UniForM Workbench

Formal Methods and the Development of Dependable Systems

Technical Report 9601

Jan Peleska
University of Bremen, FB3
D-28334 Bremen
Tel.: +49 421 218 7092
Fax: +49 421 218 3054
e-mail: jp@informatik.uni-bremen.de

$Id:$

$Revision:$

$Date:$

Abstract

In this article we describe a formal framework for the development of dependable systems, supporting the systematic development of cooperating mechanisms designed to encounter *combinations* of threats. We focus on the combination of fault-tolerance mechanisms and security mechanisms in communication protocols.

Until recently, fault-tolerant systems and secure systems were discussed and developed, at least in Germany, in different – often non-communicating – communities. Based on experiences with systems developed by the author in various industrial projects we motivate that a unified approach for the development of systems that combine both security and fault-tolerance properties will become increasingly important.

Practical experience has shown that the *a posteriori* integration of security features into a fault-tolerant system and vice versa is much more complicated than pre-planning every desired dependability and security feature simultaneously during the specification phase of the project. Moreover, for maximum-quality systems intended to operate in safety-critical or otherwise sensitive environments the application of formal methods for the development and verification of the system is becoming increasingly 'popular' and in some cases even mandatory. Therefore, we are investigating formal specification and verification techniques supporting the description of *all* dependability aspects. To be suitable for large-scale projects in an industrial development environment, a formal approach has to be supported by a development standard, providing a re-usable framework for the specification and verification activities to be performed during the various phases of system development. To this end, we will describe the application of our approach as embedded into the *Vorgehensmodell (V-Model)* [105] which is authorized by the Germany Ministry of the Interior and considered as the state-of-the-art software development standard in Germany.

Our approach will be illustrated by means of a case study describing a fault-tolerant and secure communication protocol.

Keywords: dependability — safety — reliability — availability — security — CSP — communication protocols

Acknowledgements: This work has been supported by the German Ministry of Education and Research BMBF, by ELPRO LET GmbH Berlin and by JP Software-Consulting Kiel.

1. Preface

1.1 Overview

This report describes a formal approach for the systematic development of dependable systems. The necessity for such an approach is illustrated below by means of an industrial project in the field of railway interlocking systems, which is a typical example of control systems requiring the full scale of dependability properties. After introducing the basic notions of dependability in Chapter 2, an introductory example is presented in Chapter 3, showing the complexity to be encountered when trying to combine different mechanisms to protect a system against *a combination of threats*, instead of a single one. We claim that any formal method suitable for the development of dependable systems should be applied according to a *development standard* (Section 4.1)[1]. Such a standard provides a re-usable framework for the specification and verification activities to be performed during the various phases of system development. It does not require a specific method to be applied but describes the activities to be performed and objects to be produced during the development stages. Tailored for the specific problems related to the development of dependable systems, it ensures that design errors as the one discussed in Chapter 3 are very unlikely to occur. Indeed, today many researchers and practitioners are convinced that a formal method can only be successfully applied in an industrial context if supported by both tools and a development standard. Therefore, in Chapter 4, we introduce the *Vorgehensmodell (V-Model)* [105] which is authorized by the Germany Ministry of the Interior and considered as the state-of-the-art software development standard in Germany. It is described how to apply our formal approach in this framework. Application of a standard helps to benefit from a formal method in the most systematic way during the software life cycle. Conversely, the formal approach will increase the insight about how the standard should be applied in an optimal way and what the documents informally introduced by the standard should describe to achieve a useful and reliable description of the system to be developed.

To illustrate the application of our approach, the case study of Chapter 3 will be re-worked in Chapter 5 in a systematic way, this time leading to a correct solution allowing to re-use at least one of the protocols introduced before to defend the system against isolated threats.

The work presented in this report is a revision and extension of the results described in [75], using CSP [41] as the underlying method for the specification and verification of distributed systems. To overcome the problems specifically related to dependability in a systematic way, we make use of a method developed by Schepers [99] in the context of fault-tolerant systems. We will show in Chapter 4 how his method may be extended to the field of dependability in general, transforming the design obligation to develop a dependable system into the design obligation for an "ordinary" system.

[1] In this context *"standard"* means either an *(inter-)national* standard regulating development procedures or a *"customized" company* standard, which is usually an instantiation of an international or national standard for the specific type of tasks to be solved in the company.

1.2 Related Industrial Projects

The following example of a "real-world" project is presented to support the assertion that modern control systems should combine aspects of both "classical" dependability – i. e. safety, reliability, availability – *and* security. It has motivated the case study presented in Section 3, to be continued below. The example is typical for systems where classical dependability has always been an important issue, but the demand for security mechanisms is relatively new. Further examples are given in [75].

Distributed Railway Control System In Germany today, available electronic computer based railway control systems are centralized. One signal tower controls the state of all signals, points and level crossings of a specific area. Signals, points and level crossing barriers (or traffic lights) are directly wired to the signal tower. Each train is supervised by exactly one signal tower at a time. This technique is much too expensive for small private tramways, railway or underground networks. On the other hand, in the future these smaller networks will depend on highly automated control systems, because otherwise the operational costs would be too high. Therefore several European companies and research institutions investigate concepts for decentralized railway control.

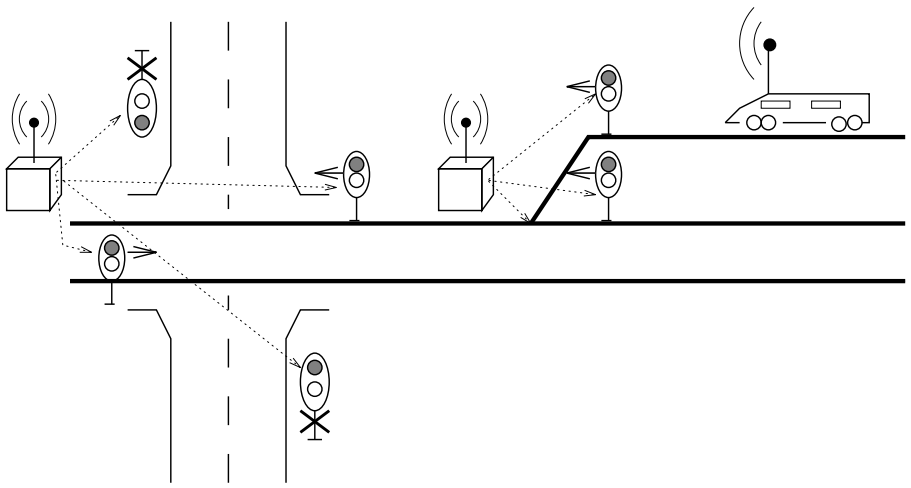

Figure 1.1: System overview of a distributed interlocking system.

In one of these concepts[2] the control logic is distributed on a collection of 'local' switch boxes and on the trains (Figure 1.1). Each switch box controls a small number of nearby signals, points and level crossings. It stores its local system state, supervises its local safety conditions and communicates with approaching trains. The control computer of a train will request local state information from (several) switch boxes and decide, based on the state information available, whether signals, points and barriers or traffic lights may be switched according to the request of the train or whether the train will have to wait at a signal. Data between trains and switch boxes is communicated via radio transmission, preferably making use of standard networks designed for the purpose of digital data transfer.

[2]The information in this paragraph is presented by courtesy of ELPRO LET GmbH. At present, I am working as a consultant for this company in the field of test automation and verification of components of interlocking systems. The concept is described in the proposal [23] in more detail.

This basic idea induces a variety of dependability requirements differing from those of centralized railway control system:

- The decision whether a train is allowed to pass a certain signal, point etc. is produced by a distributed algorithm which involves switch boxes "safe-guarding" their local safety requirements and the train control computer deducing the "global decision" from the local information of the switch boxes. The algorithm must never allow a transition into an unsafe state, even in the presence of component failures (safety requirement plus reliability requirement).
- Data exchange between trains and switch boxes has to apply protocols based on repreated package transmission to avoid the loss of data caused by accidental disturbance of the radio transmission (reliability requirement).
- The data transmission must also be protected against *systematic* manipulation by humans using radio transmission devices (security requirement).

Though the accidental physical disturbance of radio transmission will certainly be the threat with the highest probability, I believe that the idea of humans trying to manipulate the exchange of control signals does not represent a theoretical, but a very realistic threat. If hackers find it fascinating to plant viruses into other people's PCs (even though they cannot watch the effect in most cases), how much more fun would it be to stop, accelerate and re-direct a real train by "remote control", while watching the railway track! For these reasons, the distributed concept will introduce a new level of complexity. However, this does not represent an unmanageable problem, because the networks we consider here are much simpler structured and have a lower traffic rate than the big railways networks where the centralized concepts are used.

Contents

1 Preface **3**

1.1 Overview 3

1.2 Related Industrial Projects 4

2 Introduction **9**

2.1 Dependability — Basic Definitions and Concepts 9

3 Combination of Dependability Mechanisms — Case Study Part I **13**

4 A Framework for the Development of Dependable Systems **22**

4.1 Standards for the Development of Dependable Systems 22

4.1.1 Overview 22

4.1.2 System Development According to the V-Model 23

4.2 A Formal Approach for the Development of Dependable Systems 26

4.2.1 Selection Criteria for the Formal Method 26

4.2.2 System Requirements 27

4.2.3 System Architecture 33

4.2.4 Verification on System Level 35

4.2.5 Recursive Application of the Development Procedure 36

5 Combination of Dependability Mechanisms — Case Study Part II **38**

5.1 System Requirements 39

5.1.1 Operational Environment 39

5.1.2 Application Requirements 39

5.1.3 Threat Analysis (External) 40

5.1.4 Dependability Requirements 44

5.2 System Architecture 45

5.2.1 System Structure 45

5.2.2 Interfaces 45
5.2.3 Component Specifications 46
5.2.4 Threat Analysis (Internal) 47
5.3 Verification on System Level 49
5.4 Recursive Application of the Development Procedure 54
5.5 Verification of Deadlock Freedom 56

6 Conclusion **60**

A Glossary of Symbols **68**

B A Quick-Reference Guide to CSP **70**

List of Figures

1.1 System overview of a distributed interlocking system. 4

2.1 Types of system behaviour. 12

3.1 Context of the target system (network layer) and its environment. 13

3.2 Unreliable transmission media and alternating bit protocol implemented by ABP_TX, ABP_RC. 15

3.3 Security layer $SEC1_TX, SEC1_RC$ to protect the application layer against eavesdropping and tampering on $M2$ by agent X. 18

3.4 Naive combination of security layer and network layer with unreliable *and* insecure medium $M3$. 20

4.1 Relationship between standards, methods and tools. 22

5.1 System configuration for the case study, part II 38

5.2 System architecture consisting of network layer and security layer. 45

5.3 Internal threat analysis: Possible loss of data on channel abp_rc. 48

5.4 Correct combination of a new security layer with re-used network layer. . . . 55

2. Introduction

2.1 Dependability — Basic Definitions and Concepts

Throughout this work, we will follow Laprie's terminology [53, 54, 55] with respect to dependability and related terms.

Generally speaking, *dependability is the capability of a system to deliver the specified application services during its period of operation.* This definition emphasizes two aspects that influence the design, the implementation and our reasoning about system "correctness" in a crucial way:

- Dependability does *not* forbid the occurrence of failures in general. Instead, it requires that the *application service*, i. e., the functionality required by the end user of the system is delivered as specified.
- Delivery of service is only required to the extent *covered by the specification.* This is a very reasonable requirement – otherwise the supplier would never have a chance to prove that the system has been completed and the customer's requirements have been implemented. On the other hand, the approach is only useful, if the technical contents of a specification document reflecting how a system *will* behave is consistent with the end user's intuitive understanding of how the system *should* behave. As a consequence, the specification phase has become the most critical phase of the whole system development life cycle.

Laprie identified four attributes which characterize the dependability of a system: (1) A *safe* system cannot assume states that are regarded as "catastrophic" from the point of view of the application. This means that the system will only perform transitions into states satisfying the specified invariants, perform calculations that are correct with respect to the specification and output data fulfilling the desired integrity constraints. Safety does not guarantee that a desired calculation and the corresponding output will always be produced. This aspect is covered by the following two attributes: (2) *Reliability* is a characteristic specifying the probability that a system will deliver its service for a given period of time[1]. (3) *Availability* is a measure reflecting the probability that the system will be available at a certain point in time. (4) Finally, *Security* reflects the capability of the system to protect the application against damage arising from accidental or malicious human interaction. All dependability attributes refer to the specified application, and this is the very premise for the notion of *fault-tolerance*: A dependable system may be subject to various *internal* defects, as long as these problems do not affect the application behaviour.

For the security attribute, a further characterization has been provided: According to the standards [14, 25, 26, 44, 104] the notion of *Information Technology (IT) Security* is defined

[1] For example, the *mean time between failure (MTBF)* is a reliability measure.

by the attributes *Confidentiality* (prevention of unauthorized disclosure of information), *Integrity* (prevention of unauthorized modification of information) and *Availability* (prevention of unauthorized withholding of information or resources)[2]. IT security focuses on the damage that can be caused by humans. In [14, p. 3] this is made explicit by defining IT security as *"... protection of information ... by countering threats to that information arising from human activities whether malicious or otherwise."* There is at least one good reason to distinguish between threats arising from technical deficiencies and those arising from human activities: In many cases technical faults can be considered as accidental events and therefore adequately modelled by statistical approaches. These modelling techniques will fail, however, when being applied to the behaviour of malicious human intruders, who can try to cause damage in a systematic way.

Since the definition of dependability is intended to be applicable to a wide spectrum of systems and the aspects of dependability are intended to be describable by a wide range of formalisms, Laprie's definitions are of informal nature. It is useful to relate them to the formal notions of *safety* and *liveness*. Recall that in the context of formal methods a specification item S is a *safety property*, if any sequence of events or transitions etc. violating S contains a prefix all of whose infinite extensions violate S. A specification item L is called a *liveness property*, if any arbitrary finite sequence of events can be extended to an infinite sequence satisfying L [59, p. 303]. Obviously, the dependability attribute *safety* is also a safety property in the sense of formal methods. The situation is less obvious for *availability*: In the context of hard real-time systems, it is best interpreted as another safety property, because the availability of a service is then interpreted as *"available within n time units"*, and an assertion saying that the service will "finally" be available is not helpful at all. For less time-critical applications, however, the notion of availability might be associated with *"the service will not be blocked forever"* or *"the service will be activated sufficiently often"*, and these are liveness properties. As a consequence, the formalization of the availability definition depends on the specific application. *Reliability* is only indirectly related to safety and liveness: Intuitively, this dependability attribute describes the possibilities of the system to "switch" from one specified behaviour to another behaviour which is regarded as exceptional. Of course, both behaviour specifications can be decomposed into safety and liveness properties. The *confidentiality* and *integrity* aspect of security is a safety property, for the *availability* aspect of security the same considerations apply as for availability in general.

The notion of security is a "new" dependability attribute, due to Laprie. For example, in the definition given 1985 in [7], security is not yet mentioned, and even in a fairly recent survey [101] on real-time computing the authors state that *... it is not very important in a real-time operating system to provide extensive support for ... security.* I think that this fact is characteristic for the type of systems designers had – and sometimes still have – in mind when coining the notion of dependability in the context of computer science for the first time. The major threats to dependability were hardware and software errors and external physical events. Threats caused by humans were thought to be similar to natural disasters like acts of vandalism. The more "subtle" aspects of security were not considered as

[2]Obviously, these attributes introduce redundancy in the dependability definition, since availability has already been introduced as a main attribute of dependability and integrity can be regarded as a safety aspect. These issues have been discussed by Jonsson and Olovsson [50]. However, it is not our objective to try and improve existing definitions, but to find trustworthy ways how to build systems according to given dependability requirements.

relevant in the early eighties. On the other hand, it has always been an obvious requirement that a dependable system should also be secure because otherwise, a human attacker might systematically damage the integrated dependability mechanisms. However, in the "early times" of dependable computerized systems, security issues could often be ignored *in the software design*, because the operational environment was *automatically secure*, i. e. did not possess any interfaces accessible to a human attacker. In recent years, a growing demand for *open* and/or *distributed* dependable systems became apparent. Obviously, these new key attributes force us to treat security as a global system issue which has to be covered by protective mechanisms both in the operational environment *and* in the software. Examples for such projects are given in Chapter 4 and in [75]. From my experience it is interesting to note that the demand for open, distributed dependable systems came from two sides:

- In the "classical" application domains of dependable systems openness and distribution have often been introduced to reduce costs by providing a system architecture that can be customized and extended more easily.
- In less critical applications dependability became an important aspect *a posteriori*, in order to reduce costs by increasing the operational availability and to avoid loss or corruption of valuable data. These systems were often open and distributed by their very nature, and the new dependability features had to be fitted into this existing architectural framework.

I will use the term *classical dependability* when referring to safety, reliability and availability only.

To illustrate the design tasks to be tackled for the development of dependable systems the classification of system behaviours depicted in Figure 2.1 will be used. In absence of any undesired (internal or external) events, the system executes[3] *normal behaviour*. In presence of undesired events the system may perform an execution possibly differing from that of the undisturbed system. Now there are two possibilities: In the first case, an undesired event impairs the functionality of the system in a way that can be tolerated or might even be unobservable in the application layer. In this case the system shows *exceptional behaviour*. If the system performs an execution showing normal and/or exceptional behaviour, this is summarized as *acceptable behaviour*. In the second case, the occurrence of the undesired event corrupts the application functionality in a way that cannot be tolerated. This is called *catastrophic behaviour*. Although the above classification has been introduced in [56] to describe concepts of fault-tolerance only, it fits very well to deal with dependability as a whole.

In order to prevent undesired events from leading to catastrophic system behaviour, system designers must anticipate the possibility of their occurrence and incorporate protective mechanisms in their development concepts. The potential occurrence of any undesired event will be called a *Threat* or a *Fault Hypothesis*. If it is necessary to distinguish between aspects of classical dependability and security, we will denote the fault hypotheses anticipating human interactions as *Security Threats*. The actual occurrence of any undesired event is called a

[3]Depending on the formalism used, an execution may be a modelled as a *trace of events*, hiding internal state information, or as a sequence of *state transitions* triggered by events. For hybrid systems, the execution is modelled as some type of *trajectory*, traversing an appropriate space-time model spanned by state and time parameters.

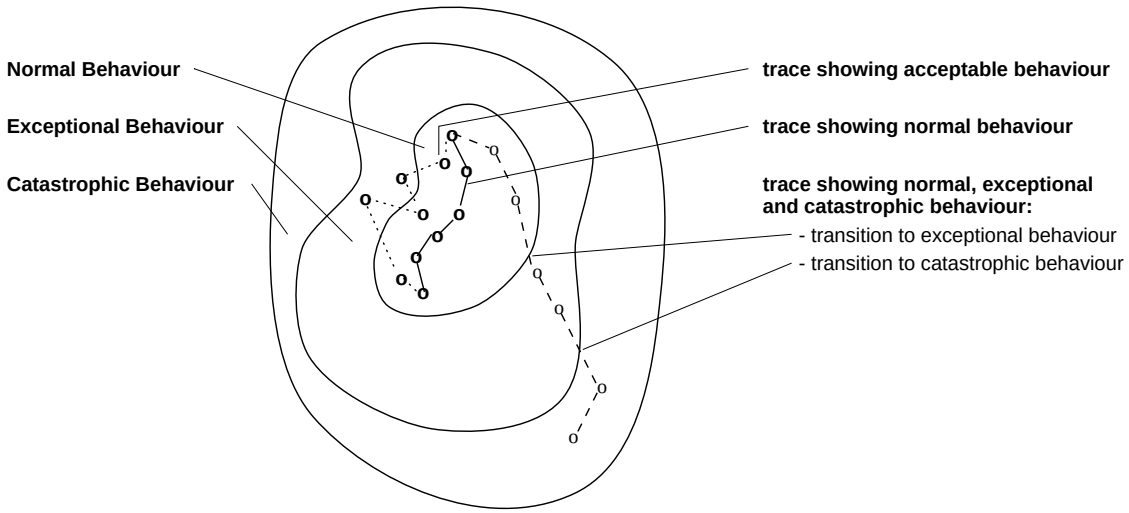

Figure 2.1: Types of system behaviour.

Fault. Faults caused by security threats are termed *Human-Made Faults*; we will also use the more suggestive term *Security Attack* and summarize human-made and other faults as *Exceptions*. The process of anticipating exceptions will be denoted as *Threat Analysis*. If an exception causes a transition into an undesired system state, this state is called an *Error*. If due to the occurrence of exceptions and errors a specified service to be performed by a (sub-)system cannot execute as required, this is called a *Failure*. In the context of real-time systems it is often necessary to distinguish *Value Failures*, where incorrect data is produced or processed by the system from *Timing Failures*, where correct data appears at the wrong time.

The means to prevent faults, errors and failures are *Fault Avoidance* (building "perfect" hardware and software components by means of special construction techniques), *Fault Tolerance* (use of techniques allowing the continuation of service in presence of faults), *Verification* (detection and removal of errors by means of analytic techniques) and *Error Forecasting* (analysis of the impact of specific errors and of the probability for their occurrence). The objects implemented to prevent the undesired consequences of security attacks are called *Security Mechanisms*.

To model the system behaviour in presence of exceptions, two main techniques have been used by different authors:

- *Explicit specification,* for example by specifying the system and the impact of exceptions in terms of a process algebra. This method will be demonstrated in the next chapter.
- *Behavioural specifications:*[4] The system behaviour and the impact of anticipated exceptions is described by predicates on traces of events or executions of a transition system [67, 99].

[4]Following Jones [49], we also use the term *implicit* specifications.

3. Combination of Dependability Mechanisms — Case Study Part I

While many publications investigate isolated dependability aspects[1], relatively few of them are concerned with combinations of these aspects. I am convinced, however, that it is just this combination that introduces a new degree of complexity in the field of dependable systems, because

- it can be dangerous to combine two dependability mechanisms in a naive way and hope that the resulting system will automatically possess the strength of both mechanisms,
- the design of correctly cooperating dependability mechanisms is likely to differ strongly from solutions where only one isolated mechanism is required.

To support these theses, we will now analyze a case study focusing on the development of a network protocol which is dependable in the presence of both unreliable and insecure transmission media (Figure 3.1). The case study is a revised version of the example worked out in [75]. In the remaining part of this chapter it will serve to illustrate the complexity to be expected when more than one dependability threat is involved: two mechanisms correctly defending the system against the reliability threat and the security threat alone, respectively, will result in an *unreliable* protocol when combined in a "naive" way. This failure to construct "globally" dependable mechanisms from isolated solutions in an intuitive way will motivate the systematic approach for the development of dependable systems as introduced in Chapter 4.

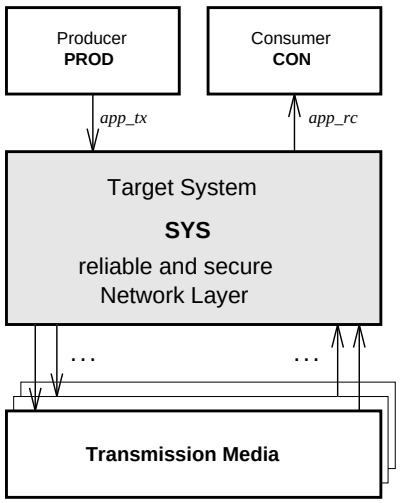

Figure 3.1: Context of the target system (network layer) and its environment.

[1] Rushby [97] has collected a comprehensive bibliography.

In the examples to follow the number of transmission media and their various deficiencies will vary in order to illustrate different aspects of dependability. The dependability requirement will always remain the same, however:

- Construct *SYS* in such a way that the resulting communication service will act for producer *PROD* and consumer *CON* as a FIFO buffer without loss of data or corruption of messages.

I will use CSP (*Communicating Sequential Processes*) as introduced by Hoare [41] to specify and reason about system behaviour. Correctness properties will be expressed by means of *failures-divergence refinement* relating the CSP process expressing correct behaviour to the protocol solutions proposed in the examples. The refinement steps presented have been mechanically checked using the FDR tool [24]. FDR allows to verify refinement properties by means of complete model checking for CSP systems consisting of cooperating finite-state sequential processes. The specification by explicit processes only serves for illustration purposes in these examples. In Chapter 4, a behavioural specification style will be used which is more adequate for the abstract description of large systems.

