Interference Analysis for Dependable Systems using Refinement and Abstraction

Claus Pahl

School of Computer Applications, Dublin City University
Dublin 9, Ireland
cpahl@compapp.dcu.ie

Abstract. A common requirement for modern distributed and reactive systems is a high dependability guaranteeing reliability and security. The rigorous analysis of dependable systems specifications is of paramount importance for the reliability and security of these systems. A two-layered modal specification notation will allow the specification of services and protocols for distributed dependable systems and their properties. Refinement and its dual – abstraction – will play the key roles in an integrated development and analysis framework. Refinement and abstraction form the basis for an interference analysis method for security properties and for automated test case generation.

1 Motivation

Current software engineering approaches are unlikely to deliver the level of dependability required to construct future distributed, decentralised, and reactive systems such as mobile systems, telecommunications management, communication and process control, or integrated e-business systems. The recent advent of Internet and other intercommunications technologies has made one aspect of properties particularly important: security properties.

We present a notation for the rigorous development and analysis of dependable systems properties. The specification of distributed systems is usually concerned with properties such as reliability or fairness of the communication. In dependable systems with high security requirements other properties are also important. Confidentiality describes that no confidential data is disclosed to unauthorised users. Integrity addresses unauthorised modification. Authentication describes that the identity of participants in a communication can be established. We have dependable systems such as public key infrastructures (PKI) in mind. PKIs are a combination of distributed systems and security technologies, which create an ideal setting to discuss reliability and security issues.

A PKI provides an infrastructure for the management of public keys in cryptographic systems [1]. It deals with entities, protocols and services in those systems. This includes for example services such as the generation, distribution and storage of keys and other secrets. The central concept is that of a certificate. A certificate is a datastructure that associates an identity to a public key by means

of a signature. This concept is used for encryption, signatures and key exchange. The objectives are to guarantee confidentiality, integrity and authentication.

We will analyse some aspects of PKIs, addressing services and protocols based on these services. We will analyse these services and protocols with respect to security issues such as confidentiality, integrity, and authentication. The security analysis is realised as an interference analysis, i.e., it is checked if an intruder can interfere with the system and violate any of the security conditions. The refinement calculus [2, 3] forms the framework for the analysis. The analysis is supported by systematic test case generation based on abstraction (abstraction is dual to refinement).

Modal logics, such as temporal or dynamic logics [4, 5], have shown their ability to define and reason about important properties of dependable systems, such as safety and liveness, through special modal operators [6, 7, 8]. Dynamic logic is suitable for the specification of finite aspects, which includes security considerations. Dynamic logic is compositional, i.e., reasoning via structural induction on commands is possible. We will argue that dynamic logic is a suitable tool for security aspects in reactive and distributed systems specification. In combination with a refinement concept it allows the analysis of dependable systems in a novel way. Another advantage of dynamic logic is that it embraces the classical pre/postcondition technique [9], which has become the foundation of various engineering methods and notations such as design-by-contract [10] or the Object Constraint Language OCL [11].

We propose refinement of modal specifications as the central concept for the analysis of dependable systems. The refinement relation can be used to develop systems starting from a simple core, but also to integrate an adversary into the specification in order to detect possible security flaws in a system specification. Refinement essentially guarantees property preservation. Assuming that a property P holds for some specification of a system S, i.e., $P(S)$, we expect a refined specification S' to preserve that property, i.e., $P(S) \Rightarrow P(S')$. Refinement is a classical software engineering technique [2, 3] developed to support transformational design and implementation, that has recently been deployed in defining essential concepts for component technology [12, 13] and also for interference analysis [14, 15]. Here, we will show a novel use of refinement as an analysis tool for detecting undesirable interferences and security violations. The refinement-based approach allows us to combine the traditional transformational development with the unusual applications of interference analyses and test case generation for the context of dependable systems.

We introduce our specification notation in Section 2. In subsections 2.3 and 2.4 we demonstrate the notation by specifying a protocol implementing an authentication service. The principles of refinement and abstraction are introduced in Section 3. An analysis looking at an authentication service is carried out in Section 4 for the protocol described in Section 2.3. Another form of analysis is addressed in Section 5 focussing on confidentiality and integrity in a key establishment and distribution service. We finish with related work and conclusions.

