modeling security concerns. In Jürjens et al. [JCF+02].

S.B. Guarro and D. Okrent. The logic flowgraph: A new approach to process failure modeling and diagnonsis for disturbance analysis applications. *Nuclear Technology*, page 67, 1984.

B.A. Gran, N. Stathiakis, G. Dahll, R. Fredriksen, A. P-J.Thunem, E. Henriksen, E. Skipenes, M.S. Lund, K. Stlen, S.H. Houmb, E.M. Knudsen, and E. Wislff. The coras methodology for model-based risk assessment. Technical report, IST Technical Report, http://sourceforge.coras.org/, 2003.

B.A. Gran, R. Winther, and O-A. Johnsen. Security assessments for safety critical systems using hazops. In *In Proceeding of Safecomp 2001*, 2001.

S.-H. Houmb, F. den Braber, M. Soldal Lund, and K. Stolen. Towards a UML profile for model-based risk assessment. In Jürjens et al. [JCF+02].

IEC 61508: 2000 Functional Safety of Electrical/Electronic/Programmable Electronic (E/E/PE) Safety-Related Systems".

J. Jürjens, V. Cengarle, E. Fernandez, B. Rumpe, and R. Sandner, editors. *Critical Systems Development with UML*, number TUM-I0208 in TUM technical report, 2002. UML'02 satellite workshop proceedings.

J. Jürjens. UMLsec: Extending UML for secure systems development. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *UML 2002 – The Unified Modeling Language*, volume 2460 of *LNCS*, pages 412–425, Dresden, Sept. 30 – Oct. 4 2002. Springer.

J. Jürjens. Developing Security-Critical Systems with UML, 2003. Series of tutorials at international conferences including OMG DOCsec 2002, IFIP SEC 2002, APPLIED INFORMATICS 2003, ETAPS 2003, OMG Workshop On UML for Enterprise Applications 2003, Formal Methods Symposium 2003. Download of material at http://www4.in.tum.de/~juerjens/csdumltut .

J. Jürjens. *Secure Systems Development with UML*. Springer, 2003. In preparation.

I.S. Kim and M. Modarres. Application of Goal Tree-Success Tree Model as the Knowledge-Base of Operator Advisory System. *Nuclear Engineering & Design J.*, 104:67–81, 1987.

P. Krutchten. *The Rational Unified Process, An Introduction*. Readings, MA. Addison-Wesley, 1999.

K. Lano, K. Androutsopoulos, and D. Clark. Structuring and Design of Reactive Systems using RSDS and B. In *FASE 2000*, LNCS. Springer-Verlag, 2000.

N. G. Leveson. *Safeware: System safety and computers*. Addison-Wesley, 1995. ISBN: 0-201-11972-2.

S. Meng. Secure database design with UML. Bachelor's thesis, Munich University of Technology. In preparation.

R. Patton. *Software Testing*. SAMS, 2000.

R.F. Paige and J.S. Ostroff. A proposal for a lightweight rigorous UML-based development method for reliable systems. In *Workshop on Practical UML-Based Rigorous Development Methods*, Lecture Notes in Informatics, pages 192–207. German Computer Society (GI), 2001. UML 2001 satellite workshop.

J.R. Putman. *Architecting with RM-ODP*. Prentice-Hall, 2000.

M. Shaw. Writing good software engineering research papers. In *25th International Conference on Software Engineering*, page 726, Portland, Oregon, May 03 - 10 2003.

K. Sørby. Relationship between security and safety in a security-safety critical system: Safety consequences of security threats. Master's thesis, Norwegian University of Science and Technology, 2002.

Udo Voges, editor. *Security Assessments of Safety Critical Systems Using HAZOPs*, volume 2187 of *Lecture Notes in Computer Science*. Springer, 2001. ISBN: 3-540-42607-8.

B. Watson. The Real-time UML standard. In *Real-Time and Embedded Distributed Object Computing Workshop*. OMG, July 15-18 2002.

G. Wyss, R. Craft, and D. Funkhouser. *The Use of Object-Oriented Analysis Methods in Surety Analysis*. Sandia National Laboratories Report, 1999.