The informal dependability requirement can be specified by relating an implementation X to the explicit CSP process[2] *FIFO* defined as

$$\alpha(FIFO) = \{|\ app_tx, app_rc\ |\}$$

$$FIFO = BUFF(\langle\ \rangle)$$

$$BUFF(s) = \begin{array}{l}(\#s < N)\&app_tx?x \rightarrow BUFF(s^\frown\langle x\rangle) \\ \Box \\ (\#s > 0)\&app_rc!(head(s)) \rightarrow BUFF(tail(s))\end{array}$$

which represents a FIFO buffer of finite capacity N: Any X consisting of the transmission media and the protocol layer *SYS* used in the implementation has to satisfy

$$FIFO \sqsubseteq_{FD} X \setminus (\alpha(X) - \{|\ app_tx, app_rc\ |\})$$

where $P \sqsubseteq_{FD} Q$ means that P is refined by Q in the failures-divergence model of CSP.

Example 3.1 (Communication disturbed by unreliable media) We first consider the case where only unreliable transmission media are available, behaving as follows:

1. **Fault hypothesis:** The medium may lose messages. The number of consecutive input messages that the medium may lose is bounded by a number $maxLoss \geq 0$.
2. The medium does not change the values of messages delivered to the consumer.
3. The medium delivers messages in the order of the corresponding inputs.

In order to design a fault-tolerant network layer taking into account the above fault hypothesis, the architecture shown in Figure 3.2 is used (the "lightning" symbols indicating

[2] We use *communication guards*: in an expression $b\&c$, the communication via channel c is refused by the process, if the Boolean expression b evaluates to *false*. $\{|\ c, d, \ldots\ |\}$ denotes the set of channel events $\{c.x_1, c.x_2, \ldots, d.y_1, d.y_2, \ldots\}$, x_i and y_j are values of the channel alphabets of c and d, respectively.

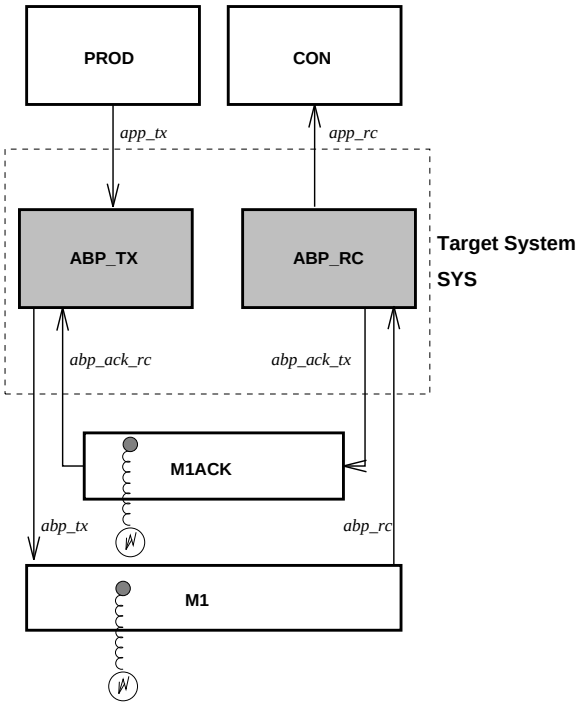

Figure 3.2: Unreliable transmission media and alternating bit protocol implemented by ABP_TX, ABP_RC.

the possibility of data losses). $M1, M1ACK$ are two instances of the unreliable medium. Their behaviour according to the above assumptions can be expressed by the following CSP processes:

$$M1 = M1(maxLoss)$$

$$\begin{aligned} M1(n) = {} & abp_tx?(b,x) \rightarrow \\ & (\textbf{if } n = 0 \textbf{ then} \\ & \quad (abp_rc!(b,x) \rightarrow M1(maxLoss)) \\ & \textbf{else} \\ & \quad (abp_rc!(b,x) \rightarrow M1(maxLoss) \\ & \quad \sqcap \\ & \quad M1(n-1))) \end{aligned}$$

$$M1ACK = M1ACK(maxLoss)$$

$$\begin{aligned} M1ACK(n) = {} & abp_ack_tx?b \rightarrow \\ & (\textbf{if } n = 0 \textbf{ then} \\ & \quad (abp_ack_rc!b \rightarrow M1ACK(maxLoss)) \\ & \textbf{else} \\ & \quad (abp_ack_rc!b \rightarrow M1ACK(maxLoss) \\ & \quad \sqcap \\ & \quad M1ACK(n-1))) \end{aligned}$$

To construct a reliable network layer on top of $M1, M1ACK$, we use the well-known *alternating bit protocol* (e. g., see [61, 99] for two alternative versions of the protocol) consisting of a transmitter ABP_TX and a receiver ABP_RC with $SYS = (ABP_TX \,|||\, ABP_RC)$. ABP_TX receives data from the producer, attaches a bit value alternating with each new message accepted on app_tx and sends it via $M1$ to ABP_RC. The receiver passes data via app_rc to the consumer, if the attached bit has the expected value in the alternating sequence. In any case ABP_RC sends the bit value received back to ABP_TX via medium $M1ACK$ to acknowledge reception. In our ABP solution ABP_TX will continuously transmit the actual $(bit, data)$-package until it receives an acknowledgement with the same bit value. The corresponding CSP processes are

$$ABP_TX = ABP_TX(1)$$

$$\begin{aligned}
&ABP_TX(bit) = \\
&\quad app_tx?x \rightarrow ATX((1 - bit), x) \\
&\quad \Box \\
&\quad abp_ack_rc?b \rightarrow ABP_TX(bit)
\end{aligned}$$

$$\begin{aligned}
&ATX(bit, v) = \\
&\quad abp_tx!(bit, v) \rightarrow ATX(bit, v) \\
&\quad \Box \\
&\quad abp_ack_rc?b \rightarrow \\
&\qquad (\textbf{if } (b = bit) \textbf{ then } ABP_TX(bit) \textbf{ else } ATX(bit, v))
\end{aligned}$$

$$ABP_RC = ABP_RC(0)$$

$$\begin{aligned}
&ABP_RC(bit) = \\
&\quad abp_rc?(b, x) \rightarrow \\
&\qquad (\textbf{if } (b = bit) \\
&\qquad\ \textbf{then } (app_rc!x \rightarrow abp_ack_tx!b \rightarrow ABP_RC(1 - bit)) \\
&\qquad\ \textbf{else } (abp_ack_tx!b \rightarrow ABP_RC(bit)))
\end{aligned}$$

The transmission media and SYS cooperate according to

$$ABP = (M1 \,|||\, M1ACK) \,\|\, (ABP_TX \,|||\, ABP_RC)$$

To prove that ABP is an acceptable implementation of our ideal transmission medium, we have to show that

$$FIFO \sqsubseteq_{FD} \; ABP \setminus \{| \, abp_tx, abp_rc, abp_ack_tx, abp_ack_rc \, |\}$$

which can be proven for finite $maxLoss$ and buffer capacity $N = 1$ using the FDR model checker.
□

Example 3.2 (Communication disturbed by insecure media) Let us now consider a transmission medium that is physically reliable but suffers from attacks by a malicious

agent X. In the field of communication protocols, the most important types of attack have been classified as follows (see [9]): *(1) Eavesdropping:* Messages sent from *PROD* to *CON* are properly received by *CON*, but X can also receive (some of) them. *(2) Blocking:* X can receive messages intended for *CON* and prevent them from being delivered to *CON*. *(3) Delay:* X can delay the transmission from *PROD* to *CON*. *(4) Masquerading:* X can send a fake message to *CON* that looks as if it has been sent by *PROD*. *(5) Replay:* X can send an old message that has already been sent from *PROD* to *CON* once more at a later point in time. *(6) Tampering:* X can alter a message while it is transmitted from *PROD* to *CON*, so *CON* receives modified data.

In this example, let the following behaviour be defined for the transmission medium:

1. **Security threat:** X can perform eavesdropping and tamper with data sent from *PROD* to *CON*. Making use of tampering, X can also replay a message by "copying" an old message onto a new package. X cannot block or delay messages or fake the identity of *PROD*. X can modify at most $maxModified \geq 0$ messages in a row.
2. The medium delivers messages in the order of the corresponding inputs.
3. The medium does not lose messages.
4. While transmission of user data can be tampered with, there exists a separate reliable and secure communication channel for the transmission of control data.

The last condition may seem slightly artificial. I have added it in order to construct an extremely simple security protocol that works properly in this example but fails when combined with a fault-tolerance mechanism, as will be shown in the next example.

To defend the application layer against the insecure transmission medium $M2$ the architecture depicted in Figure 3.3 is chosen.

The insecure behaviour of $M2$ may be modelled as follows:

$$M2 = M2(maxModified)$$

$$\begin{aligned} M2(n) = {} & sec_tx?x \rightarrow \\ & (\textbf{if } n = 0 \textbf{ then} \\ & \quad (sec_rc!x \rightarrow M2(maxModified)) \\ & \textbf{ else} \\ & \quad (sec_rc!x \rightarrow M2(maxModified) \\ & \quad \sqcap \\ & \quad sec_rc!modify(x) \rightarrow M2(n-1))) \end{aligned}$$

Here output $sec_rc!modify(x)$ represents the case where X has tampered with input $sec_tx.x$. Counter n prevents $M2$ from sending more than $maxModified$ corrupted outputs in a row. $M2ACK$ is assumed to be dependable and acts as a one-place buffer:

$$M2ACK = sec_ack_tx?b \rightarrow sec_ack_rc!b \rightarrow M2ACK$$

To meet the security threat represented by $M2$, the security layer implemented in *SYS* will use encryption techniques to detect altered or replayed messages and to prevent X from

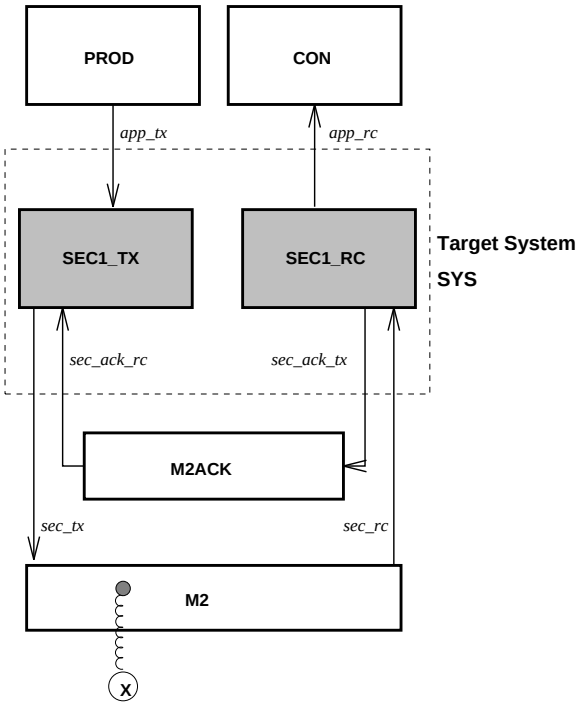

Figure 3.3: Security layer $SEC1_TX, SEC1_RC$ to protect the application layer against eavesdropping and tampering on $M2$ by agent X.

reading them. This technique will be explained in more detail in Chapter 4; for now it is sufficient that we may assume the existence of an encryption procedure ϵ and an detection algorithm which signals $isModified(x) = true$ if a package x has been tampered with. By ϵ^- we denote the corresponding decoding procedure. The security layer may now be specified as

$$SEC1_TX = app_tx?x \rightarrow sec_tx!\epsilon(x) \rightarrow S1TX(\epsilon(x))$$

$$\begin{aligned} S1TX(x) = {} & sec_ack_rc.Modified \rightarrow sec_tx!x \rightarrow S1TX(x) \\ & \Box \\ & sec_ack_rc.Valid \rightarrow SEC1_TX \end{aligned}$$

$$\begin{aligned} SEC1_RC = {} & sec_rc?y \rightarrow \\ & (\textbf{if } isModified(y) \\ & \ \textbf{then } (sec_ack_tx!Modified \rightarrow SEC1_RC)) \\ & \ \textbf{else } (app_rc!\epsilon^-(y) \ \rightarrow sec_ack_tx!Valid \rightarrow SEC1_RC) \end{aligned}$$

Process $SEC1_TX$ accepts inputs from the application layer via channel app_tx. It enciphers the message by means of ϵ and passes the result to $M2$. As long as X does not know the key to decipher the message, it can only corrupt the data at random or copy an old package over the new data block. But this will be detected by $SEC1_RC$ using the $isModified$ algorithm on the input y received on sec_rc. If the message has been modified or is a replay of an old one, $SEC1_RC$ will send a control message $Modified$ via $M2ACK$ to $SEC1_TX$ and the latter will re-transmit the enciphered message. Since according to the assumptions neither $M2$ nor $M2ACK$ lose messages and agent X can only modify, but not interrupt the communication

on $M2$, $SEC1_TX$ can be sure to receive a response from $SEC1_RC$ for each message transmitted. It is therefore not necessary to watch for timeouts while waiting for the control message. If the message has been received by $SEC1_RC$ without previous modification, the deciphered message contents will be passed on to CON, and control message *Valid* is sent back to $SEC1_TX$. After that $SEC1_TX$ is ready to accept new data from $PROD$.

The complete secure implementation cooperates according to

$$SEC = (M2 \,|||\, M2ACK) \parallel (SEC1_TX \,|||\, SEC1_RC)$$

and again it may be shown using the FDR tool that

$$FIFO \sqsubseteq_{FD} SEC \setminus \{| sec_tx, sec_rc, sec_ack_tx, sec_ack_rc |\}$$

for fixed values *maxModified* and buffer capacity $N = 1$. The example does not show how to initiate communication and exchange keys for enciphering/deciphering between $SEC1_TX$ and $SEC1_RC$. Possible solution for such identification, authentication and on-line key distribution procedures have been described in [9, 16].
□

Example 3.3 (Naive combination of protocols in presence of more than one threat) Now what happens if we are confronted with a transmission medium $M3$ which is both unreliable in the sense of the first example and insecure in the sense of the second? It may be tempting to try and combine the existing solutions in an architecture as shown in Figure 3.4. $SEC1_TX, SEC1_RC, M2ACK$ are chosen as in the previous example. $ABP1_TX, ABP1_RC$ implement an alternating bit protocol on top of $M3, M1ACK$. They are copies of ABP_TX/RC above, with channels app_tx, app_rc renamed to sec_tx, sec_rc. This seems to work at first sight, because the ABP layer makes up for lost messages and the security layer can deal with corrupted data.

We assume that $M1ACK$ behaves as in Example 1, but $M3$ now combines the fault hypotheses and security threats of the previous examples according to

$$\begin{aligned}
M3(n) = {} & abp_tx?(bit, x) \rightarrow \\
& (\textbf{if } n = 0 \textbf{ then} \\
& \quad (abp_rc!(bit, x) \rightarrow M3(maxModifiedOrLost)) \\
& \textbf{else} \\
& \quad (abp_rc!(bit, x) \rightarrow M3(maxModifiedOrLost) \\
& \quad \sqcap \\
& \quad M3(n-1) \\
& \quad \sqcap \\
& \quad abp_rc!(bit, modify(x)) \rightarrow M3(n-1) \\
& \quad \sqcap \\
& \quad abp_rc!((1-bit), x) \rightarrow M3(n-1) \\
& \quad \sqcap \\
& \quad abp_rc!((1-bit), modify(x)) \rightarrow M3(n-1)))
\end{aligned}$$

And this is the reason why this combination of mechanisms will fail: The resulting system will allow to *block* messages on app_tx because

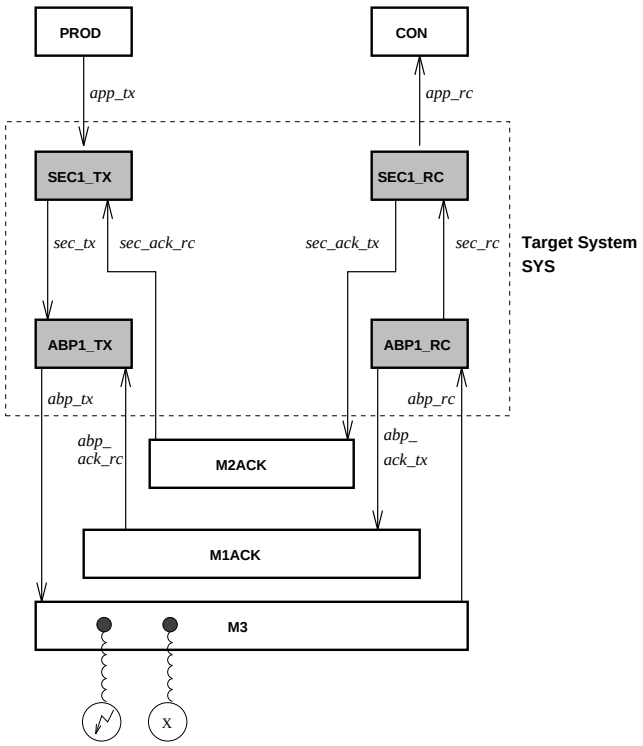

Figure 3.4: Naive combination of security layer and network layer with unreliable *and* insecure medium $M3$.

$$(M3 \,|||\, M1ACK) \,\|\, (ABP1_TX \,|||\, ABP1_RC) \,\|\, (SEC1_TX \,|||\, SEC1_RC)$$
$$\setminus \{|\, sec_tx, sec_rc, sec_ack_tx, sec_ack_rc, abp_tx, abp_rc, abp_ack_tx, abp_ack_rc \,|\}$$

diverges: There exist traces of unbounded length consisting only of events other than app_tx, app_rc. For example, consider the process $DivTrace$:

$$DivTrace =$$
$$app_tx?x \rightarrow Div0(x)$$

$$Div0(x) =$$
$$sec_tx.\epsilon(x) \rightarrow$$
$$abp_tx.(0, \epsilon(x)) \rightarrow abp_rc.(0, modify(\epsilon(x))) \rightarrow sec_rc.modify(\epsilon(x)) \rightarrow$$
$$sec_ack_tx.Modified \rightarrow sec_ack_rc.Modified \rightarrow abp_ack_tx.0 \rightarrow$$
$$abp_tx.(0, \epsilon(x)) \rightarrow abp_rc.(0, \epsilon(x)) \rightarrow$$
$$abp_ack_rc.0 \rightarrow abp_ack_tx.0 \rightarrow abp_ack_rc.0 \rightarrow$$
$$sec_tx.\epsilon(x) \rightarrow$$
$$abp_tx.(1, \epsilon(x)) \rightarrow abp_rc.(1, modify(\epsilon(x))) \rightarrow sec_rc.modify(\epsilon(x)) \rightarrow$$
$$sec_ack_tx.Modified \rightarrow sec_ack_rc.Modified \rightarrow abp_ack_tx.0 \rightarrow$$
$$abp_tx.(1, \epsilon(x)) \rightarrow abp_rc.(1, \epsilon(x)) \rightarrow$$
$$abp_ack_rc.1 \rightarrow abp_ack_tx.1 \rightarrow abp_ack_rc.1 \rightarrow Div0(x)$$

$DivTrace$ produces a trace which is correct with respect to the "naive system". This can be proven using FDR by checking that $DivTrace$ is a correct trace refinement of

$$(M3 \,|||\, M1ACK) \,\|\, (ABP1_TX \,|||\, ABP1_RC) \,\|\, (SEC1_TX \,|||\, SEC1_RC)$$

The problem arises as follows: The message x sent by $PROD$ via app_tx is modified during the first transmission on $M3$, identified by bit '0'. As a consequence the control message $sec_ack_tx.Modified$ is sent back to $SEC1_TX$. Meanwhile a re-transmission is initiated by the ABP layer because the acknowledgement associated with the first message has not yet been received by ABP_TX. This message passes $M3$ without any modifications, but it is useless, because ABP_RC discards it on account of the same bit identification '0' as in the first transmission. Next ABP_TX receives an acknowledgement for the first transmission, because is has been carried out successfully, as far as the ABP layer is concerned. Now the re-transmission $sec_tx.\epsilon(x)$ takes place, initiated by SEC_TX as a response to the $Modified$ control message. Again, this message is modified, this time carrying bit '1'. This procedure can be continued endlessly, and the hypothesis that at most $maxModifiedOrLost$ consecutive messages may be corrupted is never violated, if $maxModifiedOrLost > 0$.

Note that it does not help to swap the ABP layer and the security layer: Since $SEC1_TX$ has been built under the assumption that messages will only be changed, SEC_TX might wait forever for an acknowledgement on channel sec_ack_rc after a message has been lost on $M3$. This will prevent the ABP layer from re-transmitting the message.
□

In the next chapter, based on the systematic approach for the development of dependable systems, a correct protocol for the combined fault hypothesis and security threat will be presented.

4. A Framework for the Development of Dependable Systems

4.1 Standards for the Development of Dependable Systems

4.1.1 Overview

Software Development Standards provide frameworks for the activities to be performed and objects to be produced during the software life cycle. They describe the logical links between development objects (specification documents, software code, test data etc.) and specify the activities to be exercised on the objects (development, review, verification etc.), as well as their "synchronization" during the development life cycle. Ideally, the application of such a framework will help to meet the quality requirements applicable for the target system in the most systematic way. Standards admit a variety of methods – informal, semi-formal or formal – to be applied during the phases of system development. Therefore they are of informal nature themselves, written in natural language. However, they usually suggest the characteristics that should be present in a method, when applied to develop a system of certain criticality and according to pre-defined quality requirements. The relationship between standards, methods and tools is depicted in Figure 4.1.

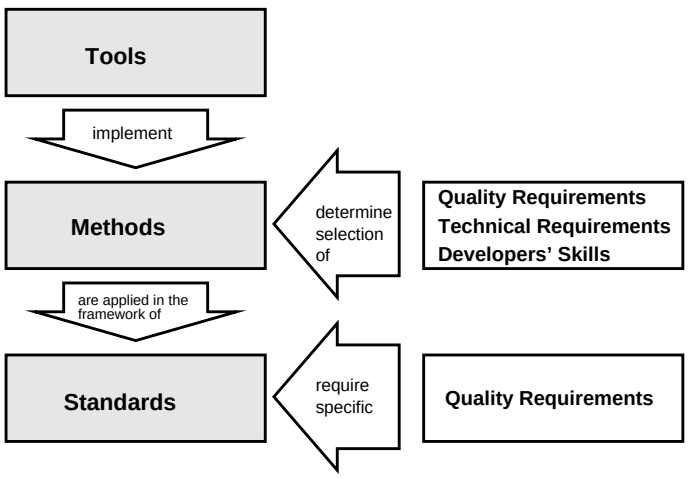

Figure 4.1: Relationship between standards, methods and tools.

Examples for standards with relevance in the field of dependable systems are

- **ISO 9000-3** [21] At present this is the most well known and at the same time the most general international standard providing rules with respect to organizational aspects of software projects and standardization and documentation of the complete software

development life cycle. Apart from rules concerning the process of software development itself, it emphasizes the regulation of *every* business process which may have a direct or indirect impact on the quality of software products developed in a company.

- **ITSEC (Information Technology Security Evaluation Criteria)** [44] This standard focuses on the security aspect of dependable systems. It is discussed in more detail in [74].
- **RTCA DO-178-B (Software Considerations in Airborne Systems and Equipment Certification)** [19] The international development standard for software in civil aircrafts. Apart from the general requirements regarding project management, configuration management and software development, its main issues are the selection and justification of validation, verification and test methods, safety aspects and reliability aspects. The CIDS system [1] discussed in [82] has been developed according to this standard.
- **V-Model *('Vorgehensmodell')*** [105] The standard for the development of software-based systems authorized by the German ministry of the interior.

Each of these standards is written in natural language. The objects and concepts used are informally defined and therefore subject to different interpretations. While each standard suggests the application of formal methods for the description and verification of the most critical system components, only few attempts have been made to show how the objects and concepts described in the standard should be related to the objects and concepts to be usually encountered when applying formal methods. With respect to the full scale of dependability issues, our approach to properly relate these different "worlds" appears to be a new enterprise. Similar investigations are currently carried out by Ravn and Stavridou [91, 92] with respect to the British Ministry of Defence Interim Standard MoD 00-55 [62]. In this chapter, we will discuss the regulations for software development according to the *V-Model*, because it may presently be regarded as one of the most advanced standards in Europe. Nevertheless, several adjustments of the standard will be made concerning the structure and the contents of the documents introduced by the *V-Model*. These changes will be introduced in Section 4.1.2 and justified while introducing the formal approach in Section 4.2.