2 The Notation

The actors in communicating distributed systems are agents. Their activities are usually described in terms of the following application-specific basic commands: generate and remember data, establish and close connections, send and receive messages, and guards to protect the execution of operations[1]. In this section, we introduce the notation that we will use to specify and reason about dependability properties. A command language can be based on the constructs listed above. However, we will reduce this language for the sake of simplicity here.

2.1 The Command and Specification Languages

We define the command primitives – *send*, *receive* and a *test*-operator – and the command combinators informally, but we will give axiomatisations later on. This process of configuring the language contributes to a better understanding of the application and its problems. Flexibility in defining basic variations even on this level is important for the analysis of security protocols.

- $snd_{A\to R}(M_1,\ldots,M_n)$: the send operation for agent A. R is the receiver, the M_i denote messages that are sent. The M_i are local variables of the agent. Their value $s(M_i)$ in the current state s is sent to R. The operation fails if there is no variable M_i defined or no communication takes place.
- $rcv_{B\leftarrow S}(M_1,\ldots,M_n)$: the receive operation for agent B. S is sender and the M_i are messages arriving from S. The reception will only be carried out, if data has been sent. The message data is assigned to local variables M_i.
- $\phi?$: the test is an operator that involves a quantifier-free formula ϕ. The semantics is to proceed if ϕ is true, and fail otherwise.

We assume that messages are created and assigned to a variable before they are sent. Received messages are assigned to variables, too.

Command combinators are defined inductively. Let c_1, c_2 be command terms:

- $c_1; c_2$ (sequential composition): c_1 is followed by c_2,
- $c_1 + c_2$ (non-deterministic choice): one possibility is chosen and executed,
- c_1^* (iteration): c_1 is iterated a non-deterministically chosen finite number of times,
- $c_1|c_2$ (parallel composition): c_1 and c_2 are executed concurrently.

The parallel composition differs from the other command combinators in that it is an operator involving two agents composed in parallel, whereas the others can be combinations of commands of one or several agents. A send and a receive operation from two different agents can be synchronised. The two agents communicate by synchronised message passing. On the receiving side, data is assigned to a local name. The following is a parallel composition of two agents:

$$snd_{A\to B}(X); rcv_{A\leftarrow B}(Y) \mid rcv_{B\leftarrow A}(Z); snd_{B\to A}(f(Z))$$

[1] Later on, we will also consider cryptographic functionality.

An agent A sends a data item X to B and receives an answer Y from B. The second agent B applies a function f to the received data Z item before sending $f(Z)$ back to A. Agent A receives $f(Z)$ as Y.

The definitions of parallel composition and communication are critical for our analysis. Our semantics allows two agents to communicate, i.e. allows a send and a receive operation to be synchronised, if the types of the in- and out-parameters coincide. Other notations for the specification of communication such as the π-calculus [16] also use dedicated channels between two agents.

Our specification language consists of two sublanguages: a command language to express behaviour and a logical part to specify and reason about properties of command executions. The language is based on dynamic logic [4] – a logic with a notion of state that makes a command language explicit in the notation. Modalities are indexed by programs, which are built from primitive commands such as send and receive. Logical connectors such as conjunction, disjunction or negation are available. There are also mixed operators – the modal operators – combining commands and logical constructs, which make the language different from classical first-order logics. We introduce a box- and a diamond-operator for safety and liveness properties, respectively. Let c be a command.

- $[c]\phi$: whenever c terminates, it must do so in a state satisfying ϕ.
- $\langle c\rangle\phi$: it is possible to execute c and terminate in a state satisfying ϕ.

If c is a simple state transition, e.g. a receive operation, then $\phi \rightarrow [c]\ \psi$ and $\phi \rightarrow \langle c\rangle\ \psi$ are *contracts* for c with a precondition ϕ and a postcondition ψ[2]. If $c \equiv snd_{A\rightarrow B}(x)|rcv_{B\leftarrow A}(y)$ is an interaction, then $\phi_x \rightarrow [snd_{A\rightarrow B}(x)|rcv_{B\leftarrow A}(y)]\ \psi_y$ is a contract for the interaction saying that properties ϕ_x of an output variable x are transferred to ψ_y of an input variable y if the interaction takes place.

We can, for example, specify that an agent B remembers a message X that has been received from A, but an intruder I should not be able to access X,

$$Knows_A(X) \rightarrow [snd_{A\rightarrow B}(X) \mid rcv_{B\leftarrow A}(X)]\ Knows_B(X) \wedge \neg Knows_I(X)$$

using a predicate $Knows$. We might expect from a key exchange service – the parallel execution of agents A and B – that a shared key is eventually in place. $Knows(key)$ is an invariant for the sender of key.

$$\langle A|B\rangle\ Knows_A(key) \wedge Knows_B(key)$$

2.2 The Inference Framework

Formulas of our logical language are based on predicates. The equality predicate $t = t'$ is satisfied in a state of a semantic structure if the interpretations of terms t and t' are equal. Let M be a semantic structure and s be a state. A satisfaction

[2] The symbol $\rightarrow$ stands for implication.

$\models$ for the modalities can be defined as follows:

$$\begin{array}{rl} M, s \models [p]\phi & \text{iff every terminating computation of } p \text{ starting in } s \\ & \text{terminates in a state satisfying } \phi \\ M, s \models \langle p \rangle \phi & \text{iff exists a computation of } p \text{ starting in state } s \\ & \text{and terminating in a state that satisfies } \phi \end{array} \quad (1)$$

We could also define one of the constructors in terms of the other: $\langle p \rangle \phi := \neg[p]\neg\phi$ (or vice versa). With x' we denote the variable x in the previous state of a command execution. This allows us to specify for example increment operations $[incr(x)]\ x = x'+1$. In order to support the formal analysis of specified behaviour, the operations – such as send and receive – shall be axiomatised in terms of the given predicates, e.g., receive $rcv_{A \leftarrow B}$ for a given agent A:

$$[rcv_{A \leftarrow B}(X)]\ Knows_A(X) \quad (2)$$

After receiving the message X from B, the agent A remembers (knows about) X. An axiomatisation can be varied if the given application requires this. These axioms express a developer's assumptions about the environment explicitly.

We can axiomatise the commands combinators:

$$\langle c_1 + c_2 \rangle \phi \Leftrightarrow \langle c_1 \rangle \phi \vee \langle c_2 \rangle \phi \quad (3)$$

$$\langle c_1 ; c_2 \rangle \phi \Leftrightarrow \langle c_1 \rangle \langle c_2 \rangle \phi \quad (4)$$

There are also dual formulations for the box-operator: $[c_1 + c_2]\phi \Leftrightarrow [c_1]\phi \wedge [c_2]\phi$ and $[c_1 ; c_2]\phi \Leftrightarrow [c_1][c_2]\phi$. In the literature (e.g. [4]), we typically find axiomatisations of the test-operator such as $\langle \phi? \rangle \psi \Leftrightarrow \phi \wedge \psi$ and $[\phi?]\psi \Leftrightarrow (\phi \rightarrow \psi)$. Due to our non-standard definition of the test-operator, these equivalences do not hold here. There is no simple axiomatisation for the iteration t^*; see [4] for axioms.

The parallel composition $p|q$ of commands p and q of agents A and B, resp., makes our framework different from dynamic logic as presented in [4]. The semantics of $c_1|c_2$ is defined as a pair of component semantics $(\llbracket c_1 \rrbracket_A, \llbracket c_2 \rrbracket_B)$ if $\llbracket c_1 \rrbracket_A$ and $\llbracket c_2 \rrbracket_B$ are the semantics of c_1 and c_2. The following axioms hold:

$$[c_1|c_2]\phi \Leftrightarrow [c_1]\phi \wedge [c_2]\phi \quad (5)$$

$$\langle c_1|c_2 \rangle \phi \Leftrightarrow \langle c_1 \rangle \phi \vee \langle c_2 \rangle \phi \quad (6)$$

This corresponds to the axioms for the non-deterministic choice (3), except that a choice between two commands is interpreted in one structure, whereas the parallel composition is interpreted in two.