In [74] a similar approach is described, this time focusing on security aspects only and referring to the ITSEC.

4.1.2 System Development According to the V-Model

Process Model

Similar to [21, 19], the *V-Model* adopts the well-accepted concept to model the whole system development procedure as a system of cooperating "meta" processes: Each team performs a well-defined task. In order to complete the task, inputs (documents, code, test data etc.) are received from the environment (e. g., the customer) and from other teams. After having processed the inputs, the corresponding outputs are distributed to the teams, to the customer and other external groups. The standard project structure comprises four cooperating processes: *Software Development Team, Project Management Team, Quality Assurance Team* and *Configuration Management Team*. In the following we will focus on

the tasks to be performed by the software development and the quality assurance teams in the context of the development of dependable systems.

Levels of Abstraction

To make the development of large and complex systems more feasible, they are described by means of a collection of documents describing different levels of abstraction and associating a modular structure with each level. In the *V-Model* five levels of modularization and abstraction are proposed for the full description of a system. They are informally introduced as *(1) System Level:* The top-level description of the whole target system without distinctions between software and hardware. Data and functions are described on the end user's level of abstraction. Interfaces to existing components in the environment (e. g., another computer) are described on the level of abstraction which is "seen" by the software to be developed for the target system. For example, if only application software has to be developed in a system layered according to the OSI model, the interface to an existing network is described on the level of abstraction offered by the send- and receive services provided by the OSI session layer for the application. *(2) Sub-System Level:* Large systems (e. g., a wide area network) may be decomposed into sub-systems (e. g., a local area network in the wide area network), in general without changing the level of abstraction for the representation of data, functions and interfaces. *(3) Segment Level:* This is a further modularization step which partitions (sub-)systems into well-defined components (e. g. one computer in a network). On this level the segments consisting of hardware only are separated from those to be built out of both hardware and software. *(4) SW Configuration Item Level:* The software to be allocated in a segment is decomposed into different configuration items, each item representing a software sub-system (e. g., the processes associated with a specific layer of the OSI model which will be allocated in the segment). This is usually the starting point from where to change also the level of abstraction for the representation of data and associated functions. *(5) Component Level:* A part of a configuration unit, performing a well-defined service (e. g., one task of the configuration unit). Several component layers may be "inserted" to refine a top-level component (e. g., tasks might be decomposed into threads). *(6) Module Level/Data Level:* This is the lowest level of abstraction and modularization to be distinguished according to the *V-Model* (e. g., a sequential function or a database table with associated attribute definitions).

Documents Associated With Each Level

The *V-Model* describes the documents to be associated with each level of abstraction and the topics to be covered by each document. In the context of this chapter two types of documents are of interest which appear (with slightly different naming and varying contents) on every level: *(1) Requirements Description:* This document type describes the functions, data and dynamic behaviour of an object associated with a specific level. In addition, boundary conditions restricting the class of possible realizations for the object are documented. *(2) Architecture Description:* This is a design structure which decomposes the object under consideration and/or refines its data structures. The process of decomposition introduces new objects to be associated with a lower level, as well as interfaces between them. Each new object is associated with its own lower-level requirements description. In this way, the alternation between requirements and architecture documents can be recursively applied to

the decomposition tree from system to module level.

To illustrate the contents of requirements and architecture documents, let us look at the system level.

System Requirements The system requirements document should describe *what* the system is supposed to do without telling us *how* this can be achieved. Specifically, the document describes the following aspects: *(1) Operational Environment:* The external interfaces and hypotheses about the normal behaviour of the environment. *(2) Application Requirements:* A specification of the functions, data and dynamic behaviour of the complete application, as seen by the end user. The application requirements describe this behaviour in relation to the operational environment (1). This means that other types of behaviour have to be expected, if the characteristics of the operational environment are changed. *(3) Threat Analysis (External):* A specification of those possible deviations from normal behaviour which are to be expected *in the environment* (e. g., crash of external computers, security attacks by unauthorized users from outside the system, corrupted data at an input interface). For each threat the list of dependability requirements it might impair is documented. Note that since at this stage nothing is known about the *internal* structure of the system, only *external* threats can be specified. *(4) Dependability Requirements:* A list associating items from the application requirements with safety, reliability/availability and security requirements. This describes the degree of protection to be established for each of these items in case of a deviation of the environment from normal behaviour. *(5) Risk Analysis (External):* An estimator associated with each threat, specifying the probability for the corresponding exceptional behaviour to occur.

This is not the complete list of items to be covered by the system requirements document, but the others described in [105] are not relevant in our context. It is more important to note that we have deviated from the description [105] in the following ways:

- According to the *V-Model*, security requirements and reliability requirements are described in different parts of the document, while safety requirements are subsumed under reliability and availability is not mentioned at all. I think that this is inappropriate, since it will be illustrated below that all dependability aspects can be covered by a unified approach. Therefore I have decided to introduce the general term *Dependability Requirements* where safety, reliability, availability and security issues should be described.
- The *V-Model* does not distinguish between external and internal threat/risk analysis. I regard this as an important aspect, because in general designers will have no influence on external threats, while they can influence the internal ones by means of their design decisions.

System Architecture The main topics to be covered by the system architecture document are *(1) System Structure:* The system is decomposed into sub-systems, segments or software configuration items. *(2) External Interfaces:* The external interfaces described in the system requirements document are allocated to the corresponding components of the system structure. *(3) Internal Interfaces:* The internal communication paths between the new components are defined. This gives rise to interface specifications for the requirements

documents associated with each component using the interface. *(4) Requirements Allocation:* Each application requirement is associated with one or more components of the system structure which have been designed to implement the requirement. This induces application requirements specifications for each component. *(5) Dependability Requirements Allocation:* The dependability requirements are mapped to corresponding components designed in the system structure. This induces dependability requirements for each of these components. *(6) Threat Analysis (Internal):* The decomposition introduces new interfaces and new components, therefore new threats may arise in addition to the external threats (e. g., component failure, security leak inside the system). *(7) Risk Analysis (Internal):* associated with the internal threat analysis. *(8) Dependability Concept:* It is explained how the system dependability requirements can be achieved by means of the chosen system structure in combination with the requirements defined for each component.

Here we have altered the suggestions of the *V-Model* in the following way:

- In [105] only the description of the security concept is required. Consistent with the change proposed for the system requirements, we will describe the complete dependability concept instead.
- The internal threat/risk analysis is missing in the *V-Model*. This also holds for the requirements and architecture documents associated with the lower levels of description. This seems to be an important aspect overlooked by the authors of [105]: Each design decision may introduce new threats to the dependability requirements. Therefore it is necessary to perform a new threat analysis at each level of decomposition.

4.2 A Formal Approach for the Development of Dependable Systems

4.2.1 Selection Criteria for the Formal Method

When choosing a suitable formal method for the development of dependable systems of the type introduced in the case study in Chapter 2, the following selection criteria have been considered:

- **Specification of distributed systems:** Since the problem belongs to the field of distributed systems, the underlying method should allow to specify aspects of concurrency, synchronization etc. in a suitable way.
- **Compositional and modular refinement:** The technique of describing the target system by means of stepwise decomposition, as suggested by the *V-model*, is related to refinement in a natural way. Moreover, there are two types of refinement that typically occur during the development process. They have been explored in detail by Zwiers [113]: If the designer has full freedom in the decomposition of a system component, being allowed to choose both the architecture and the specifications of the lower-level components created in the decomposition step, we speak of *compositional refinement*. If, on the other hand, existing components (e. g., off-the-shelf software) have to be used in the decomposition step we are confronted with pre-defined specifications and have to adjust the architecture and the specifications of new modules in

such a way that the pre-defined ones can be integrated. This situation is called *modular refinement*. Since most software projects require both the development of new components and the integration of existing ones, we require that a suitable method should allow both compositional and modular refinement.

- **Dependability features:** The formal method should allow to derive systematically the description of system behaviour in presence of (combinations of) threats.

These criteria led to the selection of CSP [41], enriched by Schepers' theory [99] for the characterization of fault hypotheses and their impact on distributed systems: The CSP process algebra has been explicitly designed to describe and verify distributed systems, several of its semantics definitions are compositional and the associated proof theories support both modular and compositional refinement. Schepers' approach introduces one of the few techniques allowing to model *combinations* of threats and their resulting impact on system behaviour in a formal way. An additional motivation for this selection was that the method and the tools for test automation in the field of safety-critical reactive systems described in [88] are also based on CSP. As a consequence, the formal specification and verification approach described here may be consistently used together with the test and validation techniques of [88].

Various denotational and operational semantics have been defined for CSP. The formalizations and results described below hold

1. in the *trace model* of CSP, as described in [41],
2. in the *failures-divergence model* of CSP, as described in [41],
3. in the *timed CSP model* in the extent as introduced in [99].

The first model is useful to describe safety aspects only. The second model may be used to specify and verify both safety and lifeness properties in the untimed case, and the last one should be applied when real-time aspects have to be explicitly considered as well.

In the sections to follow, we will associate formal notions with the system requirements and the system architecture, as defined by the *V-Model*. Moreover, it will be indicated how the formalization on system level may be continued in a recursive way to cover the requirements and architecture descriptions of lower levels as well. It might be helpful to read in parallel the case study in Section 5, because it will illustrate most of the formalizations introduced below.

4.2.2 System Requirements

In this section formalizations for the following items of the system requirements document will be presented:

- operational environment
- application requirements
- threat analysis (external)
- dependability requirements

The risk analysis is not covered in our approach, because it requires other (e. g., statistical) methods outside the scope of the formal methods applied here.

Operational Environment

To formalize the notion of an operational environment recall that a *context* is a CSP term $\mathcal{C}(X)$ with one free identifier X. Since the CSP operators $\rightarrow$, $\|$, $|||$, $\Box$, $\sqcap$, $;$, $\setminus$ used in a process term are continuous, every context can be regarded as a continuous mapping from CSP processes X to CSP processes $\mathcal{C}(X)$ (see [68, p.160]). In order to formalize the operational environment, we use a special context $\mathcal{E}(SYS)$ with the following interpretation:

1. The free identifier SYS denotes the *target system* to be "plugged into" the environment $\mathcal{E}$.
2. Every other process identifier E_i appearing in the term $\mathcal{E}(SYS)$ is an *environment process* which is considered relevant with respect to the operation of SYS, but is not a part of SYS itself.
3. The CSP operators used to define the term $\mathcal{E}(SYS)$ describe the *architecture* of the environment.

Each process parameter E_i in the term $\mathcal{E}(SYS)$ is associated with

- its alphabet $\alpha\, E_i$, as far as visible to the outside world,
- a specification E_i **sat** $S_{E_i}(h)$ representing the *normal behaviour* of E_i when the trace model of CSP is appropriate for specification and verification purposes,
- a specification E_i **sat** $S_{E_i}(h, R)$ representing the *normal behaviour* of E_i when the failures(-divergence) model of CSP is appropriate for specification and verification purposes.

We will use this behavioural specification style also in the other specifications to follow. In P **sat** $S(h, R)$, $S(h, R)$ is a predicate with free variables h and R. h denotes a trace which can be executed by P, and R is a refusal set of P/h. This holds for behavioural specifications in the failures-divergence and in the timed CSP model. If only properties about traces are described, specifications $S(h)$ are used, possessing only the trace parameter as free variable. As a consistency condition it is required that

$$ch(S(h, R)) \subseteq ch(P)$$

i. e., every channel referenced in the predicate $S(h, R)$ is also a channel of the process P.

To complete the specification of the operational environment, we will introduce a set

$$I \subseteq \bigcup_i \alpha(E_i)$$

denoting the *interface* between the environment and the target system SYS.

It is important to note that for the relevant parts of the environment, the system requirements specify behaviour *plus* architecture. Otherwise a threat analysis would become impossible, as shall be explained below.

Application Requirements

Our objective is to develop a system SYS subject to a behavioural specification $S_{SYS}(h, R)$ *when operating in the specified environment* $\mathcal{E}$. Therefore the application requirements are formalized as

$$\mathcal{E}(SYS) \textbf{ sat } S_{SYS}(h, R)$$

which is again interpreted as the *normal* system behaviour. The consistency conditions

$$I \subseteq \alpha(SYS)$$

$$ch(S_{SYS}(h, R)) \subseteq ch(SYS)$$

require that SYS must contain the pre-defined interface and its specification may only refer to SYS-events. If $ch(S_{SYS}(h, R)) \subseteq I$ the application requirements are called a *black box* specification, since they describe only the system behaviour visible at the interface.

Threat Analysis (External)

The objective of the external threat analysis is to specify the possible deviations from normal behaviour of the environment. In the CSP world, the only observable objects are traces and – in the failures-divergence model or for timed CSP – refusals. Therefore a threat can only refer to possible deviations from the specified characteristics of traces and refusals. To this end Schepers [99] has introduced assertions of the type

$$P \textbf{ sat } \Delta(h, h', R, R')$$

where Δ is a predicate with free variables h, h', R, R', to be called a *threat*[1]. In this notation (h, R) are interpreted as the trace/refusal pairs representing normal behaviour, whereas (h', R') denote the trace and refusal of the exceptional behaviour. Informally speaking, the threat $\Delta(h, h', R, R')$ is a predicate relating the exceptional behaviour to the normal behaviour. The following consistency conditions are required:

- Threats are reflexive relations with respect to (h, R): $\models \Delta(h, h, R, R)$
- Threats are prefix-closed with respect to the trace h':

$$\models (\Delta(h, h', \varnothing, \varnothing) \wedge s' \leq h') \Rightarrow (\exists\, s \bullet s \leq h \wedge \Delta(s, s', \varnothing, \varnothing))$$

- Threats are subset-closed with respect to refusals R':

$$\models (\Delta(h, h', R, R') \wedge U' \subseteq R') \Rightarrow (\exists\, U \bullet U \subseteq R \wedge \Delta(h, h', U, U'))$$

- The channels referenced in $\Delta(h, h', R, R')$ have to be channels of the process threatened by $\Delta(h, h', R, R')$.

Since – as demonstrated in the example of Chapter 2 – such a predicate can become very complex as soon as more than one threat is involved, it is useful to determine $\Delta(h, h', R, R')$ in a systematic stepwise procedure:

[1] In [99] the term *failure hypothesis* is used for $\Delta(h, h', R, R')$, since Schepers uses these predicates only for the description of exceptional behaviour in the context of fault tolerance. Failure hypotheses are typically denoted by $\chi(h, h', R, R')$ in [99].

Threat Analysis – Step 1: Definition of Isolated Threats For each environment process E defined by the context $\mathcal{E}$ and each isolated threat having impact on the process, set up a predicate specifying the acceptable behaviour of E *in presence of this isolated threat.* This results is a list of predicates

$$\Delta_{E,j}(h, h', R, R')$$

ranging over environment processes E and isolated threats indexed by j.

Threat Analysis – Step 2: Compositional Ordering of Threats In preparation of the next step, where the resulting overall threat shall be derived from the isolated threats, it is necessary to ensure that a certain *Compositional Ordering* of the threats can be obtained. A systematic derivation of the resulting threat will only be possible if the environment architecture suggests that the isolated threats $\Delta_{E,i}$ for an environment process E can be arranged in an order $\langle \Delta_{E,j_1}, \Delta_{E,j_2}, \Delta_{E,j_3}, \ldots \rangle$ such that each Δ_{E,j_ℓ} can be assumed to act on E *with the threats* $\langle \Delta_{E,j_1}, \ldots, \Delta_{E,j_{\ell-1}} \rangle$ *already present.* There seems to be no way to formalize the condition for such an ordering to exist, because no formal reference document is available on the system level from where such an ordering could be verified. As a consequence, it depends completely on the insight of the analysts how to find such an ordering and how to justify it in an intuitive way. If no compositional ordering of threats may be justified, the overall threat has to be derived from scratch without making use of isolated threat specifications.

Threat Analysis – Step 3: Derivation of the Resulting Threat If a compositional ordering $\langle \Delta_{E,j_1}, \Delta_{E,j_2}, \Delta_{E,j_3}, \ldots \rangle$ of threats could be found in step 2, the resulting threat may be derived by the *composition of threats* Δ_1, Δ_2 using the *threat introduction operator* $\wr$ defined by Schepers [99] as

$$(\Delta_1 \wr \Delta_2)(h, h', R, R') \equiv_{df} (\exists\, s, U \bullet \Delta_1(h, s, R, U) \wedge \Delta_2(s, h', U, R'))$$

$(\Delta_1 \wr \Delta_2)$ expresses the situation where the threat Δ_2 acts "on top of" threat Δ_1: Δ_1 refers to h representing normal behaviour and admits a deviation s. Then Δ_2 specifies exceptional behaviour h' *in reference to* s, which represents the exceptional behaviour caused by Δ_1.

Threat Analysis – Step 4: Derivation of Specifications in Presence of Threats After completion of Step 3, each process E of the operational environment is associated with

- a specification of *normal behaviour* E **sat** $S_E(h, R)$, as obtained in the operational environment specification,
- an *overall threat* $\Delta_E(h, h', R, R')$ associated with E and derived according to steps 1 to 3 of the threat analysis[2].

The last step of the threat analysis is devoted to the derivation of the specification of each environment process *in presence of its overall threat.* To this end, we can use the *threat*

[2] If E is not subject to exceptional behaviour, the threat can be assumed to be $\Delta_E(h, h', R, R') \equiv h = h' \wedge R = R'$.

introduction rule[3] given by Schepers [99, p. 84] for his proof theory for *failure prone processes*:

$$\frac{E \textbf{ sat } S_E(h,R)}{(E \wr \Delta_E) \textbf{ sat } (S_E \wr \Delta_E)(h',R')}$$

In this rule, $(E \wr \Delta_E)$ denotes the process capable of both normal behaviour as performed by E and exceptional behaviour as defined by Δ_E. In analogy to the composition of threats, the *composition of specifications* $S(h,R)$ *and threats* $\Delta(h,h',R,R')$ is defined by

$$(S \wr \Delta)(h',R') \equiv_{df} (\exists\, h,R \bullet S(h,R) \wedge \Delta(h,h',R,R'))$$

Note that since (h,R) are bound by the existential quantifier, this is again an "ordinary" specification with only one pair (h',R') of free variables.

Example 4.1 To provide an example where the overall threat cannot be represented by a compositional ordering of the isolated ones, let us assume that[4]

$$\begin{aligned}
&h = \langle c.1, c.1, c.1, \ldots \rangle \\
&\Delta_1(h,h') \equiv_{df} (\forall\, i : 1 \,..\, \#h' \bullet h'(i) = h(i) \vee val(h'(i)) = val(h(i)) + 2) \\
&\Delta_2(h,h') \equiv_{df} (\forall\, i : 1 \,..\, \#h' \bullet h'(i) = h(i) \vee val(h'(i)) = 4 * val(h(i)))
\end{aligned}$$

but the overall threat is

$$\begin{aligned}
\Delta_0(h,h') \equiv_{df} (\forall\, i : 1 \,..\, \#h' \bullet\ & h'(i) = h(i) \vee \\
& val(h'(i)) = val(h(i)) + 2 \vee val(h'(i)) = 4 * val(h(i)))
\end{aligned}$$

Then we have

$$\begin{aligned}
(\Delta_1 \wr \Delta_2)(h,h') \equiv (\forall\, i : 1 \,..\, \#h' \bullet\ & h'(i) = 1 \vee \\
& val(h'(i)) = 3 \vee val(h'(i)) = 4 \vee val(h'(i)) = 12) \\
(\Delta_2 \wr \Delta_1)(h,h') \equiv (\forall\, i : 1 \,..\, \#h' \bullet\ & h'(i) = 1 \vee \\
& val(h'(i)) = 3 \vee val(h'(i)) = 4 \vee val(h'(i)) = 6)
\end{aligned}$$

and thus neither $(\Delta_1 \wr \Delta_2)(h,h')$ nor $(\Delta_2 \wr \Delta_1)(h,h')$ are equivalent to $\Delta_0(h,h')$
□

Dependability Requirements

The informal definition of dependability requirements given by the *V-Model* may be rephrased as a specification describing the *maximum deviation from normal behaviour* which may be tolerated for certain objects of the application. As pointed out above, these objects may only be traces and refusals in the CSP context, and we may use the same formalism as in the threat analysis above: A dependability requirement is a threat predicate

$$\Delta_{SYS}(h,h',R,R')$$

satisfying

$$ch(\Delta_{SYS}(h,h',R,R')) \subseteq ch(SYS)$$

and interpreted as

[3] This rule is called *failure hypothesis introduction* in [99].

[4] For any event $c.x$ with channel c and value x, $val(c.x) = x$ denotes the projection on the value.

> *Even in presence of exceptional environment behaviour or exceptional internal behaviour, the specification* $(S_{SYS} \wr \Delta_{SYS})(h', R')$ *has still to be observed.*

Summary of the System Requirements

Before tackling the system architecture document, let's summarize what has been achieved during the development of system requirements. At first, the following activities were performed:

1. definition of a *context* $\mathcal{E}(X)$ describing the architecture of the environment, as far as relevant for the development of the target system,
2. specifications E **sat** $S_E(h, R)$ of the *normal behaviour of environment processes* E defined by the context $\mathcal{E}(X)$,
3. specification $\mathcal{E}(SYS)$ **sat** $S_{SYS}(h, R)$ of the *normal system behaviour*,
4. specifications of (lists of) *isolated threats* $\Delta_{E,j}(h, h', R, R')$ for the environment processes E introduced by $\mathcal{E}(X)$,
5. specification of *dependability requirements* $\Delta_{SYS}(h, h', R, R')$ describing the acceptable deviations of the target system from normal behaviour, as far as visible at the system interface.

In this list the items (1.) to (3.) will be present in *any* system development procedure. Only items (4.) and (5.) are specific for the development of *dependable* systems. The inputs (1.) to (5.) were transformed into

1. specifications $(E \wr \Delta_E)$ **sat** $(S_E \wr \Delta_E)(h', R')$ of the *acceptable behaviour of environment processes* E defined by the context $\mathcal{E}(X)$
2. a modified context $(\mathcal{E} \wr \Delta)(X) = \mathcal{E}(X)[E/(E \wr \Delta_E), \ldots]$, where every process parameter E used in $\mathcal{E}(X)$ has been replaced by the process $(E \wr \Delta_E)$ showing acceptable behaviour in presence of threats
3. a predicate $(S_{SYS} \wr \Delta_{SYS})(h', R')$ defining the *acceptable system behaviour* in presence of all possible threats. This specification summarizes the normal behaviour specification plus the dependability requirements

Analysis of $(E \wr \Delta_E)$, $(\mathcal{E} \wr \Delta)(X)$ and $(S_{SYS} \wr \Delta_{SYS})(h', R')$ shows:

> *The obligations associated with the system requirements for a dependable systems can be transformed into "ordinary" design obligations, containing the results of the threat analysis and the dependability requirements as integral parts of the behavioural specifications for environment processes and target system.*

Note that in our approach the threats have *not* been associated with the corresponding dependability requirements, as it is suggested in the *V-Model*: At the present stage of the development, the relation between the objects to be protected and the objects threatened by the environment is not clear because the architecture document is still missing. Therefore my suggestion is to describe the relation between threats and dependability requirements in the architecture document.

4.2.3 System Architecture

We will now describe the formal treatment of the items to be covered by the system architecture document, as has been informally defined by the *V-Model*. We claim that only the items

- System Structure
- Interfaces
- Internal Threat Analysis
- Internal Risk Analysis

have any formal relevance for the development and verification process. The *Correspondence between Application Requirements and Architecture* and the *Dependability Concept* only serve to illustrate the system concept, as will be made clear in the following paragraphs. As in the description of the system requirements document I will skip the *Internal Risk Analysis*.

System Structure

The target system SYS is decomposed into an *implementable* collection of cooperating processes. Formally speaking, the system structure is a CSP equation

$$SYS = \mathcal{A}_{SYS}(P_1, \ldots, P_n)$$

where $\mathcal{A}_{SYS}$ is a continuous function mapping processes $(P_1, \ldots, P_n)$ to a new CSP process by making use of the CSP operators. The processes P_i are components of a lower level, e. g., sub-systems. However, the terms *sub-system, segment, SW-configuration item* etc. do not have any formal meaning; their objective is only to aid the intuition of the reader trying to understand the specification document.