2.3 The Needham-Schroeder Protocol Specification

Public key infrastructures provide services such as key generation and distribution, or authentication. A number of these services are implemented by protocols. The Needham-Schroeder protocol is a possible way to implement key exchange

and authentication [17]. It allows to bring a shared secret in place, assuming that an encryption mechanism is already in place.

The Needham-Schroeder key exchange protocol is usually introduced using the following informal notation:

$$\begin{aligned} A &\rightarrow B : \{A, N_a\}_{K_b} \\ B &\rightarrow A : \{N_a, N_b\}_{K_a} \\ A &\rightarrow B : \{N_b\}_{K_b} \end{aligned}$$

Two agents, A and B, attempt to share a secret. A starts by sending its own identity and a randomly chosen number N_a (called a nonce – number used once) to B. A uses B's public encryption key K_b to encrypt the message. B decrypts the message with its own private decrypion key and sends the number sent by A, N_a, together with another nonce N_b (created by B itself) back to A – again using encryption, but now A's public key K_a. Since A has used B's public key, only B can decrypt the message. If A receives its nonce N_a, it can be sure that it has communicated with B. In order to allow B to also verify the authenticity of A, A sends the nonce N_b produced by B back to B. Two results should be achieved: authenticity of the participants and confidentiality of the nonces.

We reformulate the informal protocol specification using our command language, before specifying properties. The specification shall be divided into activities of agent A and agent B. Agent A acts as follows:

$$snd_{A \rightarrow B}(A, N_a); rcv_{A \leftarrow B}(N_a, N_b); snd_{A \rightarrow B}(N_b) \tag{7}$$

A sends a message to B, receives one from B, and sends a second message. The data items received are stored in variables. Here is B's behaviour:

$$rcv_{B \leftarrow A}(A, N_a); snd_{B \rightarrow A}(N_a, N_b); rcv_{B \leftarrow A}(N_b) \tag{8}$$

The authenticity of the agents A and B and the confidentiality the nonces is not guaranteed here. Encryption – to be added later – will achieve this.

2.4 Properties of the Needham-Schroeder Protocol

A dynamic logic specification consists of command terms and properties that specify the commands in their behaviour. We can classify security properties into authentication: authenticity of agents is guaranteed, confidentiality: secret data remains secret, and integrity: data remains intact. We use different forms of constraints to address them:

- Firstly, an *access control constraint* describes that a data item is accessible for the receiver, e.g., that data actually arrives if it is sent. This allows us to deal with confidentiality and integrity.
- An *authentication constraint* describes that after a sequence of message exchanges one agent is sure about the identity of another agent. This requires the use of cryptographic methods.

– The *correctness constraint* is a data-specific consistency condition, e.g., that data, which has been sent, satisfies a certain condition.

Later, we add cryptographic constraints. If a message has been encrypted with a public key, then the message can only be decrypted with the corresponding private key. Authentication and confidentiality can be achieved using cryptographic methods, but, still, an intruder attack can violate all these properties.

The receive-operation of agent A in the Needham-Schroeder protocol shall be specified by the following *access control constraint*:

$$[rcv_{A \leftarrow B}(N_a, N_b)]\ Knows_A(N_a, N_b) \tag{9}$$

After receiving data, A remembers the information, i.e., stores it locally in variables N_a and N_b, expressed using the predicate $Knows$. Agent B receives two messages. Each reception is remembered in the corresponding variables:

$$[rcv_{B \leftarrow A}(A, N_a)]\ Knows_B(A, N_a) \tag{10}$$

$$[rcv_{B \leftarrow A}(N_b)]\ Knows_B(N_b) \tag{11}$$

Previous assignments are overwritten, otherwise older assignments are remembered. After discussing single protocol steps, we address the full behaviour of a single agent. After finishing their execution sequences both agents shall share a secret, or at least the same two values[3]. Here are agent A (12) and B (13):

$$[snd_{A \rightarrow B}(A, N_a); rcv_{A \leftarrow B}(N_a, N_b); snd_{A \rightarrow B}(N_b)]\ Knows_A(N_a, N_b) \tag{12}$$

$$[rcv_{B \leftarrow A}(A, N_a); snd_{B \rightarrow A}(N_a, N_b); rcv_{B \leftarrow A}(N_b)]\ Knows_B(N_a, N_b) \tag{13}$$