Interfaces

In general, the definition of the system structure will also introduce new channels, whose channel alphabets constitute the internal interface descriptions. Internal plus external interfaces are completely defined by the alphabets $\alpha(P_i \setminus L_{P_i})$, denoting the events visible at the interface of each process, while L_{P_i} denotes the set of internal P_i-events. Since one of the objectives of the architecture document is to completely establish the interface between target system and environment, we demand that the interface set I should be contained in the component alphabets, i. e.,

$$I \subseteq \bigcup_i \alpha(P_i)$$

Component Specifications

Each lower-level component P_i (sub-system, segment etc.) has to be equipped with a specification of its *normal* behaviour,

$$P_i \textbf{ sat } S_{P_i}(h, R)$$

This specification will be located in the component requirements document associated with P_i. However, it has to be developed during the definition of the system architecture, because otherwise the architecture cannot be verified against the system requirements.

Threat Analysis (Internal)

As is well-known in the context of dependable systems, a general refinement step may introduce new *internal threats* that could not be captured on system requirements level, *because they completely depend on the architecture selected in the refinement step.* As a consequence a new threat analysis and associated risk analysis have to be performed after the definition of the system structure. Though specific boundary conditions ensuring that dependability will be preserved under refinement are of great interest, especially in applications to security (see [96]), we think that in general it has to be accepted that the selection of a specific architecture may introduce new threats and lead to a "feed-back loop" between threat analysis and revision of refinement decisions.

The distinction between external and internal threat analysis is justified by the fact that the external threats are caused by the environment processes whose acceptable behaviour cannot be influenced by the system designer, whereas internal threats may be altered by means of architectural decisions.

Fortunately, the internal threat analysis may be treated in analogy to the external threat analysis described above: Starting with an association of isolated threats $\Delta_{P_i,j}$ for each process P_i, the threat analysis will derive specifications

$$(P_i \backslash \Delta_{P_i}) \textbf{ sat } (S_{P_i} \backslash \Delta_{P_i})(h, R)$$

which again constitute ordinary specifications of the acceptable behaviour of each P_i.

Dependability Concept

All information needed to verify the correctness of the architecture decisions is contained in the definition of $\mathcal{A}_{SYS}$ and the specifications of the lower-level components P_i including internal threats. As a consequence, the dependability concept will not add anything to the architecture document which is essential for specification or verification purposes. It is good design practice, however, to "separate concerns" in specifications. If a global specification of $\mathcal{C}((P_1 \backslash \Delta_{P_1}), \ldots, (P_n \backslash \Delta_{P_n}))$ can be separated into a conjunction of predicates $App(h, R)$ and $Dep(h, R)$ and the assertions expressed by $Dep(h, R)$ can be shown to be responsible for the preservation of dependability requirements while $App(h, R)$ expresses properties of the application system, then $Dep(h, R)$ will be called the *(formal) dependability policy* of the system.

Requirements Allocation

This is a means to *trace* the system requirements in the system architecture. In practice it is achieved by maintaining tables listing requirements features and associated components in the architecture which have been intended to implement the requirement. This is only an informal aid, since our formal approach offers the opportunity to *prove* that the architectural decisions have been correct, as can be seen in the next section. In the context of formal methods, a requirements allocation document would be a table listing the premises for the proofs which are necessary to demonstrate that the system decomposition designed really implies the requirements specification.

4.2.4 Verification on System Level

We wish to prove that the system structure selected is appropriate for the global development task in presence of external and internal threats. To this end, two things have to be shown:

1. If the lower-level processes perform according to their *normal* behaviour specification, their cooperation according to $\mathcal{A}_{SYS}$ results in a behaviour consistent with the *normal* behaviour specification *on system level.*
2. If the lower-level processes perform according to their *acceptable* behaviour specification, their cooperation according to $\mathcal{A}_{SYS}$ results in a behaviour consistent with the *acceptable* behaviour specification *on system level.*

At first glance, condition (1.) might seem superfluous, because the formal definition of threats Δ requires that normal behaviour always satisfies the threat predicate, that is, $(h, R) \in \mathit{Fail}(P) \models \Delta(h, h, R, R)$. However, this requirement does not guarantee that the normal behaviour will really be *implemented* in the system: An implementation only showing *exceptional* behaviour would satisfy requirement (2.), but certainly not be consistent with the customer's expectations.

Formally speaking, these considerations result in the following proof obligations:

1. System Verification Obligation – Normal Behaviour:

$$\frac{(\forall\, i \bullet P_i \textbf{ sat } S_{P_i}(h, R))\quad (\forall\, j \bullet E_j \textbf{ sat } S_{E_j}(h, R))}{\mathcal{E}(\mathcal{A}_{SYS}(P_1, \ldots, P_n) \textbf{ sat } S(h, R)}$$

2. System Verification Obligation – Acceptable Behaviour:

$$\frac{(\forall\, i \bullet (P_i \wr \Delta_{P_i}) \textbf{ sat } (S_{P_i} \wr \Delta_{P_i})(h, R))\quad (\forall\, j \bullet (E_j \wr \Delta_{E_j}) \textbf{ sat } (S_{E_j} \wr \Delta_{E_j})(h, R))}{(\mathcal{E} \wr \Delta)(\mathcal{A}_{SYS}((P_1 \wr \Delta_{P_1}), \ldots, (P_n \wr \Delta_{P_n}))) \textbf{ sat } (S \wr \Delta_{SYS})(h, R)}$$

In proof obligation (2.), $(\mathcal{E} \wr \Delta)(X) = \mathcal{E}(X)[E_1/(E_1 \wr \Delta_{E_1}), \ldots]$, where $[E_1/(E_1 \wr \Delta_{E_1}), \ldots]$ denotes the substitution of every environment process E_i showing only normal behaviour by its associated process $(E_i \wr \Delta_{E_i})$ showing acceptable behaviour.

Observe that on system level, we are confronted with the obligation to verify a *mixed modular and compositional refinement step.* The environment processes which cannot be altered according to the designer's preferences introduce modular refinement obligations. Compositional refinement, on the other hand, applies for the processes P_i created during the architecture design for the target system.

Furthermore, note that also the system verification obligation for acceptable behaviour is an ordinary proof obligation, where all references to dependability issues have been transformed into ordinary requirements during the threat analysis. Therefore the rules for the standard compositional proof theory of CSP can be applied to verify this assertion. Observe, however, that the creative process of finding suitable components P_i crucially depends on the existence

of internal threats: If internal threats do not have to be taken into account, the system verification obligation for acceptable behaviour represents a standard proof obligation according to the *invent and verify paradigm*:

> *Given* $(E_j \wr \Delta_{E_j})$ ***sat*** $(S_{E_j} \wr \Delta_{E_j})(h', R')$, $(\mathcal{E} \wr \Delta)(X)$ *and* $(S \wr \Delta_{SYS})(h, R)$ *as defined in the system requirements document, invent an architecture* $\mathcal{A}_{SYS}$ *and component specifications* P_i ***sat*** $S_{P_i}(h, R)$*, such that*
>
> $$(\mathcal{E} \wr \Delta)(\mathcal{A}_{SYS}(P_1, \ldots, P_n)) \textbf{ sat } (S \wr \Delta_{SYS})(h, R)$$
>
> *holds.*

The freedom to invent $\mathcal{A}_{SYS}$ and the specifications of P_i is severely restricted, if internal threats are present: Now the *invent and verify paradigm* has to be re-phrased as:

> *Given* $(E_j \wr \Delta_{E_j})$ ***sat*** $(S_{E_j} \wr \Delta_{E_j})(h', R')$, $(\mathcal{E} \wr \Delta)(X)$ *and* $(S \wr \Delta_{SYS})(h, R)$ *as defined in the system requirements document, invent an architecture* $\mathcal{A}_{SYS}$ *and component specifications* P_i ***sat*** $S_{P_i}(h, R)$*, such that, taking into account the threats* Δ_{P_i} *induced by* $\mathcal{A}_{SYS}$*,*
>
> $$(\mathcal{E} \wr \Delta)(\mathcal{A}_{SYS}((P_1 \wr \Delta_{P_1}), \ldots, (P_n \wr \Delta_{P_n}))) \textbf{ sat } (S \wr \Delta_{SYS})(h, R)$$
>
> *holds.*

4.2.5 Recursive Application of the Development Procedure

After having established the system architecture with its corresponding component specifications, the approach can be recursively applied in top-down fashion:

1. For a given sub-system or lower-level component P and an associated specifications P **sat** $S_P(h, R)$ for normal behaviour and $(P \wr \Delta)$ **sat** $(S_P \wr \Delta)(h, R)$ for acceptable behaviour develop a component architecture $\mathcal{A}_P(P_1, \ldots, P_k)$.
2. Associate specifications of normal behaviour for each P_i: P_i **sat** $S_{P_i}(h, R)$.
3. Analyzing the architecture chosen for P, perform an internal threat analysis and derive the resulting overall threat Δ_{P_i} for each P_i.
4. Using the threat introduction rule, derive the specification of acceptable behaviours $(P_i \wr \Delta_{P_i})$ **sat** $(S_{P_i} \wr \Delta_{P_i})(h, R)$.
5. Prove the

   1. Component Verification Obligation – Normal Behaviour:

   $$\frac{(\forall\, i \bullet P_i \textbf{ sat } S_{P_i}(h, R))}{\mathcal{A}_P(P_1, \ldots, P_k) \textbf{ sat } S_P(h, R)}$$

   otherwise improve the architecture chosen for P or revise the requirements for the sub-components, P_i **sat** $S_{P_i}(h, R)$.

6. Prove the

   2. Component Verification Obligation – Acceptable Behaviour:

$$\frac{(\forall\, i \bullet (P_i \wr \Delta_{P_i})\ \textbf{sat}\ (S_{P_i} \wr \Delta_{P_i})(h, R))}{\mathcal{A}_P((P_1 \wr \Delta_{P_1}), \ldots, (P_k \wr \Delta_{P_k}))\ \textbf{sat}\ (S_P \wr \Delta)(h, R)}$$

   otherwise improve the architecture chosen for P or revise the requirements for the sub-components, P_i **sat** $S_{P_i}(h, R)$.

7. Continue this process for each P_i until a sufficient level of decomposition has been reached.

In this stepwise refinement procedure, steps (3.) and (4.) are needed to transform the development obligation involving dependability issues into an "ordinary" development obligation. They are only necessary if the refinement step forces the designer to consider new dependability threats.

Note that from sub-system level downward only compositional refinement is needed, as long as the designer does not have to integrate re-usable off-the-shelf software at a certain refinement level.

5. Combination of Dependability Mechanisms — Case Study Part II

We will now develop a revised version of the case study introduced in Section 3. As before it will be assumed that the target system to be developed is only the fault-tolerant and secure network layer. Producer *PROD*, consumer *CON* and the transmission media are therefore considered to be objects located in the environment, as depicted in Figure 5.1. However, this time a more realistic setting will be assumed: We will use two instances of the transmission media; M will carry application data and *MACK* will transmit control messages to acknowledge reception of application data. Both M and *MACK* are subject to the following assumptions:

1. **Fault hypothesis:** The transmission media may lose messages. The number of consecutive input messages that each medium may lose is bound by $maxLoss \geq 0$.
2. **Security Threat:** A malicious agent X can perform eavesdropping and tamper with messages both on M and *MACK*. Making use of tampering, X can also replay messages by "copying" an old message onto a new package. X cannot block or delay messages or fake the identity of a sender. On each medium, X can modify at most $maxModified \geq 0$ messages in a row.
3. Neither M nor *MACK* change the ordering of the messages.

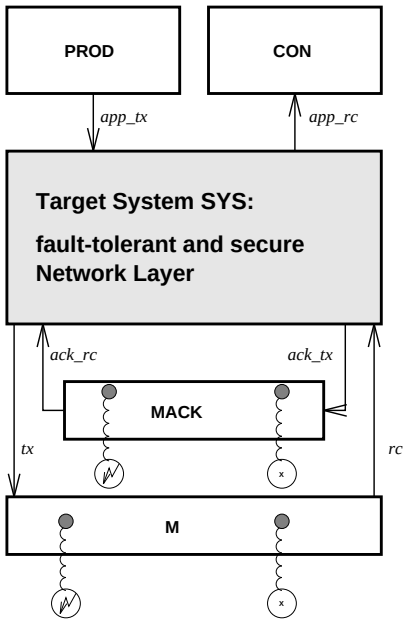

Figure 5.1: System configuration for the case study, part II

The presentation will be structured according to the formal framework given in the previous section. We will first use the trace model of CSP to specify and prove safety properties based

on behavioural specifications of type P **sat** $S_P(h)$. After having reached the final level of decomposition, a separate argument will be presented allowing to prove liveness properties by means of automated model checking. I regard this approach as a promising alternative to the simultaneous processing of safety and lifeness properties in the failures-divergence model which is frequently used in CSP applications. The latter technique has been applied by the author in [71].

5.1 System Requirements

5.1.1 Operational Environment

Following the configuration sketched in Figure 5.1 the system context may be specified by

$$\mathcal{E}(SYS) = (PROD \,|||\, CON \,|||\, M \,|||\, MACK) \,\|\, SYS$$

where the normal behaviour of the environment processes satisfies[1]

E	$\alpha(E)$	E **sat** $S_E(h)$
$PROD$	$\{\!\mid app_tx \mid\!\}$	$true$
CON	$\{\!\mid app_rc \mid\!\}$	$true$
M	$\{\!\mid tx, rc \mid\!\}$	$val^*(h \upharpoonright \{\!\mid rc \mid\!\}) \leq^1 val^*(h \upharpoonright \{\!\mid tx \mid\!\})$
$MACK$	$\{\!\mid ack_tx, ack_rc \mid\!\}$	$val^*(h \upharpoonright \{\!\mid ack_rc \mid\!\}) \leq^1 val^*(h \upharpoonright \{\!\mid ack_tx \mid\!\})$

$\mathcal{E}(SYS)$ introduces an environment architecture where each environment process is completely independent of the others, but operates in parallel with the target system. Therefore the system interface consists of every channel visible in the environment:

$$I = \{app_tx, app_rc, tx, rc, ack_tx, ack_rc\}$$

Obviously, $ch(S_E(s)) \subseteq ch(E)$ is fulfilled for every specification, since the free trace variable h only appears filtered to the channels of the corresponding processes. Thus the consistency condition for the interface (p. 28) are fulfilled.

5.1.2 Application Requirements

The application requirements may then be formally defined as follows[2]:

$$\alpha\,(SYS \setminus L_{SYS}) = \{\!\mid app_tx, app_rc, tx, rc, ack_tx, ack_rc \mid\!\}$$

$$\mathcal{E}(SYS) \textbf{ sat } S_{SYS}(h)$$

$$S_{SYS}(h) \equiv_{df} val^*(h \upharpoonright \{\!\mid app_rc \mid\!\}) \leq^N val^*(h \upharpoonright \{\!\mid app_tx \mid\!\})$$

[1] For traces s, u, the notation $s \leq^n u$ is used to denote that s is a prefix of u with $\#u - \#s \leq n$.

[2] We use the notation $val^*(h)$ to denote the trace derived from h by projecting each $h(i)$ onto its value component $val(h(i))$.

where $N > 0$ is an unspecified but fixed natural number. $\alpha\,(SYS \setminus L_{SYS})$ is the alphabet of the interface of SYS visible to the environment, where L_{SYS} denotes the set of local SYS-events which will become visible during the decomposition steps to follow.

$S_{SYS}(h)$ requires $(SYS \setminus L_{SYS})$ to input values on *app_tx* and output them on channel *app_rc* without changing their order or value. The constant N in the prefix relation ensures that each output may be expected within a bounded number of consecutive inputs. The fixed upper bound N is appropriate in the context of real-time applications, where the assertion that a message will "finally" reach its destination is not useful at all, because this may still admit failures in the time domain.

Since $I = ch(S_{SYS}(h))$ we are dealing with a black box specification. SYS is obviously underspecified by $S_{SYS}(h)$. For example, it does not show that the transmission media should be used for the purpose of data transfer between $PROD$ and CON. However, this is an architectural issue and will therefore be covered in the system architecture described below. The consistency conditions relating the interface to the channels of SYS (p. 29) are obviously fulfilled.

5.1.3 Threat Analysis (External)

Threat Analysis – Step 1: Definition of Isolated Threats According to the hypotheses informally described above, we are confronted with a reliability threat and a security threat present for both transmission lines M and $MACK$. The security threat can be specified as follows[3]:

$$
\begin{array}{l}
\Delta_{M,1}(h,h') \equiv_{df} \\
\quad ch^*(h' \upharpoonright \{\!|\ tx, rc\ |\!\}) = ch^*(h \upharpoonright \{\!|\ tx, rc\ |\!\}) \\
\quad \wedge \\
\quad h' \upharpoonright \{\!|\ tx\ |\!\} = h \upharpoonright \{\!|\ tx\ |\!\} \\
\quad \wedge \\
\quad (\forall\, i : 1\,..\,(\#(h' \upharpoonright \{\!|\ rc\ |\!\}) - maxModified) \bullet \\
\qquad\qquad (\exists\, j : i\,..\,(i + maxModified) \bullet (h' \upharpoonright \{\!|\ rc\ |\!\})(j) = (h \upharpoonright \{\!|\ rc\ |\!\})(i))) \\
\quad \wedge \\
\quad (\mathrm{ran}\ val^*(h' \upharpoonright \{\!|\ tx\ |\!\}) \subseteq \mathrm{ran}\ \mathsf{e} \Rightarrow \\
\qquad\qquad (\forall\, x : \mathrm{ran}\ val^*(h' \upharpoonright \{\!|\ rc\ |\!\}) \bullet x \notin \mathrm{ran}\,\mathsf{e} \vee x \in \mathrm{ran}\ val^*(h' \upharpoonright \{\!|\ tx\ |\!\}))
\end{array}
$$

In presence of the isolated security threat, M will allow the same inputs as in the case of normal behaviour and preserve the number of outputs, as well as the ordering between inputs and outputs. However, out of $maxModified + 1$ outputs only one can be guaranteed to be unmodified. The last term of the conjunction is a security hypothesis expressing the possibilities to detect modified messages by means of encryption. e denotes an encryption function, whose existence is assumed in the hypothesis, such that we may depend on the following properties: If the values on channel *tx* are encrypted by means of e (i. e., every $(h \upharpoonright \{\!|\ tx\ |\!\})$-value is contained in the range of e), then a malicious agent only has two possibilities to modify the messages, before they reach channel *rc*. Either a replay is performed (i. e.,

[3] $ch^*(h)$ denotes the trace derived from h by projecting each $h(i)$ onto its channel $ch(h(i))$. ran h is the set of events contained in h.

the modified value is contained in the set of messages already transmitted), or the modified value is not contained in the range of e.

Given any encryption function f, e may be implemented by enciphering the message plus a checksum by means of f: If it is safe to assume that agents have no possibility to decipher the message and encipher an altered message with corresponding checksum, then it is also safe to assume that any modification of the enciphered message will be detected by means of the mismatch between modified message and checksum. More sophisticated encryption mechanisms allowing to detect tampered messages are provided by the *public-key mechanisms*, described for example in [90]. In practice, any reasonably secure e will require the distribution of *encryption keys* between producer and consumer. A formal treatment of key distribution protocols is given in [9].

According to the definition of threats, $\Delta_{M,1}(h, h')$ characterizes *acceptable behaviour*. This is expressed by the hypotheses about e, assuming that an agent may only tamper with messages to the extent specified by the fourth conjunct in $\Delta_{M,1}(h, h')$. If an agent might "crack" the encryption code for e and alter messages, such that the result would still look like a valid encryption, this would represent *catastrophic behaviour*, and our protocols to be introduced and verified below would produce unpredictable results.

An analogous threat $\Delta_{MACK,1}(h, h')$ is defined for $MACK$.

The second threat present for M is the hypothesis assuming the possibility of data losses[4]:

$$
\begin{array}{l}
\Delta_{M,2}(h, h') \equiv_{df} \\
\quad h' \restriction \{\!|\ tx, rc\ |\!\} \trianglelefteq h \restriction \{\!|\ tx, rc\ |\!\} \\
\quad \wedge \\
\quad h' \restriction \{\!|\ tx\ |\!\} = h \restriction \{\!|\ tx\ |\!\} \\
\quad \wedge \\
\quad (\#(h \restriction \{\!|\ rc\ |\!\}) > maxLoss \Rightarrow \\
\qquad \#(h' \restriction \{\!|\ rc\ |\!\}) > 0 \\
\qquad \wedge \\
\qquad (\exists\, h^0, h^1, \ldots, h^{\#(h' \restriction \{|rc|\})} : \{\!|\ rc\ |\!\}^* \bullet \\
\qquad\quad \#h^0 \leq maxLoss \wedge h^0 \frown \ldots \frown h^{\#(h' \restriction \{|rc|\})} = h \restriction \{\!|\ rc\ |\!\} \\
\qquad\quad \wedge \\
\qquad\quad (\forall\, i : 1 \,..\, \#(h' \restriction \{\!|\ rc\ |\!\}) \bullet \\
\qquad\qquad (h' \restriction \{\!|\ rc\ |\!\})(i) = head(h^i) \wedge \#h^i \leq maxLoss + 1)))
\end{array}
$$

Again, h' allows the same inputs on $\{\!|\ tx\ |\!\}$ as h, but this time it outputs only a subtrace of $(h \restriction \{\!|\ rc\ |\!\})$. The number of consecutive data losses in limited by $maxLoss$. Let $\Delta_{MACK,2}(h, h')$ be analogously defined for $MACK$. It is trivial to see that $\Delta_{M,j}(h, h')$ and $\Delta_{MACK,j}(h, h')$ are reflexive and prefix-closed, and the full set of consistency conditions (p. 29) for threats holds.

Threat Analysis – Step 2: Compositional Ordering of Threats In the network example a possible ordering of threats can be found along the following lines:

[4] $h' \trianglelefteq h$ denotes the *subtrace* relation: Every h'-event is contained in h, and the ordering of h' is also preserved in h.

1. If it is reasonable to assume that the malicious agent attacks directly "behind" the input channel tx of M, then the ordering $\langle \Delta_{M,1}, \Delta_{M,2} \rangle$ is justified: *First* a package is possibly corrupted by the agent, *then* it may still be lost by the unreliable network.
2. If it is reasonable to assume that the agent gets a chance to attack directly before M outputs on channel rc, the reverse ordering $\langle \Delta_{M,2}, \Delta_{M,1} \rangle$ is justified: The attacker will now only tamper with packages having already been securely transmitted over the medium.
3. If it cannot be predicted whether the agent may attack at the beginning or at the end of the transmission, it is not possible to make use of the isolated threats $\Delta_{M,1}, \Delta_{M,2}$. If it can be guaranteed, however, that at least $n\%$ of the messages are neither lost nor corrupted, a global threat can be defined as

$$
\begin{array}{l}
\Delta_{M,0}(h, h') \equiv_{df} \\
\quad ch^*(h' \upharpoonright \{\!|\, tx, rc \,|\!\}) \preceq ch^*(h \upharpoonright \{\!|\, tx, rc \,|\!\}) \\
\quad \wedge \\
\quad h' \upharpoonright \{\!|\, tx \,|\!\} = h \upharpoonright \{\!|\, tx \,|\!\} \\
\quad \wedge \\
\quad (\exists\, h'' \bullet (h'' \upharpoonright \{\!|\, tx, rc \,|\!\}) \preceq (h' \upharpoonright \{\!|\, tx, rc \,|\!\}) \wedge \\
\qquad\qquad h'' \upharpoonright \{\!|\, tx \,|\!\} = h' \upharpoonright \{\!|\, tx \,|\!\} \wedge \\
\qquad\qquad \#(h' \upharpoonright \{\!|\, tx \,|\!\}) * n/100 \leq \#(h'' \upharpoonright \{\!|\, rc \,|\!\}) \wedge \\
\qquad\qquad h'' \upharpoonright \{\!|\, rc \,|\!\} \preceq h'' \upharpoonright \{\!|\, tx \,|\!\}) \\
\quad \wedge \\
\quad (\mathrm{ran}\ val^*(h' \upharpoonright \{\!|\, tx \,|\!\}) \subseteq \mathrm{ran}\ \mathsf{e} \Rightarrow \\
\qquad\qquad (\forall\, x : \mathrm{ran}\ val^*(h' \upharpoonright \{\!|\, rc \,|\!\}) \bullet x \notin \mathrm{ran}\, \mathsf{e} \vee x \in \mathrm{ran}\ val^*(h' \upharpoonright \{\!|\, tx \,|\!\}))
\end{array}
$$

Threat Analysis – Step 3: Derivation of the Resulting Threat Assuming the ordering $\langle \Delta_{M,1}, \Delta_{M,2} \rangle$ of threats results – after having eliminated the bound variable s – in the overall threat

$$
\begin{array}{l}
(\Delta_{M,1} ; \Delta_{M,2})(h, h') \equiv_{df} \\
\quad ch^*(h' \upharpoonright \{\!|\, tx, rc \,|\!\}) \preceq ch^*(h \upharpoonright \{\!|\, tx, rc \,|\!\}) \\
\quad \wedge \\
\quad h' \upharpoonright \{\!|\, tx \,|\!\} = h \upharpoonright \{\!|\, tx \,|\!\} \\
\quad \wedge \\
\quad (\#(h \upharpoonright \{\!|\, rc \,|\!\}) > maxLoss \Rightarrow \\
\qquad \#(h' \upharpoonright \{\!|\, rc \,|\!\}) > 0 \wedge \\
\qquad (\exists\, h^0, h^1, \ldots, h^{\#(h' \upharpoonright \{|rc|\})} : \{\!|\, rc \,|\!\}^* \bullet \#h^0 \leq maxLoss \wedge \\
\qquad\quad (\forall\, i : 1 \,..\, (\#(h \upharpoonright \{\!|\, rc \,|\!\}) - maxModified) \bullet \\
\qquad\qquad (\exists\, j : i \,..\, (i + maxModified) \bullet \\
\qquad\qquad\quad (h \upharpoonright \{\!|\, rc \,|\!\})(j) = (h^0 \frown \ldots \frown h^{\#(h' \upharpoonright \{|rc|\})})(j)) \wedge \\
\qquad\qquad (\forall\, k : 1 \,..\, \#(h' \upharpoonright \{\!|\, rc \,|\!\}) \bullet \\
\qquad\qquad\quad (h' \upharpoonright \{\!|\, rc \,|\!\})(k) = head(h^k) \wedge \#h^k \leq maxLoss + 1)))) \\
\quad \wedge \\
\quad (\mathrm{ran}\ val^*(h' \upharpoonright \{\!|\, tx \,|\!\}) \subseteq \mathrm{ran}\ \mathsf{e} \Rightarrow \\
\qquad\qquad (\forall\, x : \mathrm{ran}\ val^*(h' \upharpoonright \{\!|\, rc \,|\!\}) \bullet x \notin \mathrm{ran}\, \mathsf{e} \vee x \in \mathrm{ran}\ val^*(h' \upharpoonright \{\!|\, tx \,|\!\}))
\end{array}
$$

For $maxModified > 0 \wedge maxLoss > 0$, this predicate admits traces h' containing only modified values on output channel $\{\!|\ rc\ |\!\}$: Choose j such that $(h \upharpoonright \{\!|\ rc\ |\!\})(j)$ never coincides with $head(h^k)$. As a consequence it is impossible to construct a fault-tolerant and secure network service for the medium under the hypothesis $(\Delta_{M,1} \wr \Delta_{M,2})$[5].

Since $\wr$ reflects the compositional ordering of threats, it is not symmetric. Indeed, assuming the ordering $\langle \Delta_{M,2}, \Delta_{M,1} \rangle$, the overall threat derived is

$$
\begin{array}{l}
(\Delta_{M,2} \wr \Delta_{M,1})(h, h') \equiv_{df} \\
\quad ch^*(h' \upharpoonright \{\!|\ tx, rc\ |\!\}) \trianglelefteq ch^*(h \upharpoonright \{\!|\ tx, rc\ |\!\}) \\
\quad \wedge \\
\quad h' \upharpoonright \{\!|\ tx\ |\!\} = h \upharpoonright \{\!|\ tx\ |\!\} \\
\quad \wedge \\
\quad (\#(h \upharpoonright \{\!|\ rc\ |\!\}) > maxLoss \Rightarrow \\
\qquad \#(h' \upharpoonright \{\!|\ rc\ |\!\}) > 0 \wedge \\
\qquad (\exists\, h^0, h^1, \ldots, h^{\#(h' \upharpoonright \{\!|rc|\!\})} : \{\!|\ rc\ |\!\}^* \bullet \#h^0 \leq maxLoss \wedge \\
\qquad\quad (\forall\, k : 1 \,..\, \#(h' \upharpoonright \{\!|\ rc\ |\!\}) \bullet \#h^k \leq maxLoss + 1) \wedge \\
\qquad\quad h^0 \frown \ldots \frown h^{\#(h' \upharpoonright \{\!|rc|\!\})} = h \upharpoonright \{\!|\ rc\ |\!\} \wedge \\
\qquad\quad (\forall\, i : 1 \,..\, (\#(h' \upharpoonright \{\!|\ rc\ |\!\}) - maxModified) \bullet \\
\qquad\qquad (\exists\, j : i \,..\, i + maxModified \bullet (h' \upharpoonright \{\!|\ rc\ |\!\})(j) = head(h^j))))) \\
\quad \wedge \\
\quad (\text{ran}\ val^*(h' \upharpoonright \{\!|\ tx\ |\!\}) \subseteq \text{ran}\ \mathsf{e} \Rightarrow \\
\qquad\quad (\forall\, x : \text{ran}\ val^*(h' \upharpoonright \{\!|\ rc\ |\!\}) \bullet x \notin \text{ran}\ \mathsf{e} \vee x \in \text{ran}\ val^*(h' \upharpoonright \{\!|\ tx\ |\!\}))
\end{array}
$$

If this ordering of threats is realistic, it is possible to extract valid data from the transmission medium, if it does not block completely: The predicate states that the length of $h' \upharpoonright \{\!|\ rc\ |\!\}$ grows with the number of inputs on tx:

Lemma 1 *Let h, h' be traces such that*

$$(\Delta_{M,2} \wr \Delta_{M,1})(h, h') \wedge 0 \leq \#(h \upharpoonright \{\!|\ tx\ |\!\}) - \#(h \upharpoonright \{\!|\ rc\ |\!\}) \leq 1$$

Then

$$\frac{\#(h' \upharpoonright \{\!|tx|\!\}) - maxLoss - 1}{maxLoss + 1} \leq \#(h' \upharpoonright \{\!|\ rc\ |\!\})$$

holds.

Proof.
$0 \leq \#(h \upharpoonright \{\!|\ tx\ |\!\}) - \#(h \upharpoonright \{\!|\ rc\ |\!\}) \leq 1$ ensures that after $maxLoss + 2$ inputs $\#(h \upharpoonright \{\!|\ rc\ |\!\}) > maxLoss$ is fulfilled. Therefore the consequence in the third conjunct of $(\Delta_{M,2} \wr \Delta_{M,1})(h, h')$ holds, so that we may assume the existence of h^i such that

$$h^0 \frown \ldots \frown h^{\#(h' \upharpoonright \{\!|rc|\!\})} = h \upharpoonright \{\!|\ rc\ |\!\}$$

Since $h' \upharpoonright \{\!|\ tx\ |\!\} = h \upharpoonright \{\!|\ tx\ |\!\}$ by the second conjunct of $(\Delta_{M,2} \wr \Delta_{M,1})(h, h')$,
$0 \leq \#(h \upharpoonright \{\!|\ tx\ |\!\}) - \#(h \upharpoonright \{\!|\ rc\ |\!\}) \leq 1$ implies $\#(h' \upharpoonright \{\!|\ tx\ |\!\}) - 1 \leq \#h \upharpoonright \{\!|\ rc\ |\!\})$, so

$$\#(h' \upharpoonright \{\!|\ tx\ |\!\}) - 1 \leq \#(h^0 \frown \ldots \frown h^{\#(h' \upharpoonright \{\!|rc|\!\})})$$

[5] If we weaken $\Delta_{M,1}$ such that only every nth message can be modified and $maxLoss < n - 1$, a fault-tolerant and secure network service would exist.

follows. Since $\#h^0 \leq maxLoss$ and $\#h^k \leq maxLoss + 1$ for $i > 0$, we get

$$\#(h' \upharpoonright \{\!|\ tx\ |\!\}) - 1 \leq maxLoss + \#(h' \upharpoonright \{\!|\ rc\ |\!\}) * (maxLoss + 1)$$

As a consequence

$$\frac{\#(h' \upharpoonright \{\!|tx|\!\}) - maxLoss - 1}{maxLoss+1} \leq \#(h' \upharpoonright \{\!|\ rc\ |\!\})$$

□

As soon as $\#(h' \upharpoonright \{\!|\ rc\ |\!\}) > maxModified$, it will be possible to find at least one valid data package out of $maxModified + 1$ consecutive values received on rc. For the remaining part of the case study it is therefore assumed that $\langle \Delta_{M,2}, \Delta_{M,1} \rangle$ is the appropriate compositional ordering of threats.

This example elucidates the fact that design aspects have to be taken into account when calculating the effect of possible faults and security threats. It is a fundamental difference whether malicious agents can only tamper with data successfully transmitted by the medium or whether they can modify data packages while it is still uncertain if a package will reach its destination.

Threat Analysis – Step 4: Derivation of Specifications in Presence of Threats Using the specification $S_M(h)$ of M given in the operational environment specification (p. 39) and assuming that the overall threat to M is represented by $(\Delta_{M,2} \wr \Delta_{M,1})$ as derived in the previous example, application of the threat introduction rule and of Lemma 1 results in

$$(M \wr (\Delta_{M,2} \wr \Delta_{M,1}))\ \textbf{sat}\ (S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h')$$

$$\begin{aligned}
&(S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h') \equiv \\
&\quad (\exists\, s \bullet val^*(s \upharpoonright \{\!|\ rc\ |\!\}) \leq^1 val^*(s \upharpoonright \{\!|\ tx\ |\!\}) \\
&\qquad\qquad \wedge\ ch^*(h' \upharpoonright \{\!|\ tx, rc\ |\!\}) \preceq ch^*(s \upharpoonright \{\!|\ tx, rc\ |\!\}) \wedge (h' \upharpoonright \{\!|\ tx\ |\!\}) = (s \upharpoonright \{\!|\ tx\ |\!\})) \\
&\quad \wedge \\
&\quad (\exists\, h'' \bullet (h'' \upharpoonright \{\!|\ tx, rc\ |\!\}) \preceq (h' \upharpoonright \{\!|\ tx, rc\ |\!\}) \wedge val^*(h'' \upharpoonright \{\!|\ rc\ |\!\}) \leq val^*(h'' \upharpoonright \{\!|\ tx\ |\!\}) \\
&\qquad\qquad \wedge\ \tfrac{\#(h' \upharpoonright \{\!|tx|\!\}) - maxLoss - 1}{(maxModified+1)*(maxLoss+1)} \leq \#(h'' \upharpoonright \{\!|\ rc\ |\!\})) \\
&\quad \wedge \\
&\quad (\text{ran}\ val^*(h' \upharpoonright \{\!|\ tx\ |\!\}) \subseteq \text{ran}\ \mathsf{e} \Rightarrow \\
&\qquad\qquad (\forall\, x : \text{ran}\ val^*(h' \upharpoonright \{\!|\ rc\ |\!\}) \bullet x \notin \text{ran}\ \mathsf{e} \vee x \in \text{ran}\ val^*(h' \upharpoonright \{\!|\ tx\ |\!\}))
\end{aligned}$$

In this specification the bound variable s represents the normal behaviour trace. The subtrace h'' of the acceptable behaviour trace h' contains the messages that are neither lost nor corrupted. This means that $(M \wr (\Delta_{M,2} \wr \Delta_{M,1}))$ will transmit at least a subtrace of inputs correctly. The behavioural specification in presence of threats is completely analogous for $MACK$.

5.1.4 Dependability Requirements

For SYS it is not allowed that irregularities become visible at the interface to $PROD$ and CON, therefore the dependability requirements say

$$\Delta_{SYS}(h, h') \equiv_{df} h' \upharpoonright \{\!|\ app_tx, app_rc\ |\!\} = h \upharpoonright \{\!|\ app_tx, app_rc\ |\!\}$$

Since the specification $S_{SYS}(h)$ of normal behaviour only refers to channels app_tx, app_rc, application of the threat introduction rule results in the same assertion:

$$(S_{SYS} \backslash \Delta_{SYS})(h') \equiv val^*(h' \upharpoonright \{\!| \ app_rc \ |\!\}) \leq^N val^*(h' \upharpoonright \{\!| \ app_tx \ |\!\})$$

5.2 System Architecture

5.2.1 System Structure

For the reliable and secure network service we choose a layered architecture

$$SYS = \mathcal{A}_{SYS}(ABP, SEC) = (ABP \parallel SEC)$$

as depicted in Figure 5.2.

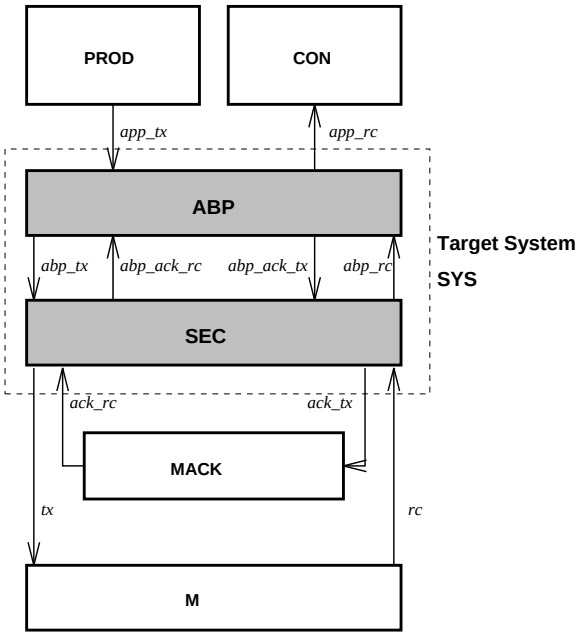

Figure 5.2: System architecture consisting of network layer and security layer.

This choice is inspired by the observation that the specification of $(M \backslash (\Delta_{M,2} \backslash \Delta_{M,1}))$ implies the existence of a subtrace $h''' \trianglelefteq h'$ satisfying $(h''' \upharpoonright \{\!| \ tx \ |\!\}) = (s \upharpoonright \{\!| \ tx \ |\!\}) \wedge h''' \trianglelefteq s$, where s is the trace denoting correct behaviour. This type of problem can be solved by means of the alternating bit protocol, which we intend to implement in the ABP layer. To make use of this protocol, we first have to extract h''' from h'. This will be the task of the SEC layer. The allocation of parts of the layers on different computers is deferred to a further decomposition step.

5.2.2 Interfaces

For the architecture of SYS introduced above we have alphabets

$$\alpha(ABP \setminus L_{ABP}) = \{\!| \ app_tx, app_rc, abp_tx, abp_rc, abp_ack_tx, abp_ack_rc \ |\!\}$$

$$\alpha(SEC \setminus L_{SEC}) = \{\!| \ tx, rc, ack_tx, ack_rc, abp_tx, abp_rc, abp_ack_tx, abp_ack_rc \ |\!\}$$

The interface definition is completed by describing the channel alphabets: Let $DATA$ denote the set of application data to be passed from $PROD$ to CON. Define $BIT = \{0, 1\}$.

Channel	Channel Alphabet
app_tx, app_rc	$DATA$
abp_tx, abp_rc	$BIT \times DATA$
abp_ack_tx, abp_ack_rc	BIT
tx, rc, ack_tx, ack_rc	BIT^*

The channel alphabets of tx and rc are defined to contain "raw data", i. e., each message on tx, rc is just an uninterpreted sequences of bits. Since the transmission media establish a point-to-point connection, we did not introduce a header data structure, which would precede the data block on a real network.

Obviously, the system interface I is contained in $ch(ABP) \cup ch(SEC)$.

5.2.3 Component Specifications

For the new component ABP we choose the following specification for the alternating bit protocol (see [99, pp. 44]):

$$ABP \textbf{ sat } S_{ABP}(h)$$

$$\begin{aligned}
&S_{ABP}(h) \equiv_{df} \\
&\quad data^*(\rho(h \restriction \{\!|\ abp_tx\ |\!\})) \leq^1 val^*(h \restriction \{\!|\ app_tx\ |\!\}) \\
&\quad \wedge \\
&\quad val^*(\rho(h \restriction \{\!|\ abp_ack_rc\ |\!\})) \leq^1 bit^*(\rho(h \restriction \{\!|\ abp_tx\ |\!\})) \\
&\quad \wedge \\
&\quad val^*(h \restriction \{\!|\ app_rc\ |\!\}) \leq^1 data^*(\rho(h \restriction \{\!|\ abp_rc\ |\!\})) \\
&\quad \wedge \\
&\quad val^*(\rho(h \restriction \{\!|\ abp_ack_tx\ |\!\})) \leq^1 bit^*(\rho(h \restriction \{\!|\ abp_rc\ |\!\}))
\end{aligned}$$

In this definition, bit and $data$ are the projections onto the bit and data component, respectively, of channel events of abp_tx and abp_rc. For channels c with alphabet BIT we may also write $bit(c.b)$ instead of $val(c.b)$ to extract the bit value from a channel event. ρ is a filter for traces deleting duplicated messages identified by means of the bit value on channels with alphabet BIT or $BIT \times DATA$:

$$\rho(\langle\ \rangle) = \langle\ \rangle$$

$$\rho(s^\frown\langle e\rangle) = \textbf{if } \#s > 0 \wedge bit(e) = bit(last(s)) \textbf{ then } \rho(s) \textbf{ else } \rho(s)^\frown\langle e\rangle$$

For SEC we specify

$$SEC \textbf{ sat } S_{SEC}(h)$$

$$\begin{aligned}
&S_{SEC}(h) \equiv_{df} \\
&\quad val^*(h \restriction \{\!|\ tx\ |\!\}) \leq^1 \epsilon(val^*(h \restriction \{\!|\ abp_tx\ |\!\})) \\
&\quad \wedge \\
&\quad val^*(h \restriction \{\!|\ ack_tx\ |\!\}) \leq^1 \epsilon(val^*(h \restriction \{\!|\ abp_ack_tx\ |\!\})) \\
&\quad \wedge \\
&\quad val^*(h \restriction \{\!|\ abp_rc\ |\!\}) \leq^1 data^*(\varphi(val^*(h \restriction \{\!|\ rc\ |\!\}))) \\
&\quad \wedge \\
&\quad val^*(h \restriction \{\!|\ abp_ack_rc\ |\!\}) \leq^1 data^*(\varphi(val^*(h \restriction \{\!|\ ack_rc\ |\!\})))
\end{aligned}$$

In this specification, ϵ denotes an encryption filter adding a sequence number to each value of a trace and encrypting this pair by means of e:

$$\epsilon(\langle\,\rangle) = \langle\,\rangle$$

$$\epsilon(s^\frown\langle x\rangle) = \epsilon(s)^\frown\langle \mathsf{e}(x, \#s + 1)\rangle$$

The filter φ will discard every message which is not correctly encrypted ($x \notin \operatorname{ran}\mathsf{e}$) or which is a replay of an older message, as can be detected by means of the sequence number $snum(\mathsf{e}^{-1}(x))$ of the decoded message:

$$\varphi(\langle\,\rangle) = \langle\,\rangle$$

$$\begin{aligned}
\varphi(s^\frown\langle x\rangle) = \ &\textbf{if } x \notin \operatorname{ran}\mathsf{e} \vee (\#\varphi(s) > 0 \wedge snum(\mathsf{e}^{-1}(x)) \leq snum(last(\varphi(s)))) \\
&\textbf{then } \varphi(s) \\
&\textbf{else } \varphi(s)^\frown\langle \mathsf{e}^{-1}(x)\rangle
\end{aligned}$$

If the message is accepted by φ the filter outputs its decoded value $\mathsf{e}^{-1}(x)$, consisting of the pair $(data, sequence_number)$.

5.2.4 Threat Analysis (Internal)

Let us assume that for the architecture $\mathcal{A}_{SYS}(ABP, SEC)$ the possibility of data losses on channel abp_rc has to be taken into account (Figure 5.3). As before it is assumed that only $maxLoss \geq 0$ messages may be lost in a row. Formally speaking, there exists a threat

$$\begin{aligned}
&\Delta_{SEC,1}(h, h') \equiv_{df} \\
&\quad (h' \restriction \alpha(SEC)) \preceq (h \restriction \alpha(SEC)) \\
&\quad \wedge \\
&\quad (h' \restriction (\alpha(SEC) - \{\!|\ abp_rc\ |\!\})) = (h \restriction (\alpha(SEC) - \{\!|\ abp_rc\ |\!\})) \\
&\quad \wedge \\
&\quad (\#(h \restriction \{\!|\ abp_rc\ |\!\}) > maxLoss \Rightarrow \\
&\qquad \#(h' \restriction \{\!|\ abp_rc\ |\!\}) > 0 \\
&\qquad \wedge \\
&\qquad (\exists\, h^0, h^1, \ldots, h^{\#(h' \restriction \{|abp_rc|\})} : \{\!|\ rc\ |\!\}^* \bullet \\
&\qquad\quad \#h^0 \leq maxLoss \wedge h^0{}^\frown \ldots {}^\frown h^{\#(h' \restriction \{|abp_rc|\})} = h \restriction \{\!|\ abp_rc\ |\!\} \\
&\qquad\quad \wedge \\
&\qquad\quad (\forall\, i : 1 \,..\, \#(h' \restriction \{\!|\ abp_rc\ |\!\}) \bullet \\
&\qquad\qquad (h' \restriction \{\!|\ abp_rc\ |\!\})(i) = head(h^i) \wedge \#h^i \leq maxLoss + 1)))
\end{aligned}$$

Deriving the specification for SEC in presence of this threat uses a lemma analogous to Lemma 1 and results in

$$(SEC \wr \Delta_{SEC,1}) \textbf{ sat } (S_{SEC} \wr \Delta_{SEC,1})(h')$$

$$\begin{aligned}
&(S_{SEC} \wr \Delta_{SEC,1})(h') \equiv_{df} \\
&\quad val^*(h' \restriction \{\!|\ tx\ |\!\}) \leq^1 \epsilon(val^*(h' \restriction \{\!|\ abp_tx\ |\!\})) \\
&\quad \wedge \\
&\quad val^*(h' \restriction \{\!|\ ack_tx\ |\!\}) \leq^1 \epsilon(val^*(h' \restriction \{\!|\ abp_ack_tx\ |\!\})) \\
&\quad \wedge \\
&\quad val^*(h' \restriction \{\!|\ abp_ack_rc\ |\!\}) \leq^1 data^*(\varphi(val^*(h' \restriction \{\!|\ ack_rc\ |\!\}))) \\
&\quad \wedge \\
&\quad (\exists\, s \bullet val^*(s \restriction \{\!|\ abp_rc\ |\!\}) \leq^1 data^*(\varphi(val^*(s \restriction \{\!|\ rc\ |\!\}))) \wedge \\
&\qquad\qquad h' \restriction \{\!|\ rc, abp_rc\ |\!\} \preceq s \restriction \{\!|\ rc, abp_rc\ |\!\}) \wedge h' \restriction \{\!|\ rc\ |\!\} = s \restriction \{\!|\ rc\ |\!\}) \\
&\quad \wedge \\
&\quad \frac{\#(\varphi(val^*(h' \restriction \{\!|rc|\!\}))) - maxLoss - 1}{maxLoss+1} \leq \#(h' \restriction \{\!|\ abp_rc\ |\!\})
\end{aligned}$$

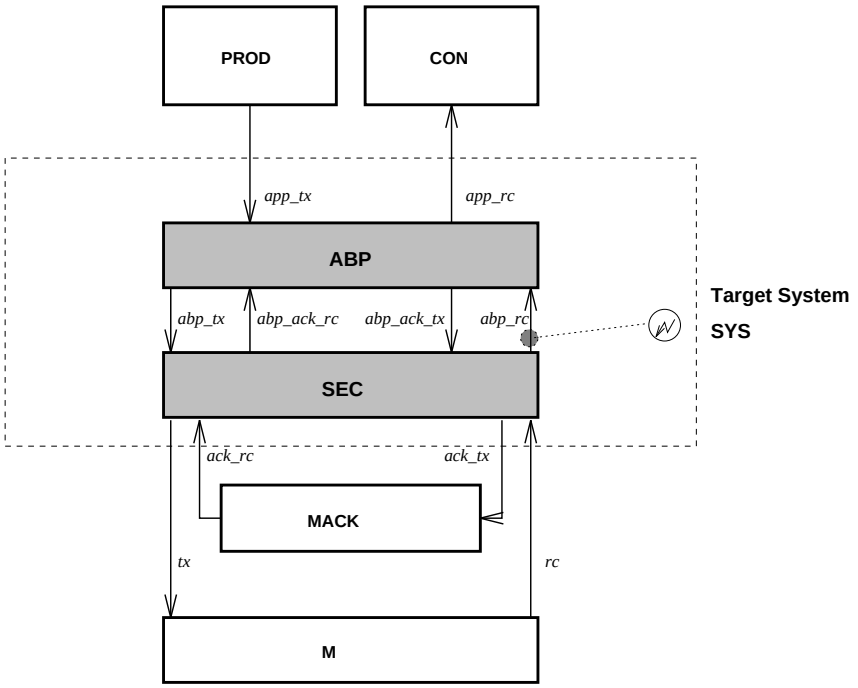

Figure 5.3: Internal threat analysis: Possible loss of data on channel abp_rc.