An agent A *authenticates* another agent B, if A sends a random number N_a to B encrypted with B's public key. If A receives N'_a back from B and $N_a = N'_a$, then A can be sure that only B – the owner of the public key K_B – could have decrypted $K_B(N_a)$ and sent N_a back. We expect

$$[snd_{A \rightarrow B}(K_B(N_A)); rcv_{A \leftarrow B}(K_A(N'_A))]\ N_A = N'_A \tag{14}$$

for the authentication. We define the authentication predicate $Auth_A(B)$ for agent A and a target agent B to become true in that case.

Data sent or received is subjected to constraints in some cases. The first send-operation of agent A is not restricted with respect to a *correctness constraint*,

$$[snd_{A \rightarrow B}(A, N_a)]\ true \tag{15}$$

but A should receive its nonce N_a back from B, i.e., the received value in N_a is the same as the value in the previous state N'_a:

$$[rcv_{A \leftarrow B}(N_a, N_b)]\ N_a = N'_a \tag{16}$$

[3] The fact that B also remembers the identity of A – see (10) – is not relevant at this stage and therefore neglected.

A should only proceed if this is satisfied – expressed using a precondition:

$$N_a = N'_a \rightarrow [snd_{A \rightarrow B}(N_b)]\ true \tag{17}$$

The first value that A receives must coincide with its own nonce N_a. Similar constraints can be imposed on agent B, e.g., on the second receive operation: $[rcv_{B \leftarrow A}(N_b)]\ N_b = N'_b$.

Access control, authentication, and correctness form different views on the problem – e.g., the accessibility formula $[rcv_{A \leftarrow B}(N_a, N_b)]\ Knows_A(N_a, N_b)$ and the correctness condition $rcv_{A \leftarrow B}(N_a, N_b)]\ N_a = N'_a$ can be combined to the formula $[rcv_{A \leftarrow B}(N_a, N_b)]\ Knows_A(N_a, N_b) \wedge N_a = N'_a$ using inference rules of the logic, see Section 2.2.

3 Refinement and Abstraction

The two main concepts for our interference analysis shall now be introduced. Traditionally, refinement is used to develop a specification step by step. Refinement also serves another purpose in our approach. Security analysis – intruder integration and interference analysis – can also be supported. We use the concept of abstraction – the dual of refinement – to test for interferences. We will briefly show how to add encryption to a simplified protocol specification in order to illustrate the transformational development approach based on refinement. The concepts for interference analysis and testing will be applied in Section 4.

3.1 Refinement

The refinement relation is essentially defined based on implication. A specification is a refinement of another if it implies it, i.e., if the refinement preserves the properties of the original specification. Let ϕ and ψ be formulas: ϕ refines ψ iff $\phi \rightarrow \psi$. For commands p and q specified by $\phi \rightarrow [c]\ \phi'$ and $\psi \rightarrow [c']\ \psi'$ we define – based on the monotonicity of [.], see [4] Th. 4(2) – for the box-operator:

$$c \text{ is refined by } c' \text{ , or } c \sqsubseteq c' \text{ , iff } \phi \rightarrow \psi \wedge \psi' \rightarrow \phi' \tag{18}$$

We have chosen to define a sufficient and necessary condition. An intruder cannot be introduced (using refinement – see Section 4) that violates the refinement, but does not affect the security conditions of the original specification. A violation of a refinement should only occur if security specifications are violated.

We do not constrain the commands c and c' in any way. This allows a single command to be refined by a sequence of commands. We could also insert commands into a sequence of commands without violating the refinement condition. The refinement of commands is here defined on properties of the state that is reached through command execution. More support for a refinement calculus can be based on an inference system for dynamic logic, see [4].