We will see below that $\mathcal{A}_{SYS}(ABP, SEC)$ correctly implements the system requirements, even in the presence of threat $\Delta_{SEC,1}(h, h')$. However, if also a security threat allowing to modify data on channel abp_tx had to be taken into account, it is easy to see that the strength of the mechanisms pre-planned for ABP and SEC would not suffice to provide a dependable network service. An idea to re-design the architecture could be to make the channel abp_tx "invulnerable" to agents located inside SYS. For example, the transmission component in SEC could be linked into the transmission component of ABP, so that the communication on abp_tx could be implemented by means of a procedure call using only local variables not accessible to other processes than ABP. Formally this architecture decision could be

modelled as

$$\mathcal{A}'_{SYS}(ABP, SEC) = (ABP \parallel SEC) \setminus \{\!|\ abp_tx\ |\!\}$$

5.3 Verification on System Level

Applying the System Verification Obligations introduced in Section 4.2.4 to the example and observing that the specifications of *PROD* and *CON* are just *true*, it has to be shown that the following properties can be derived:

1. System Verification Obligation – Normal Behaviour:

$$\frac{\begin{array}{l} ABP \textbf{ sat } S_{ABP}(h) \\ SEC \textbf{ sat } S_{SEC}(h) \\ M \textbf{ sat } S_M(h) \\ MACK \textbf{ sat } S_{MACK}(h) \end{array}}{(M \,|||\, MACK) \parallel ABP \parallel SEC \textbf{ sat } S(h)}$$

2. System Verification Obligation – Acceptable Behaviour:

$$\frac{\begin{array}{l} ABP \textbf{ sat } S_{ABP}(h) \\ (SEC \wr \Delta_{SEC,1}) \textbf{ sat } (S_{SEC} \wr \Delta_{SEC,1})(h) \\ (M \wr (\Delta_{M,2} \wr \Delta_{M,1})) \textbf{ sat } (S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h) \\ (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1})) \textbf{ sat } (S_{MACK} \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))(h) \end{array}}{((M \wr (\Delta_{M,2} \wr \Delta_{M,1})) \,|||\, (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))) \parallel ABP \parallel (SEC \wr \Delta_{SEC,1}) \textbf{ sat } (S \wr \Delta_{SYS})(h)}$$

We skip the proof of the normal behaviour obligation, because the verification concept will become clear in the treatment of the obligation for acceptable behaviour. For the second verification obligation we have assumed that the alternating bit protocol layer is not subject to internal threats. Therefore $S_{ABP}(h)$ and ABP appear without application of the threat introduction operator. We will split this proof obligation into two steps: First it will be derived that the security layer in cooperation with the transmission media will only pass messages to the network layer which have not been corrupted by security attacks, i. e.,

Proof Obligation 1:

$$\frac{\begin{array}{l} (SEC \wr \Delta_{SEC,1}) \textbf{ sat } (S_{SEC} \wr \Delta_{SEC,1})(h) \\ (M \wr (\Delta_{M,2} \wr \Delta_{M,1})) \textbf{ sat } (S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h) \\ (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1})) \textbf{ sat } (S_{MACK} \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))(h) \end{array}}{((M \wr (\Delta_{M,2} \wr \Delta_{M,1})) \,|||\, (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))) \parallel (SEC \wr \Delta_{SEC,1}) \textbf{ sat } S_1(h) \wedge S_2(h)}$$

where $S_1(h)$ defines the behaviour of the transmission via M and $S_2(h)$ the behaviour of the transmission via $MACK$:

$$\begin{array}{l} S_1(h) \equiv_{df} \\ \quad val^*(h \upharpoonright \{\!|\ abp_rc\ |\!\}) \preceq val^*(h \upharpoonright \{\!|\ abp_tx\ |\!\}) \\ \quad \wedge \\ \quad \frac{\#(h \upharpoonright \{\!|\ abp_tx\ |\!\}) - (maxModified+1)*(maxLoss+1)^2 - maxLoss - 2}{(maxModified+1)*(maxLoss+1)^2} \leq \#(h \upharpoonright \{\!|\ abp_rc\ |\!\}) \end{array}$$

$$S_2(h) \equiv_{df}$$
$$val^*(h \upharpoonright \{\!|\ abp_ack_rc\ |\!\}) \ \trianglelefteq\ val^*(h \upharpoonright \{\!|\ abp_ack_tx\ |\!\})$$
$$\wedge$$
$$\frac{\#(h\upharpoonright\{\!|abp_ack_tx|\!\})-(maxModified+1)*(maxLoss+1)^2-maxLoss-2}{(maxModified+1)*(maxLoss+1)^2} \leq \#(h \upharpoonright \{\!|\ abp_ack_rc\ |\!\})$$

For the second step of the proof we observe that the alternating bit protocol with the behavioural specification as given for ABP above will provide a reliable network service when working in parallel with transmission components satisfying $S_1(h)$ and $S_2(h)$ respectively. We will only give the proof for Proof Obligation 1; a proof for the second step may be found for example in [99, pp. 44].

Lemma 2 *Proof Obligation 1 is implied by*

Proof Obligation 2: *If*

$$(S_{SEC} \wr \Delta_{SEC,1})(h) \wedge (S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h)$$

then $S_1(h)$ *holds.*
and
Proof Obligation 3: *If*

$$(S_{SEC} \wr \Delta_{SEC,1})(h) \wedge (S_{MACK} \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))(h)$$

then $S_2(h)$ *holds.*

Proof.
The premises of Proof Obligation 1 state that
$(SEC \wr \Delta_{SEC,1})$ **sat** $(S_{SEC} \wr \Delta_{SEC,1})(h)$, $(M \wr (\Delta_{M,2} \wr \Delta_{M,1}))$ **sat** $(S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h)$
and $(MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))$ **sat** $(S_{MACK} \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))(h)$.

We use the law

$$\frac{P \textbf{ sat } S_P(h) \quad Q \textbf{ sat } S_Q(h)}{(P \parallel Q) \textbf{ sat } S_P(h \upharpoonright \alpha(P)) \wedge S_Q(h \upharpoonright \alpha(Q))}$$

which holds in the trace model of CSP [41, p. 90]. In our context, the free variable h of a specification P **sat** $S(h)$ appears only in expressions restricted to channels of P, therefore we always have

$$S_P(h \upharpoonright \alpha(P)) \wedge S_Q(h \upharpoonright \alpha(Q)) \Leftrightarrow S_P(h) \wedge S_Q(h)$$

for a trace variable h of $(P \parallel Q)$. Moreover $|||$ is identical to $\parallel$ for disjoint alphabets. It follows that

$$((M \wr (\Delta_{M,2} \wr \Delta_{M,1})) \,|||\, (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))) \parallel (SEC \wr \Delta_{SEC,1}) \textbf{ sat}$$
$$(S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h) \wedge (S_{MACK} \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))(h) \wedge (S_{SEC} \wr \Delta_{SEC,1})(h)$$

If Proof Obligation 2 is valid, application of the consequence rule results in

$$((M \wr (\Delta_{M,2} \wr \Delta_{M,1})) \,|||\, (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))) \parallel (SEC \wr \Delta_{SEC,1}) \textbf{ sat } S_1(h)$$

Similarly, if Proof Obligation 3 is valid, application of the consequence rule results in

$$((M \wr (\Delta_{M,2} \wr \Delta_{M,1})) \;|||\; (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))) \parallel (SEC \wr \Delta_{SEC,1}) \textbf{ sat } S_2(h)$$

Therefore, if both obligations are valid, we may apply the conjunction rule which yields Proof Obligation 1.
□

The following lemmas are used to prepare the proof of Obligation 2.

Lemma 3 *If*

$$(S_{SEC} \wr \Delta_{SEC,1})(h) \wedge (S_M \wr (\Delta_{M,2} \wr \Delta_{M,1})(h)$$

then

$$(\forall\, x : \text{ran } val^*(h \upharpoonright \{\!|\, rc \,|\!\}) \bullet x \notin \text{ran } \mathsf{e} \vee x \in \text{ran } \epsilon(val^*(h \upharpoonright \{\!|\, abp_tx \,|\!\})))$$

holds

Proof.
According to $(S_{SEC} \wr \Delta_{SEC,1})(h)$ (p. 48) and the definition of ϵ (p. 47), every value passed over channel tx is in the range of e. The lemma now follows from the last conjunct in predicate $(S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h)$ (p. 44).
□

The following lemma summarizes the implications of specification $(S_{SEC} \wr \Delta_{SEC,1})(h)$ (p. 48) with respect to events observed on channels abp_rc and rc: An event $abp_rc.y$ is always preceded by a communication $rc.x$, such that

- x is correctly encrypted by e.
- y is equal to the data part of the decoded message x.
- The sequence number associated with message x is higher than the sequence number of the most recent previous message on rc which was accepted by the filter φ (p. 47).

Lemma 4 *The validity of* $(S_{SEC} \wr \Delta_{SEC,1})(h)$ *implies*

$$\begin{array}{l}(\forall\, y \bullet last(h \upharpoonright \{\!|\, rc, abp_rc \,|\!\}) = abp_rc.y \Rightarrow \\ \quad (\exists\, x \bullet last(front(h \upharpoonright \{\!|\, rc, abp_rc \,|\!\}))) = rc.x \wedge \\ \qquad x \in \text{ran } \mathsf{e} \wedge \\ \qquad y = data(\mathsf{e}^{-1}(x)) \wedge \\ \qquad (\#(h \upharpoonright \{\!|\, rc \,|\!\}) > 1 \Rightarrow \\ \qquad\quad snum(\mathsf{e}^{-1}(x)) > snum(last(\varphi(val^*(front(h \upharpoonright \{\!|\, rc \,|\!\}))))))))\end{array}$$

Proof.
According to the fourth conjunct in $(S_{SEC} \wr \Delta_{SEC,1})(h)$ (p. 48), we may assume the existence of a trace s satisfying

$$\begin{array}{l}val^*(s \upharpoonright \{\!|\, abp_rc \,|\!\}) \leq^1 data^*(\varphi(val^*(s \upharpoonright \{\!|\, rc \,|\!\}))) \wedge \\ \qquad\qquad h \upharpoonright \{\!|\, rc, abp_rc \,|\!\} \preceq s \upharpoonright \{\!|\, rc, abp_rc \,|\!\}) \wedge h \upharpoonright \{\!|\, rc \,|\!\} = s \upharpoonright \{\!|\, rc \,|\!\}\end{array}$$

Assume that the lemma has been proven for those traces h satisfying $h = s$, i. e., for those h where data losses on abp_rc did not occur. Then the lemma will also hold for arbitrary h satisfying $(S_{SEC} | \Delta_{SEC,1})(h)$, because $h \restriction \{\!|\ rc, abp_rc\ |\!\} \trianglelefteq s \restriction \{\!|\ rc, abp_rc\ |\!\}) \wedge h \restriction \{\!|\ rc\ |\!\} = s \restriction \{\!|\ rc\ |\!\}$, and the implication of the lemma remains valid for any trace derived from s by deleting abp_rc-events only.

For rest of the proof we may therefore assume

$$val^*(h \restriction \{\!|\ abp_rc\ |\!\}) \leq^1 data^*(\varphi(val^*(h \restriction \{\!|\ rc\ |\!\}))) \tag{0}$$

Let $(h \restriction \{\!|\ rc, abp_rc\ |\!\}) = (u \restriction \{\!|\ rc, abp_rc\ |\!\}) \frown \langle abp_rc.y \rangle$. Then $(u \restriction \{\!|\ rc, abp_rc\ |\!\})$ cannot be empty because of (0).

We will first prove that $ch(last(u \restriction \{\!|\ rc, abp_rc\ |\!\})) = rc$ by assuming the contrary and deriving a contradiction. Assume therefore that $ch(last(u \restriction \{\!|\ rc, abp_rc\ |\!\})) = abp_rc$. Then

$$\#(h \restriction \{\!|\ abp_rc\ |\!\}) = \#(front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ abp_rc\ |\!\}) + 2 \tag{1}$$

and

$$h \restriction \{\!|\ rc\ |\!\} = front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ rc\ |\!\} \tag{2}$$

Now (0) together with (2) implies

$$val^*(h \restriction \{\!|\ abp_rc\ |\!\}) \leq^1 data^*(\varphi(val^*(front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ rc\ |\!\}))))$$

and therefore

$$\#(h \restriction \{\!|\ abp_rc\ |\!\}) \leq \#(\varphi(val^*(front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ rc\ |\!\}))) \tag{3}$$

Since $u \leq h$ and (0) is also valid for every prefix of h, this yields

$$val^*(front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ abp_rc\ |\!\}))) \leq^1 \\ data^*(\varphi(val^*(front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ rc\ |\!\}))))$$

and therefore

$$\#(\varphi(val^*(front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ rc\ |\!\})))) \leq \\ \#(front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ abp_rc\ |\!\})) + 1$$

With (3), this results in

$$\#(h \restriction \{\!|\ abp_rc\ |\!\}) \leq \#(front(u \restriction \{\!|\ rc, abp_rc\ |\!\}) \restriction \{\!|\ abp_rc\ |\!\})) + 1$$

which contradicts (1).

Let therefore $last(u \restriction \{\!|\ rc, abp_rc\ |\!\}) = rc.x$.

Case (C1): Assume $(h \restriction \{\!|\ rc\ |\!\}) = \langle rc.x \rangle$. If $(h \restriction \{\!|\ rc, abp_rc\ |\!\}) = \langle rc.x, abp_rc.y \rangle$, (0) implies $val^*(h \restriction \{\!|\ abp_rc\ |\!\}) = data^*(\varphi(val^*(h \restriction \{\!|\ rc\ |\!\})))$, that is, $\langle y \rangle = data^*(\varphi(\langle x \rangle))$. Now the definition of φ applied to the case $\varphi(\langle\ \rangle \frown \langle x \rangle)$ yields $x \in \mathrm{ran}\, \mathsf{e} \wedge \varphi(\langle x \rangle) = \langle \mathsf{e}^{-1}(x) \rangle$ which proves the lemma for $\#(h \restriction \{\!|\ rc\ |\!\}) = 1$.

Case (C2): Assume $\#(h \restriction \{\!|\ rc\ |\!\}) > 1 \wedge last(h \restriction \{\!|\ rc, abp_rc\ |\!\}) = abp_rc.y$. Again, (0) implies $val^*(h \restriction \{\!|\ abp_rc\ |\!\}) = data^*(\varphi(val^*(h \restriction \{\!|\ rc\ |\!\})))$, which now results in $y =$

$data(last(\varphi(val^*(h \upharpoonright \{\!|\ rc\ |\!\})))))$. From the recursive definition of φ (p. 47) we get (using induction over the length of sequences z)

$$\#\varphi(z) > 0 \Rightarrow last(\varphi(z)) = \mathsf{e}^{-1}(last(z))$$

It follows that

$$last(\varphi(val^*(h \upharpoonright \{\!|\ rc\ |\!\}))) = \mathsf{e}^{-1}(val(last(h \upharpoonright \{\!|\ rc\ |\!\}))) = \mathsf{e}^{-1}(x)$$

and

$$\varphi(val^*(h \upharpoonright \{\!|\ rc\ |\!\})) = \varphi(val^*(front(h \upharpoonright \{\!|\ rc\ |\!\})))^\frown\langle \mathsf{e}^{-1}(x)\rangle$$

Now the definition of φ implies

$$snum(\mathsf{e}^{-1}(x)) > snum(last(\varphi(val^*(front(h \upharpoonright \{\!|\ rc\ |\!\})))))$$

which completes the proof.
□

Lemma 5 *If*

$$(S_{SEC} \wr \Delta_{SEC,1})(h) \wedge (S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h)$$

holds, this implies

$$\frac{\#(h \upharpoonright \{\!|abp_tx|\!\}) - maxLoss - 2}{(maxModified+1)*(maxLoss+1)} \leq \#\varphi(val^*(h \upharpoonright \{\!|\ rc\ |\!\}))$$

Proof.
The first conjunct in $(S_{SEC} \wr \Delta_{SEC,1})(h)$ is $val^*(h \upharpoonright \{\!|\ tx\ |\!\}) \leq^1 \epsilon(val^*(h \upharpoonright \{\!|\ abp_tx\ |\!\}))$. Since $\#\epsilon(s) = \#s$ for every trace s, this yields

$$\#(h \upharpoonright \{\!|\ abp_tx\ |\!\})) - 1 \leq \#(h \upharpoonright \{\!|\ tx\ |\!\})$$

Together with the second conjunct of $(S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h)$ this implies

$$(\exists\, h'' \bullet (h'' \upharpoonright \{\!|\ tx, rc\ |\!\}) \trianglelefteq (h \upharpoonright \{\!|\ tx, rc\ |\!\}) \wedge val^*(h'' \upharpoonright \{\!|\ rc\ |\!\}) \leq val^*(h'' \upharpoonright \{\!|\ tx\ |\!\}) \wedge \frac{\#(h \upharpoonright \{\!|abp_tx|\!\}) - maxLoss - 2}{(maxModified+1)*(maxLoss+1)} \leq \#(h'' \upharpoonright \{\!|\ rc\ |\!\}))$$

The lemma will now follow if $\#(h'' \upharpoonright \{\!|\ rc\ |\!\})) \leq \#\varphi(val^*(h \upharpoonright \{\!|\ rc\ |\!\}))$ can be established. To this end, we will show that every $(h'' \upharpoonright \{\!|\ rc\ |\!\})$-event in h adds an entry to $\varphi(val^*(h \upharpoonright \{\!|\ rc\ |\!\}))$. Assume therefore that $last(h \upharpoonright \{\!|\ tx, rc\ |\!\}) = last(h'' \upharpoonright \{\!|\ rc\ |\!\})$. We show that this implies

$$\varphi(val^*(h \upharpoonright \{\!|\ rc\ |\!\})) = \varphi(val^*(front(h \upharpoonright \{\!|\ rc\ |\!\})))^\frown\langle \mathsf{e}^{-1}(val(last(h'' \upharpoonright \{\!|\ rc\ |\!\})))\rangle$$

Since $val^*(h'' \upharpoonright \{\!|\ rc\ |\!\})$ is a subtrace of tx-values, the first conjunct in $(S_{SEC} \wr \Delta_{SEC,1})(h)$ implies that $\operatorname{ran} val^*(h'' \upharpoonright \{\!|\ rc\ |\!\}) \subseteq \operatorname{ran} \mathsf{e}$. If $\#(h \upharpoonright \{\!|\ rc\ |\!\}) = 1$, this means that $val(h \upharpoonright \{\!|\ rc\ |\!\})$ will be accepted by the filter φ. If $\#(h \upharpoonright \{\!|\ rc\ |\!\}) > 1$, the first conjunct of $(S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h)$ implies that $val(last(h'' \upharpoonright \{\!|\ rc\ |\!\}))$ is the value of the most recent tx-event transmitted before, because $M \wr (\Delta_{M,2} \wr \Delta_{M,1})$ acts as an (unreliable) one-place buffer. Since ϵ adds strictly increasing sequence numbers, the third conjunct in $(S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h)$ implies that $snum(\mathsf{e}^{-1}(val(last(h'' \upharpoonright \{\!|\ rc\ |\!\}))))$ is greater than any other sequence number of $(h \upharpoonright \{\!|\ rc\ |\!\})$-events. Now the definition of φ implies that $\mathsf{e}^{-1}(val(last(h'' \upharpoonright \{\!|\ rc\ |\!\})))$ is appended to the sequence filtered by φ. This completes the proof.
□

Proof of Obligation 2 : If $(S_{SEC} \wr \Delta_{SEC,1})(h) \wedge (S_M \wr (\Delta_{M,2} \wr \Delta_{M,1}))(h)$ then $S_1(h)$ follows.

To prove $val^*(h \restriction \{\!|\ abp_rc\ |\!\}) \trianglelefteq val^*(h \restriction \{\!|\ abp_tx\ |\!\})$, we will use induction over the length of $(h \restriction \{\!|\ rc, abp_rc\ |\!\})$. Clearly, the assertion holds for $(h \restriction \{\!|\ rc, abp_rc\ |\!\}) = \langle\ \rangle$. Now assume that it holds for all traces of length less or equal $n \geq 0$ and let $(h \restriction \{\!|\ rc, abp_rc\ |\!\}) = (u \restriction \{\!|\ rc, abp_rc\ |\!\})^\frown \langle a \rangle$ with $\#(u \restriction \{\!|\ rc, abp_rc\ |\!\}) = n$ and some rc- or abp_rc-event a.

Case (C1): Assume that $(u \restriction \{\!|\ rc, abp_rc\ |\!\}) = \langle\ \rangle$ or $ch(last(u \restriction \{\!|\ rc, abp_rc\ |\!\})) = abp_rc$. Then Lemma 4 implies that $(u \restriction \{\!|\ rc, abp_rc\ |\!\})$ can only be continued by an rc-event, i. e., we may assume $(h \restriction \{\!|\ rc, abp_rc\ |\!\}) = (u \restriction \{\!|\ rc, abp_rc\ |\!\})^\frown \langle rc.x \rangle$ for some $x \in \alpha(rc)$. As a consequence

$$\begin{aligned} val^*(h \restriction \{\!|\ abp_rc\ |\!\}) &= val^*(u \restriction \{\!|\ abp_rc\ |\!\}) \\ &\trianglelefteq val^*(u \restriction \{\!|\ abp_tx\ |\!\}) && \text{[induction hypothesis]} \\ &\leq val^*(h \restriction \{\!|\ abp_tx\ |\!\}) && [u \leq h] \end{aligned}$$

Case (C2): Assume that $last(u \restriction \{\!|\ rc, abp_rc\ |\!\}) = rc.x$.

Case (C21): Assume that
$x \notin \operatorname{ran} \mathsf{e} \vee snum(\mathsf{e}^{-1}(x)) \leq snum(last(\varphi(val^*(front(u \restriction \{\!|\ rc\ |\!\})))))$
Then according to Lemma 4 a must be an rc-event and we have again $h \restriction \{\!|\ abp_rc\ |\!\} = u \restriction \{\!|\ abp_rc\ |\!\}$ as in Case (C1).

Case (C22): Assume that
$x \in \operatorname{ran} \mathsf{e} \wedge snum(\mathsf{e}^{-1}(x)) > snum(last(\varphi(val^*(front(u \restriction \{\!|\ rc\ |\!\})))))$
Then a may be an rc-event ($rc.x$ is lost before an abp_rc-output occurs) or an abp_rc event. In the former case we use the same argument as in (C1). In the latter case we get

$$\begin{aligned} val^*(h \restriction \{\!|\ abp_rc\ |\!\}) &= val^*((u \restriction \{\!|\ abp_rc\ |\!\})^\frown \langle data(\mathsf{e}^{-1}(x)) \rangle) && \text{[Lemma 4]} \\ &\trianglelefteq val^*((u \restriction \{\!|\ abp_tx\ |\!\})^\frown \langle data(\mathsf{e}^{-1}(x)) \rangle) && \text{[induction hypothesis]} \\ &\trianglelefteq val^*(h \restriction \{\!|\ abp_tx\ |\!\}) \\ & && \text{[Lemma 3, def. of } \epsilon \text{ and } snum(\mathsf{e}^{-1}(x)) > snum(last(\varphi(val^*(front(u \restriction \{\!|\ rc\ |\!\})))))] \end{aligned}$$

This completes the inductive proof of $val^*(h \restriction \{\!|\ abp_rc\ |\!\}) \trianglelefteq val^*(h \restriction \{\!|\ abp_tx\ |\!\})$.

For the estimate of $\#(h \restriction \{\!|\ abp_rc\ |\!\})$ we calculate

$$\begin{aligned} \#(h \restriction \{\!|\ abp_rc\ |\!\}) &\geq \tfrac{\#(\varphi(val^*(h \restriction \{\!|rc|\!\}))) - maxLoss - 1}{maxLoss+1} && [(S_{SEC} \wr \Delta_{SEC,1})(h)] \\ &\geq \tfrac{\frac{\#(h \restriction \{\!|abp_tx|\!\}) - maxLoss - 2}{(maxModified+1)*(maxLoss+1)} - maxLoss - 1}{maxLoss+1} && \text{[Lemma 5]} \\ &= \tfrac{\#(h \restriction \{\!|abp_tx|\!\}) - (maxModified+1)*(maxLoss+1)^2 - maxLoss - 2}{(maxModified+1)*(maxLoss+1)^2} \end{aligned}$$

This completes the proof of Obligation 2; Obligation 3 is derived in an analogous way.
□

5.4 Recursive Application of the Development Procedure

For the architecture $\mathcal{A}_{SYS}(ABP, SEC)$ depicted in Figure 5.2 further decomposition steps may be taken according to Figure 5.4.

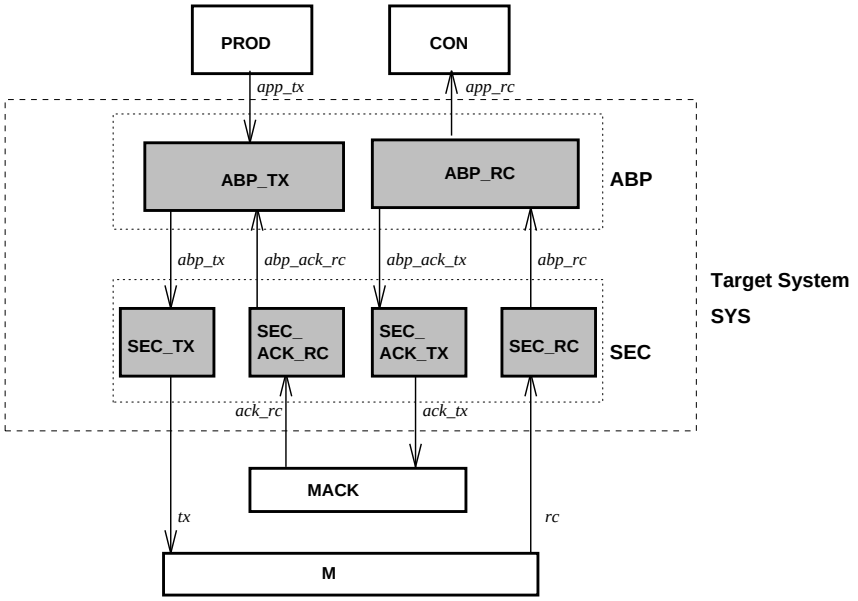

Figure 5.4: Correct combination of a new security layer with re-used network layer.

For ABP the architecture $\mathcal{A}_{ABP}(ABP_TX, ABP_RC) = (ABP_TX \,|||\, ABP_RC)$ is chosen. ABP_TX and ABP_RC have alphabets $\alpha(ABP_TX) = \{\!|\ app_tx, abp_tx, abp_ack_rc\ |\!\}$ and $\alpha(ABP_RC) = \{\!|\ app_rc, abp_rc, abp_ack_tx\ |\!\}$ and are specified by

$$ABP_TX \textbf{ sat } S_{ABP_TX}(h)$$

$$\begin{aligned} &S_{ABP_TX}(h) \equiv_{df} \\ &\quad data^*(\rho(h \restriction \{\!|\ abp_tx\ |\!\})) \leq^1 val^*(h \restriction \{\!|\ app_tx\ |\!\}) \\ &\quad \wedge \\ &\quad val^*(\rho(h \restriction \{\!|\ abp_ack_rc\ |\!\})) \leq^1 bit^*(\rho(h \restriction \{\!|\ abp_tx\ |\!\})) \end{aligned}$$

and

$$ABP_RC \textbf{ sat } S_{ABP_RC}(h)$$

$$\begin{aligned} &S_{ABP_RC}(h) \equiv_{df} \\ &\quad val^*(h \restriction \{\!|\ app_rc\ |\!\}) \leq^1 data^*(\rho(h \restriction \{\!|\ abp_rc\ |\!\})) \\ &\quad \wedge \\ &\quad val^*(\rho(h \restriction \{\!|\ abp_ack_tx\ |\!\})) \leq^1 bit^*(\rho(h \restriction \{\!|\ abp_rc\ |\!\})) \end{aligned}$$

Since ABP_TX and ABP_RC have disjoint alphabets and are composed by means of interleaving, it is trivial that their specifications imply the component requirements specification $S_{ABP}(h)$ since the latter is just the conjunction of $S_{ABP_TX}(h)$ and $S_{ABP_RC}(h)$.

Analogously, SEC is decomposed by

$$\begin{aligned} &\mathcal{A}_{SEC}(SEC_TX, SEC_RC, SEC_ACK_TX, SEC_ACK_RC) = \\ &\qquad (SEC_TX \,|||\, SEC_RC \,|||\, SEC_ACK_TX \,|||\, SEC_ACK_RC) \end{aligned}$$

with the corresponding terms from $S_{SEC}(h)$ as component specifications.

In the next decomposition for each of the components $ABP_TX, \ldots, SEC_ACK_RC$ it will be appropriate to introduce *explicit* specifications, because we will reach the level of isolated sequential processes, which are straight forward to implement in the target programming language. The following processes may be used to implement the implicit specifications of the security layer:

$$SEC_TX = STX(1)$$

$$STX(n) = abp_tx?x \rightarrow tx!\mathsf{e}(x, n) \rightarrow STX(n+1)$$

$$SEC_RC = SRC(0)$$

$$\begin{aligned} SRC(n) = rc?z \rightarrow \\ &(\textbf{if } z \notin \operatorname{ran} \mathsf{e} \vee snum(\mathsf{e}^{-1}(z)) \leq n \\ &\textbf{then } SRC(n) \\ &\textbf{else } (abp_rc!(bit(\mathsf{e}^{-1}(z)), data(\mathsf{e}^{-1}(z))) \rightarrow SRC(n+1))) \end{aligned}$$

SEC_ACK_TX, SEC_ACK_RC are defined analogously. For the network layer we may re-use the processes ABP_TX and ABP_RC defined in Chapter 2 (p. 16).

To prove that these processes correctly implement their corresponding behavioural specifications two techniques may be used:

- Application of CSP laws connecting explicit process algebraic representations to their corresponding implicit specifications, as given by Hoare [41].
- Assertional reasoning about CSP processes in *normal form representation* according to the proof rules introduced by Apt and Olderog [5]. The link between normal form processes and their corresponding behavioural specifications can be realized by means of auxiliary variables representing traces and refusals, as described by the author in [71].

The second approach has already been illustrated in [71], we will therefore skip the proofs here.

5.5 Verification of Deadlock Freedom

The verification in the trace model of CSP performed above may be informally summarized as: "*If* values are delivered on output channels, they will be correct with respect to the system requirements". As a by-product, the verification process also produced estimates for the number of messages to be delivered over output channels. For example, Proof Obligation 2 (p. 50) showed for the security layer that the number of messages delivered via output channel *abp_rc* grows with the number of inputs on *abp_tx*. Analogous estimates may be given for channels *abp_ack_rc* (Proof Obligation 3) and finally for *app_rc*. These estimates simplify the proof of liveness properties: It only has to be shown that the complete system will allow an unbounded number of events on certain input channels. For our case study these channels are *app_tx*, *abp_tx* and *abp_ack_tx*. Formally, the obligation to allow unbounded numbers of events on a channel may be expressed using a refinement relation in the failures-divergence model of CSP. For the case study we have the following proof obligations:

Liveness Proof Obligation 1: For $P_1 = app_tx?x \rightarrow P_1$ show that

$$P_1 \sqsubseteq_{FD} (ESYS \setminus (\alpha(SYS) - \{\!|\ app_tx\ |\!\}))$$

Liveness Proof Obligation 2: For $P_2 = abp_tx?x \rightarrow P_2$ show that

$$P_2 \sqsubseteq_{FD} ESYS \setminus (\alpha(SYS) - \{\!|\ abp_tx\ |\!\}))$$

Liveness Proof Obligation 3: For $P_3 = abp_ack_tx?x \rightarrow P_3$ show that

$$P_3 \sqsubseteq_{FD} ESYS \setminus (\alpha(SYS) - \{\!|\ abp_ack_tx\ |\!\}))$$

In these obligations, *ESYS* stands for the completely developed system and its environment with explicit representations of $ABP_TX, \ldots, SEC_ACK_RC$:

$$\begin{aligned} ESYS = {} & (PROD \,|||\, CON) \,\| \\ & (ABP_TX \,|||\, ABP_RC) \,\| \\ & (SEC_TX \,|||\, (SEC_RC \wr \Delta_{SEC,1}) \,|||\, SEC_ACK_TX \,|||\, SEC_ACK_RC) \,\| \\ & ((M \wr (\Delta_{M,2} \wr \Delta_{M,1})) \,|||\, (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))) \end{aligned}$$

Since these obligations only refer to the possible *occurrences* of events, but do not require to consider actual data values, it seems promising to try a mechanized proof. To this end, let us follow the technique of *abstract interpretations* for distributed systems (see Clarke *et al.* [13]) and analyze the following processes that represent simplified but less deterministic

versions of the corresponding processes $ABP_TX, \ldots$:

$$
\begin{array}{lll}
ATX_1 & = app_tx?x \rightarrow ATX_{11} \,\Box\, abp_ack_rc?y \rightarrow ATX_1 \\
ATX_{11} & = (\sqcap_{z:\alpha(abp_tx)} abp_tx!z \rightarrow ATX_{11}) \,\Box\, (abp_ack_rc?x \rightarrow (ATX_1 \sqcap ATX_{11})) \\
ARC_1 & = abp_rc?x \rightarrow \\
 & \qquad (\sqcap_{z:\alpha(app_rc)} app_rc!z \rightarrow (\sqcap_{w:\alpha(abp_ack_tx)} abp_ack_tx!w \rightarrow ARC_1)) \\
 & \qquad \sqcap \\
 & \qquad (\sqcap_{w:\alpha(bp_ack_tx)} abp_ack_tx!w \rightarrow ARC_1) \\
STX_1 & = abp_tx?x \rightarrow (\sqcap_{y:\alpha(tx)} tx!y \rightarrow STX_1) \\
SRC_1 & = SRC_1(m_1) \\
SRC_1(n) & = rc?x \rightarrow \\
 & \qquad (\textbf{if } (n = 0) \\
 & \qquad \textbf{then } (\sqcap_{y:\alpha(abp_rc)} abp_rc!y \rightarrow SRC_1(m_1)) \\
 & \qquad \textbf{else } (SRC_1(n-1) \sqcap (\sqcap_{y:\alpha(abp_rc)} abp_rc!y \rightarrow SRC_1(m_1)))) \\
M_1 & = M_1(maxLoss) \\
M_1(n) & = tx?x \rightarrow \\
 & \qquad (\textbf{if } (n = 0) \\
 & \qquad \textbf{then } (\sqcap_{y:\alpha(rc)} rc!y \rightarrow M_1(maxLoss)) \\
 & \qquad \textbf{else } (M_1(n-1) \sqcap (\sqcap_{y:\alpha(rc)} rc!y \rightarrow M_1(maxLoss)))) \\
MACK_1 & = MACK_1(maxLoss) \\
MACK_1(n) & = ack_tx?x \rightarrow \\
 & \qquad (\textbf{if } (n = 0) \\
 & \qquad \textbf{then } (\sqcap_{y:\alpha(ack_rc)} ack_rc!y \rightarrow MACK_1(maxLoss)) \\
 & \qquad \textbf{else } (MACK_1(n-1) \sqcap (\sqcap_{y:\alpha(ack_rc)} ack_rc!y \rightarrow MACK_1(maxLoss))))
\end{array}
$$

In these definitions, m_1 is a constant derived from Lemma 5. It indicates that SRC_1 will not refuse output for an unbounded number of input events.

Obviously, $ATX_1 \sqsubseteq_{FD} ABP_TX$, because the communication structure of ATX_1 is the same as in ABP_TX, but for ATX_1 the degree of nondeterminism has been increased by replacing internal decisions of ABP_TX by the $\sqcap$-operator. Similarly, $ARC_1, \ldots, MACK_1$ are refined by $ABP_RC, \ldots, (MACK \wr (\Delta_{MACK,2} \wr \Delta_{MACK,1}))$ in the failures-divergence model. Therefore Liveness Proof Obligations 1 to 3 are implied by the corresponding liveness proofs for system $ESYS_1 = ESYS[ATX_1/ABP_TX, \ldots]$.

Next observe that the behaviour of $ATX_1, \ldots, MACK_1$ does not depend on the values received on any input channel. Therefore a liveness property referring only to the occurrence of *any* channel event holds for $ESYS_1$ if and only if it holds for $ESYS_2 = ESYS_1[ATX_2/ATX_1, \ldots]$

with processes

$$
\begin{array}{lll}
ATX_2 & = app_tx_2 \rightarrow ATX_{21} \;\Box\; abp_ack_rc_2 \rightarrow ATX_2 \\
ATX_{21} & = (abp_tx_2 \rightarrow ATX_{21}) \;\Box\; (abp_ack_rc_2 \rightarrow (ATX_2 \sqcap ATX_{21})) \\
ARC_2 & = abp_rc_2 \rightarrow \\
 & \qquad (app_rc_2 \rightarrow (abp_ack_rc_2 \rightarrow ARC_2)) \\
 & \qquad \sqcap \\
 & \qquad (abp_ack_tx_2 \rightarrow ARC_2) \\
STX_2 & = abp_tx_2 \rightarrow (tx_2 \rightarrow STX_2) \\
SRC_2 & = SRC_2(m_1) \\
SRC_2(n) & = rc_2 \rightarrow \\
 & \qquad (\textbf{if } (n = 0) \\
 & \qquad \textbf{then } (abp_rc_2 \rightarrow SRC_2(m_1)) \\
 & \qquad \textbf{else } (SRC_2(n-1) \sqcap (abp_rc_2 \rightarrow SRC_2(m_1)))) \\
M_2 & = M_2(maxLoss) \\
M_2(n) & = tx_2 \rightarrow \\
 & \qquad (\textbf{if } (n = 0) \\
 & \qquad \textbf{then } (rc_2 \rightarrow M_2(maxLoss)) \\
 & \qquad \textbf{else } (M_2(n-1) \sqcap (rc_2 \rightarrow M_2(maxLoss)))) \\
MACK_2 & = MACK_2(0) \\
MACK_2(n) & = ack_tx_2 \rightarrow \\
 & \qquad (\textbf{if } (n = 0) \\
 & \qquad \textbf{then } (ack_rc_2 \rightarrow MACK_2(maxLoss)) \\
 & \qquad \textbf{else } (MACK_2(n-1) \sqcap (ack_rc_2 \rightarrow MACK_2(maxLoss))))
\end{array}
$$

In these definitions $app_tx_2, abp_ack_tx_2, \ldots$ are no longer channels but single events.

System $ESYS_2$ fulfills the requirements for the FDR model checker [24]: every sequential process has only a finite number of states and works with finite alphabets only. Indeed, model checking with FDR shows that the Liveness Proof Obligations 1 to 3 hold for system $ESYS_2$, as was to be shown.

6. Conclusion

In this report we have presented a re-usable and in parts standardized framework for the development of dependable systems. Although the framework is based on the *V-Model*, it may be adapted to other standards (for example, [19]) with only minor modifications, like using other names for the documents and slightly different orderings of the items to be described. This global applicability of the framework is an important aspect, because there seems to be small chance for an internationally accepted development standard to emerge from the various standards presently discussed. On the other hand, we do not consider the *formal method* applied in this chapter to be universally applicable. It just served as an example of a technique well suited for the specification and verification of dependable systems possessing specific characteristics listed in 4.2.1.

The method used has its limitations which might be critical in other applications. For example, you might want to

- specify complex data structures
- use explicit and implicit specification styles in combination
- simulate the specification
- perform a risk analysis based on formal specifications
- describe hybrid systems
- use generic specifications allowing re-use for different
- describe tightly coupled shared memory systems
- ...

This list is by no means considered as complete; it just gives an impression how many features might be desirable in the context of dependable systems development, but can never be united in a single method (and certainly never be implemented in a single tool). For these reasons, I do not believe that one universal method will emerge out of the (potentially non-terminating) competition of (formal) methods. Instead, different techniques will be applied to rather narrowly defined problem types, where they promise optimal efficiency. Indeed, I am convinced that a change of methods during the different development stages of one project will become quite natural in the future.

The advantage of combining different methods and associated tools leads to two important obligations:

- On the level of methodology it has to be ensured that objects described and results derived in the context of one method are consistently transformed into the framework of another method to be used in a further development step.

- On the tool level it has to be ensured that the transfer of development objects (specifications, code, test cases, proofs, ...) between different tools is not only "somehow" implemented, but consistent with the semantic properties of the associated methods. A specific problem is presented by the combination of semi-formal CASE tools with formal specification and verification techniques, because reasoning in the formal framework requires precise semantics, and this is not *a priori* available for CASE tools in general.

Making use of the results developed in the ESPRIT project ProCoS (*Provably Correct Systems*) [10], these obligations will be analyzed in detail in the UniForM project [85], proposed by the author in collaboration with ELPRO LET GmbH and the universities of Bremen and Oldenburg, and supported by the German Ministry *Bundesministerium für Bildung und Forschung BMBF*. Further applications of the framework as described in this chapter are planned in the context of the UniForM project (case study) and in the project [23] (large-scale development of a "real-world" system).

The investigation of formal semantics for CASE methods [17, 34, 106, 107] has been extensively investigated by the author in collaboration with de Roever, Hamer, Hörcher, Huizing and Petersohn [73, 76, 77, 78, 79, 81, 82]. Further notable efforts in this direction have been presented by other autors in [12, 8, 32, 33, 40, 42, 51].

Bibliography

[1] DA Airbus Industrie No 2370M1F000101: *Cabin Intercommunication Data System System Specification, A330/340, CIDS* Volume I, Issue 5 (1993).

[2] K. R. Apt, L. Bougé and Ph. Clermont: Two Normal Form Theorems for CSP Programs. *Information Processing Letters* (1987) 26:165-171.

[3] K. R. Apt, N. Francez and W. P. de Roever: A Proof System for Communicating Sequential Processes. *ACM Transactions on Programming Languages and Systems* (1980) 9: 359-385.

[4] K. R. Apt: Correctness Proofs of Distributed Termination Algorithms. *ACM Trans. on Progr. Lang. and Syst.* (1986) 8: 388-405.

[5] K. R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs.* Springer-Verlag, Berlin Heidelberg New York (1991).

[6] R. Atzmüller: *Das REMO Referenzmodell.* BSI (1994).

[7] F. Belli, K. Echtle and W. Görke: Methoden und Modelle der Fehlertoleranz. *Informatik Spektrum* (1986) 9: 68-81.

[8] A. Benveniste and G. Berry. *The Synchronous Approach to Reactive and Real-Time Systems*, in IEEE-Proceedings *"Another Look at Real-Time Programming"*, 1992.

[9] P. Bieber and N. Boulahia-Cuppens: Formal Development of Authentication Protocols. In D. Till (Ed.): *6th Refinement Workshop.* Proceedings of the 6th Refinement Workshop, organised by BCS FACS, London, 5-7 January 1994, Springer-Verlag, Berlin Heidelberg New York (1994) 80-102.

[10] D. Björner, H. Langmaack and C. A. R. Hoare: *ProCoS – Provably Correct Systems.* Esprit BRA 3104. Technical University of Denmark, Lyngby (1993).

[11] E. Brinksma: A theory for the deriviation of tests. In P. H. J. van Eijk, C. A. Vissers and M. Diaz (Eds.): *The Formal Description Technique LOTOS.* Elsevire Science Publishers B. V. (North-Holland), (1989) 235-247.

[12] M. Broy: Towards a Formal Foundation of the Specification and Description Language SDL. *Formal Aspects of Computing* (1991) 3: 21-57

[13] E. M. Clarke, O. Grümberg and D. E. Long: Model Checking and Abstraction. In *Proc. of the 19th Annual ACM Symposium on Principles of Programming Languages (POPL '92)* (1992).

[14] *Common Criteria for Information Technology Security Evaluation.* Draft Version 0.6. CCEB-94/034.

[15] J. Davies: *Specification and Proof in Real-Time CSP.* Cambridge University Press (1993).

[16] D. W. Davies: Protection. In B. E. Lampson, M. Paul, H. J. Siegert (Eds.): *Distributed Systems.* Springer-Verlag, Berlin Heidelberg New York (1983) 211-245.

[17] T. DeMarco: *Structured Analysis and System Specification.* Prentice-Hall International, Englewood Cliffs NJ (1979).

[18] Using Z and DST-fuzz, an Introduction. DST Deutsche System-Technik GmbH, Kiel (1993).

[19] RTCA DO178B: *Development considerations in airborne computer systems.* (1993).

[20] E. W. Dijkstra: Self-stabilizing systems in spite of distributed control. *Communications of the ACM* (1974) 17: 643-644.

[21] *DIN ISO 9000-3: Qualitätsmanagement- und Qualitätssicherungsnormen – Leitfaden für die Anwendung von ISO 9001 auf die Entwicklung, Lieferung und Wartung von Software.* DIN Deutsches Institut für Normung e. V., Beuth Verlag Berlin (1990).

[22] ELPRO LET GmbH: *Programmablaufplan - Bahnübergang.* ELPRO LET GmbH (1994).

[23] ELPRO LET GmbH: *Radio-based Integrated Control Information and Signalling System for Railways.* Proposal for EU Telematics Applications Programme 'Telematics for Transport Sector', ELPRO LET GmbH (1995).

[24] Formal Systems: *Failures Divergence Refinement.* User Manual and Tutorial Version 1.4. Formal Systems (Europe) Ltd (1994).

[25] Federal Criteria for Information Technology Security, Volume I, Version 1.0. U. S. National Institute of Standards and Technology & National Security Agency (1992).

[26] Federal Criteria for Information Technology Security, Volume II, Version 1.0. U. S. National Institute of Standards and Technology & National Security Agency (1992).

[27] M.-C. Gaudel: Testing can be formal, too. In P. D. Mosses, M. Nielsen and M. I. Schwartzbach (Eds.): *Proceedings of TAPSOFT '95: Theory and Practice of Software Development.* Aarhus, Denmark, May 1995, Springer-Verlag, Berlin Heidelberg New York (1995).

[28] W. W. Gibbs: Software chronisch mangelhaft. *Spektrum der Wissenschaft* (1994) 12: 56-63.

[29] J. Graham-Cunning and J. W. Sanders: On the Refinement of Non-interference. In *Proceedings of the Computer Security Foundations Workshop IV* IEEE (1991).

[30] J. Graham-Cunning: Some Laws of Non-interference. In *Proceedings of the Computer Security Foundations Workshop V* IEEE (1992).

[31] M. Grochtmann and K. Grimm: Classification Trees for Partition Testing. *Software Testing, Verification and Reliability* (1993) 3: 63-82.

[32] D. Harel, A. Pnueli, J. Pruzan-Schmidt and R. Sherman. *On the formal semantics of Statecharts.* In *Proceedings Symposium on Logic in Computer Science*, (1987) 54-64.

[33] D. Harel. *On visual formalisms.* Communications of the ACM (1988) 31: 514-530.

[34] D. J. Hatley and I. A. Pirbhai: *Strategies for Real-Time System Specification.* Dorset House, New York (1987).

[35] M. Heisel and D. Weber-Wulff: Korrekte Software: Nur eine Illusion? *Informatik Forschung und Entwicklung* (1994) 9: 192-200

[36] M. C. Hennessy: *Algebraic Theory of Processes.* MIT Press (1988).

[37] M. C. Hennessy and A. Ingólfsdóttir: Communicating Processes with Value-passing and Assignments. *Formal Aspects of Computing* (1993) 5: 432-466

[38] H. M. Hörcher: Improving Software Tests using Z Specifications. To appear in J. P. Bowen and M. G. Hinchey (Eds.): *ZUM '95: 9th International Conference of Z Users*, LNCS, Springer-Verlag, Berlin Heidelberg New York (1995).

[39] HP-UX System Security. Hewlett-Packard Company (1991).

[40] C. Huizing, R. Gerth and W.-P. de Roever. *Modelling statecharts behaviour in a fully abstract way.* In *Proc. 13th CAAP*, LNCS 299, Springer-Verlag, Berlin Heidelberg New York (1988) 271-294.

[41] C.A.R. Hoare. *Communicating sequential processes.* Prentice-Hall International, Englewood Cliffs NJ (1985).

[42] C. Huizing and R.T. Gerth. *Semantics of Reactive Systems in Abstract Time*, in *"Real-Time: Theory in Practice"*, proceedings of a REX workshop, June 1991, Mook, edited by J.W. de Bakker, W.-P. de Roever, G. Rozenberg, LNCS 600, Springer-Verlag, Berlin Heidelberg New York, 1992.

[43] Inmos Ltd., *Occam 2 Reference Manual.* Prentice-Hall International, Englewood Cliffs NJ, (1988).

[44] Information Technology Security Evaluation Criteria (ITSEC). Version 1.2. EGKS-EWG-EAG, Brussels • Luxemburg (1991).

[45] J. Jacob: Security Specifications. In *IEEE Symposium on Security and Privacy '88* IEEE (1988).

[46] P. Jalote: Fault tolerant processes. *Distributed Computing* (1989) 3: 187-195.

[47] M. K. Joseph: Integration Problems in Fault Tolerant, Secure Computer Design. In A. Avižienis, J. C. Laprie (Eds.): *Dependable Computing for Critical Applications.* Springer-Verlag, Berlin Heidelberg New York (1991) 347-364.

[48] He Jifeng and C. A. R. Hoare: Algebraic specification and proof of a distributed recovery algorithm. *Distributed Computing* (1987) 2: 1-12.

[49] C. B. Jones: *Systematic Software Development Using VDM.* Prentice-Hall International, Englewood Cliffs NJ (1986).

[50] E. Jonsson and T. Olovsson: Security in a Dependability Perspective. In *Proc. of Nordic Seminar on Dependable Computing NSDCS '94.*

[51] Y. Kesten and A. Pnueli: Timed and hybrid statecharts and their textual representation. In J. Vytopil (Ed.) *Formal Techniques in Real-Time and Fault-Tolerant Systems.* Springer-Verlag, Berlin Heidelberg New York (1992) 591-619.

[52] B. W. Lampson, M. Paul and H. J. Siegert (Eds.): *Distributed Systems.* Springer-Verlag, Berlin Heidelberg New York (1983).

[53] J. C. Laprie: Dependable Computing and Fault Tolerance: Concepts and Terminology. In *Proc. 15th IEEE International Symposium on Fault-Tolerant Computing (FTCS-15)*, (1985).

[54] J. C. Laprie et al.: *Dependability: Basic Concepts and Terminology.* Springer-Verlag, Berlin Heidelberg New York (1992).

[55] J. C. Laprie: Dependability: a unifying concept for reliable, safe and secure computing. In *Proc. of the 12th IFIP World Computer Congress, Madrid, Spain.* (1992).

[56] P. A. Lee and T. Anderson: *Fault tolerance: Principles and practice.* Springer-Verlag, Berlin Heidelberg New York (1990).

[57] J. Loeckx and K. Sieber: *The Foundations of Program Verification.* 2nd Ed.-Wiley,Teubner, Stuttgart (1987).

[58] T. F. Lunt and R. A. Whitehurst: *The Seaview Formal Top Level Specification.* A007: Final Report, Volume 3A. SRI International, Menlo Park, CA, (1989).

[59] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, Berlin Heidelberg New York (1992).

[60] E. Mikk: Compilation of Z Specifications into C for Automatic Test Result Evaluation. To appear in J. P. Bowen and M. G. Hinchey (Eds.): *ZUM '95: 9th International Conference of Z Users*, LNCS, Springer-Verlag, Berlin Heidelberg New York (1995).

[61] R. Milner: *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs NJ (1989).

[62] Requirements for the procurement of safety critical software in defence equipment, *Interim Defence Standard 00-55*, Ministry of Defence, Directorate of Standardization, Kentigern House, 65 Brown St., Glasgow G2 8EX, May (1989).

[63] C. C. Morgan: *Programming from Specifications.* Prentice-Hall International, Englewood Cliffs NJ (1990).

[64] C. C. Morgan, K. A. Robinson and P. H. B. Gardiner: On the refinement calculus. *Oxford University Computing Laboratory.* PRG-70 (1989).

[65] M. Müllerburg: Systematic Testing: a Means for Validating Reactive Systems. In *EuroSTAR'94: Proceedings of the 2nd European Intern. Conf. on Software Testing, Analysis&Review.* British Computer Society, (1994).

[66] R. de Nicola and M. C. Hennessy: Testing Equivalence for Processes. *Journal of Theoretical Computer Science* (1983)34: 83-133

[67] J. Nordahl: *Specification and Design of Dependable Communicating Systems.* Thesis, Department of Computer Science, Technical University of Denmark, Lyngby (1992).

[68] E.-R. Olderog: *Nets, Terms and Formulas.* Cambridge University Press (1991).

[69] J. Peleska: A characterization for isometries and conformal mappings of pseudo-Riemannian manifolds. *Aequationes Mathematicae*(1984)27: 20-31.

[70] J. Peleska: *Das PST-NET Dopelrechnersystem.* Technical Report, Philips GmbH, Unternehmensbereich Systeme und Sondertechnik (1989).

[71] J. Peleska: Design and verification of fault tolerant systems with CSP. *Distributed Computing* (1991) 5: 95-106.

[72] J. Peleska: Formale Methoden beim Entwurf ausfallsicherer, verteilter Systeme. In Lippold, Schmitz (Eds.): *Sicherheit in netzgestützten Informationssystemen.* Proceedings des BIFOA-Kongresses SECUNET '92, Vieweg (1992) 293-308.

[73] J. Peleska: CSP, Formal Software-Engineering and the Development of Fault-Tolerant Systems. In Vytopil (Ed.): *Formal Techniques in Real-Time and Fault-Tolerant Systems.* Kluwer Academic Publishers (1993) 167-207.

[74] J. Peleska: Formale Spezifikation generischer ITSEC Funktionalitätsklassen. In Reichel (Ed.): Informatik, Wirtschaft, Gesellschaft. 23. GI Jahrestagung, Springer-Verlag, Berlin Heidelberg New York (1993) 354-364.

[75] J. Peleska: A Unified Approach for the Formal Specification and Verification of Safety- and Security Requirements. In *Proceedings of the Nordic Seminar on Dependable Computing '94* (invited keynote).

[76] U. Hamer, H. M. Hörcher and J. Peleska: Safer Software - an Introduction into Formal Software Engineering. Technical Report, DST (1993).

[77] J. Peleska: Formales Software-Engineering mit strukturierten Methoden und Z: nachweisbare Korrektheit für sicherheits-relevante Anwendungen. In Scheibel (Ed.): *Software-Entwicklung - Methoden, Werkzeuge, Erfahrungen '93.* Technische Akademie Esslingen (1993) 753-762.

[78] J. Peleska, C. Huizing and C. Petersohn: A Comparison of Ward&Mellor's Transformation Schema with State-&Activitycharts. Computing Science Note 94/11, Eindhoven University of Technology (1994) – Submitted for publication in *IEEE Transactions on Software Engineering.*

[79] C. Petersohn, C. Huizing, J. Peleska and W. P. de Roever: Formal Semantics for Ward&Mellor's Transformation Schemas. In D. Till (Ed.): *6th Refinement Workshop.* Proceedings of the 6th Refinement Workshop, organised by BCS FACS, London, 5-7 January 1994, Springer-Verlag, Berlin Heidelberg New York (1994) 14-41.

[80] H. M. Hörcher and J. Peleska: The Role of Formal Specifications in Software Test. Tutorial, held at the FME '94.

[81] J. Peleska: Simulation und Wirklichkeit - Stellungnahme aus Sicht der Industrie. In *Computersimulation: (k)ein Spiegel der Wirklichkeit.* Bundesamt für Sicherheit in der Informationstechnik, SecuMedia-Verlag (1994).

[82] U. Hamer and J. Peleska: Z Applied to the A330/340 CIDS Cabin Communication System. In M. Hinchey, J. Bowen (Eds.) *Applications of Formal Methods.* Prentice-Hall International, Englewood Cliffs NJ (1995).

[83] J. Peleska: *Bahnübergangssteuerung Straßenbahn – ELPRO LET GmbH: Prüfspezifikation für formale Verifikation und automatisierte Testdurchführung.* JP Software-Consulting (1994).

[84] J. Peleska: *Bahnübergangssteuerung Straßenbahn – ELPRO LET GmbH: Sicherheitsspezifikation und BUE-Spezifikation.* JP Software-Consulting (1994).

[85] A. Baer, B. Krieg-Brückner, E.-R. Olderog and J. Peleska: *Universelle Entwicklungsumgebung für Formale Methoden (UniForM).* Vorhaben im Fördergebiet 'Softwaretechnologie' des BMBF (1995).

[86] J. Peleska: *Trustworthy Tests for Reactive Systems – Automation of Real-Time Testing.* In preparation, JP Software-Consulting (1995).

[87] J. Peleska: *Testautomatisierung für diskrete Steuerungen Anwendung: Bahnübergangssteuerung Abschlußbericht Phase 1* JP Software-Consulting (1995).

[88] J. Peleska: *Trustworthy Automated Tests for Reactive Systems* Technical Report 0595, JP Software-Consulting (1995).

[89] G. Plotkin: *An operational semantics for CSP.* In *Proceedings of the IFIP Conference on the Formal Description of Programming Concepts II,* North Holland (1983) 199-225.

[90] N. Pohlmann: Sicherheit der Kommunikation in Netzen. In: *Proceedings of the 3. Deutscher IT-Sicherheitskongreß des BSI 1993.* Bundesamt für Sicherheit in der Informationstechnik. SecuMedia, Ingelheim (1994) 154-180.

[91] A. P. Ravn and V. Stavridou: *Criteria for specification and programing languages for engineering safety-critical software.* Technical report, Technical University of Denmark (ID/DTH) (1995).

[92] A. P. Ravn and V. Stavridou: *Specification and development of safety-critical software: An assessment of MOD draft Standard 00-55.* Technical report, Technical University of Denmark (ID/DTH) (1995).

[93] G. Reed and A. W. Roscoe: A timed model for Communicating Sequential Processes. In: *Proc. 13th International Colloquium on Automata, Languages and Programming.* LNCS 226, Springer-Verlag, Berlin Heidelberg New York (1986) 314-323.

[94] A. W. Roscoe and G. Barret: Unbounded Nondeterminism in CSP. In *MFPS '89*, LNCS 298, Springer-Verlag, Berlin Heidelberg New York (1989).

[95] A. W. Roscoe: Model-Checking CSP. In *A Classical Mind, Essays in Honour of CAR Hoare.* Prentice-Hall International, Englewood Cliffs NJ (1994).

[96] A. W. Roscoe, J. C. P. Woodcock and L. Wulf: Non-interference through Determinism. Oxford University Computing Laboratory (1994).

[97] J. Rushby: Critical system properties: survey and taxonomy. *Reliability Engineering and System Safety* 43: 189-219 (1994).

[98] H. Schepers: Terminology and Paradigms for Fault Tolerance. In Vytopil (Ed.): *Formal Techniques in Real-Time and Fault-Tolerant Systems.* Kluwer Academic Publishers (1993) 3-31.

[99] H. Schepers: *Fault Tolerance and Timing of Distributed Systems.* Thesis, Eindhoven University of Technology (1994).

[100] M. Schneider: Self-stabilization. *ACM Computing Surveys* (1993) 25: 45-67.

[101] K. G. Shin and P. Ramanathan: Real-Time Computing: A new Discipline of Computer Science and Engineering. *Proceedings of the IEEE* (1994) 82: 6-23.

[102] M. J. Spivey. *The Z Notation.* Prentice-Hall International, Englewood Cliffs NJ (1992).

[103] M. E. Stickel: Elimination of Inference Channels by Optimal Upgrading. In *IEEE Symposium on Security and Privacy '94* IEEE (1994).

[104] U. S. Department of Defense Trusted Computer System Evaluation Criteria (TCSEC). DoD 5200.28-STD (1985).

[105] Der Bundesminister des Innern: *Planung und Durchführung von IT-Vorhaben – Vorgehensmodell.* KBSt Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (1992). English version available from IABG Industrieanlagen Betriebsgesellschaft, Ottobrunn (1992).

[106] P. T. Ward: *The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing. IEEE Transactions on Software Engineering* (1986) SE-12: 198-210.

[107] P. T. Ward and S. J. Mellor: *Structured Development for Real-Time Systems.* (3 vols), Yourdon Press Computing Series, Prentice-Hall International, Englewood Cliffs NJ Cliffs, 1985.

[108] J. Wegener, R. Pitschinetz, K. Grimm and M. Grochtmann: TESSY - Yet Another Computer-Aided Software Testing Tool? Daimler-Benz AG, Forschung und Technik, Berlin (1994).

[109] R. A. Whitehurst and T. F. Lunt: The Seaview Verification Effort. In *Proceedings of the 12th National Computer Security Conference.* Baltimore, MD, (1989).

[110] J. B. Wordsworth : *Software Development with Z.* Addison-Wesley,Wokingham (U.K.) (1992).

[111] Zhiming Liu, E. V. Sørensen, A. P. Ravn and Chaochen Zhou: Towards a Calculus of System Dependability. *Journal of high integrity systems* (1994) 1: 49-65 .

[112] D. Zöbel: Normalform-Transformationen für CSP-Programme. *Informatik Forsch. Entw.* (1988) 3: 64-76.

[113] J. Zwiers: *Compositionality, Concurrency and Partial Correctness.* LNCS 321, Springer-Verlag, Berlin Heidelberg New York (1989).

Appendix A. Glossary of Symbols

Functions

Symbol	Meaning
$f : X \longrightarrow Y$	function f with domain X and range Y
$f : X \nrightarrow Y$	partial function f
$f : X \twoheadrightarrow Y$	surjective function f
$f : X \rightarrowtail Y$	injective function f
$f : X \rightarrowtail\!\!\!\!\twoheadrightarrow Y$	bijective function f
$\mathrm{dom}\, f$	domain of f
$\mathrm{ran}\, f$	range of f
$f \oplus g$	function f overridden on dom g by values of g

Traces

Symbol	Meaning	Example
$s = \langle s(1), s(2), s(3), \ldots \rangle$	trace notation	
$\langle\,\rangle$	empty trace	
$\#s$	length of s	
$\mathrm{ran}\, s$	range of s	$\mathrm{ran}\langle a, b, c, b, b\rangle = \{a, b, c\}$
$head(s)$	head of s	$head(\langle a, b, c, b, b\rangle) = a$
$front(s)$	front of s	$front(\langle a, b, c, b, b\rangle) = \langle a, b, c, b\rangle$
$tail(s)$	tail of s	$tail(\langle a, b, c, b, b\rangle) = \langle b, c, b, b\rangle$
$last(s)$	last of s	$last(\langle a, b, c, b, b\rangle) = b$
$s \leq^n u$	s prefix of u, $\#u - \#s \leq n$	$\langle a, b\rangle \leq^1 \langle a, b, c\rangle$
$s \trianglelefteq u$	s subtrace of u	$\langle a, c\rangle \trianglelefteq \langle a, b, c, d\rangle$
s **in** u	s segment of u	$\langle b, c\rangle$ **in** $\langle a, b, c, d\rangle$
$s \frown u$	concatenation of s and u	$\langle b, c\rangle \frown \langle a, b, c, d\rangle = \langle b, c, a, b, c, d\rangle$
$s \restriction X$	s projected on elements of set V	$\langle a, b, c, d, c\rangle \restriction \{a, c\} = \langle a, c, c\rangle$
$f^*(s)$	function f applied to each element of s	$f^*(\langle b, c\rangle) = \langle f(b), f(c)\rangle$
$ch^*(s)$	trace s projected on its channels	$ch^*(\langle c.x, d.y\rangle) = \langle c, d\rangle$
$val^*(s)$	trace s projected on its values	$val^*(\langle c.x, d.y\rangle) = \langle x, y\rangle$
$\{\!\mid c \mid\!\}$	set of all channel events of c	

CSP Processes

Symbol	**Meaning**
$\alpha(P)$	alphabet of P
$Traces(P)$	traces of P
$Fail(P)$	failures of P
$Ref(P)$	refusals of P
$refMax(P)$	maximal refusals of P
$Div(P)$	divergences of P
$[P]^0$	next events possible for process P:
	$\{e \in \alpha P \mid (\exists\, u \in Traces(P) \bullet head(u) = e)\}$
$STOP$	deadlock process
$SKIP$	terminating process, producing event $\surd$
$a \rightarrow P$	prefixing operator
$P \sqcap Q$	internal choice
$(\sqcap_{x:\{a_1,\ldots,a_n\}}(x \rightarrow P(x))$	abbreviates $a_1 \rightarrow P(a_1) \sqcap \ldots \sqcap a_n \rightarrow P(a_n)$
$P \,\Box\, Q$	external choice
$(x : \{a_1, \ldots, a_n\} \rightarrow P(x))$	abbreviates $a_1 \rightarrow P(a_1) \,\Box \ldots \Box\, a_n \rightarrow P(a_n)$
$P\,;Q$	sequential composition
$X = F(X)$,	
$\mu\, X \bullet F(X)$	recursive equation
$g \& B$	guarded command, abreviates
	if g **then** B **else** $STOP$
$P \parallel Q$ parallel operator	
$P \,\|\|\|\, Q$	interleaving operator
$P \setminus H$	hiding operator
$P \hat{} Q$	interrupt operator
$(P \backslash \Delta_P)$	threat introduction operator

Appendix B. A Quick-Reference Guide to CSP

The CSP semantics $[\![\bullet]\!]$ for nondeterministic processes [41] maps each syntactic CSP unit P onto a triple of sets

$$\mathcal{I}[\![\, P \,]\!] = (A, F, D)$$

subject to the following conditions:

D0: A, F, D have the following structure:

- A is a finite set, called the *alphabet* of P, denoted $A = \alpha P$.
- $F \subseteq A^* \times \mathbb{P}\, A$ is called the *failures* of P, denoted $F = \mathit{Fail}(P)$. For each element $(s, X) \in F$, s is called a *trace* of P and X a *refusal (set)* of P/s. The set of all traces is denoted by $\mathit{Traces}(P)$.
- $D \subseteq A^*$ is called the *divergences* of P, denoted $D = \mathit{Div}(P)$.

C0: $(\langle\,\rangle, \varnothing) \in F$

C1: $(s \frown t, X) \in F \Rightarrow (s, \varnothing) \in F$

C2: $(s, Y) \in F \wedge X \subseteq Y \Rightarrow (s, X) \in F$

C3: $(s, X) \in F \wedge x \in A \Rightarrow$
$(s, X \cup \{x\}) \in F \vee (s \frown \langle x \rangle, \varnothing) \in F$

C4: $D \subseteq \mathrm{dom}\, F$

C5: $s \in D \wedge t \in A^* \Rightarrow s \frown t \in D$

C6: $s \in D \wedge X \subseteq A \Rightarrow (s, X) \in F$

Three refinement relations between processes P, Q are defined by means of the semantics $[\![P]\!] = (A, F, D)$ and $[\![Q]\!] = (A', F', D')$:

Trace Refinement: $P \sqsubseteq_T Q \equiv_{df} A = A' \wedge \mathit{Traces}(Q) \subseteq \mathit{Traces}(P)$

Failures Refinement: $P \sqsubseteq_F Q \equiv_{df} A = A' \wedge F' \subseteq F$

Failures-Divergence Refinement: $P \sqsubseteq_F Q \equiv_{df} A = A' \wedge F' \subseteq F \wedge D' \subseteq D$

We explain the meaning of CSP operators by giving their failure sets and their divergences.

Nondeterministic OR Operator The OR operator $\sqcap$ selects one of its operands nondeterministically without giving th environment the possibility to influence this decision:

$$Fail(P \sqcap Q) = Fail(P) \cup Fail(Q)$$

$$Div(P \sqcap Q) = Div(P) \cup Div(Q)$$

Prefixing Operator The prefixing operator $\rightarrow$ executes $P(x)$ after occurrence of an initial event $x \in B$.

$$Fail(x : B \rightarrow P(x)) = \\ \{(\langle\,\rangle, U) \mid U \subseteq (\alpha P - B)\} \cup \{(\langle x \rangle \frown s, U) \mid x \in B \wedge (s, U) \in Fail(P(x))\}$$

$$Div(x : B \rightarrow P(x)) = \{\langle x \rangle \frown s \mid s \in Div(P(x)) \wedge x \in B\}$$

Parallel Operator The parallel operator $\parallel$ defines joint execution of P and Q with events common to both their alphabets executed synchronously.

$$Fail(P \parallel Q) = \\ \{(s, U \cup V) \mid s \in (\alpha P \cup \alpha Q)^* \wedge \\ (s \upharpoonright \alpha P, U) \in Fail(P) \wedge (s \upharpoonright \alpha Q, V) \in Fail(Q)\} \cup \\ \{(t, U) \mid t \in Div(P \parallel Q)\}$$

$$Div(P \parallel Q) = \\ \{s \frown t \mid t \in (\alpha P \cup \alpha Q)^* \wedge \\ (s \upharpoonright \alpha P \in Div(P) \wedge s \upharpoonright \alpha Q \in Traces(Q) \vee \\ s \upharpoonright \alpha Q \in Div(Q) \wedge s \upharpoonright \alpha P \in Traces(P))\}$$

General Choice Operator The alternative operator $\Box$ executes P if the initial event is from P's alphabet and not from Q's alphabet, executes Q if the initial event is from Q's alphabet and not from P's alphabet and choses nondeterministically if the initial event is in the intersection of their alphabets.

$$Fail(P \Box Q) = \\ \{(s, U) \mid (s, U) \in Fail(P) \cap Fail(Q) \vee (s \neq \langle\,\rangle \wedge (s, U) \in Fail(P) \cup Fail(Q))\} \\ \cup \{(s, U) \mid s \in Div(P \Box Q)\}$$

$$Div(P \Box Q) = Div(P) \cup Div(Q)$$

Hiding Operator The hiding operator $\setminus$ defines a new process as the original P with all events from C hidden.

$$Fail(P \setminus C) = \\ \{(t \upharpoonright (\alpha P - C), U) \mid (t, U \cup C) \in Fail(P)\} \cup \{(t, U) \mid t \in Div(P \setminus C)\}$$

$$Div(P \setminus C) = \\ \{(s \upharpoonright (\alpha P - C)) \frown t \mid t \in (\alpha P - C)^* \wedge \\ (s \in Div(P) \vee (\forall\, n \bullet (\exists\, u \in C^* \bullet (\#u > n \wedge (s \frown u) \in Traces(P)))))\}$$

Interleaving Operator The interleaving operator $|||$ defines interleaved execution of P and Q with events common to both alphabets nondeterministically executed either by P or by Q.

$$\begin{aligned}
&Fail(P\,|||Q) = \\
&\qquad \{(s, U) \mid (\exists\, t, u \bullet s\ interleaves(t, u) \wedge (t, U) \in Fail(P) \wedge (u, U) \in Fail(Q))\} \cup \\
&\qquad \{(s, U) \mid s \in Div(P\,|||Q)\}
\end{aligned}$$

$$\begin{aligned}
&Div(P\,|||C) = \\
&\qquad \{u \mid (\exists\, s, t \bullet u\ interleaves(s, t) \wedge \\
&\qquad\quad (s \in Div(P) \wedge t \in Traces(Q) \vee s \in Traces(P) \wedge t \in Div(Q)))\}
\end{aligned}$$

Interrupt Operator: $(P\hat{}Q)$ is defined to be the process that acts like P, but may be interrupted at any time by the first event of Q. From this event on, Q continues to run and P is never resumed. We have

$$\alpha(P\hat{}Q) = \alpha P \cup \alpha Q$$

and

$$Traces(P\hat{}Q) = \{s \frown t \mid s \in TracesP \wedge t \in TracesQ\}$$

It is assumed that Q's initial events are not contained in αP.