To add encryption to the protocol specification from Section 2.3 using refinement, we define two new functions for encryption and decryption and axiomatise their behaviour. The particular encryption method (RSA, Merkle-Hellman, etc. [18]) shall not matter. We assume that the encryption scheme is secure, i.e., that there are no principal problems such as mathematical flaws. We assume a public key encryption scheme. K_A is A's (public) encryption key and K_A^{-1} is its (private) decryption key (analogously for B). $K_A(X)$ is the encryption operation and $K_A^{-1}(Y)$ the decryption operation[4]. The cryptographic law is:

$$K_A^{-1}(K_A(X)) \;=\; X \tag{19}$$

In order to fully specify cryptographic basics, we would need to express that not only can the original message be recovered with the corresponding private key, but also that no other key except the corresponding private key can decrypt the message. For the sake of simplicity, we have left out properties like this.

Each agent shall know the public keys of the agents it wants to communicate with securely. In our case, for two agents A and B, the predicates $Knows_A(K_B)$ for A and $Knows_B(K_A)$ for B are true. The specification of agent B, who receives encrypted data from A, now looks as follows:

$$[rcv_{A\leftarrow B}(X,Y)]\; Knows_B(K_B^{-1}(X,Y)) \tag{20}$$

B tries to decrypt the received pair of two data items X and Y. B should only proceed if the decryption is successful, i.e., results in $K_B^{-1}(X,Y) = (A,N_a)$.

$$[(K_B^{-1}(X,Y) = (A,N_a))?; snd_{B\to A}(K_A(N_a,N_b))]\; true \tag{21}$$

Composed in parallel, A and B can communicate – the send and receive operation are synchronised and data is transferred from A to B. After applying the axioms (5) and $true \wedge \phi \Leftrightarrow \phi$ to (15) and (10), the simplified specification

$$[snd_{A\to B}(A,N_a) \mid rcv_{B\leftarrow A}(A,N_a)]\; Knows_B(A,N_a) \tag{22}$$

can be refined by

$$[snd_{A\to B}(K_B(A,N_a)) \mid rcv_{B\leftarrow A}(K_B(A,N_a))]\; Knows_B(K_B^{-1}(K_B(A,N_a))) \tag{23}$$

With the assumption that a cryptosystem is in place, we get $Knows_B(A,N_a) = Knows_B(K_B^{-1}(K_B(A,N_a)))$ by applying the cryptographic law (19). Thus, the refinement relation is satisfied. We have proved that properties from the original specification (22) are actually preserved by (23).

3.2 Abstraction and Testing

The parallel composition of agents is the essential combinator for our interference analysis. We can automate the analysis by testing the composition of sequential

[4] This notation is not sufficient for cryptographic techniques such as signatures. To keep the notation simple for the given protocol form, we have used this simple form.

agent behaviours. Each sequential agent behaviour – called a scenario – is a test case for a non-sequential, non-deterministic agent specification. Agents of our ideal protocol have been defined in a sequential deterministic way, but we assume a non-deterministic behaviour for the intruder. These scenarios shall be constrained by the abstraction $Spec \sqsupseteq Scen$, i.e., the system specification $Spec$ is abstracted by the scenario $Scen$, or, the specification refines the scenario. We will systematically try to find intruder scenarios that – in composition with the protocol agents – violate the refinement relation.

The basic principle of test case generation in the context of the refinement calculus is that test cases abstract contracts[5]. The *abstraction* is dual to the refinement relation. c abstracts c' – or c' is abstracted by c – if c' refines c:

$$c' \sqsupseteq c \quad := \quad c \sqsubseteq c' \wedge c \neq false$$

for any two commands c and c'. $false$ is the trivial abstraction, which should be excluded. Specifications can involve sequential, iterative and non-deterministic behaviour. For an intruder, we cannot assume sequential or deterministic behaviour, but we will test the system using various sequential intruder scenarios.

The first step shall be to define a simple input/output test case for a command. A *test case* TC_c for a command c is defined by:

$$TC_c(\alpha, \beta) \;:=\; \alpha \rightarrow [c]\ \beta$$

α and β are conditions describing input and output values. The following proposition states when a pair of conditions α and β is a suitable test case for a command, i.e., when it abstracts a command c specified by $\phi \rightarrow [c]\ \psi$.

$$c \sqsupseteq TC_c(\alpha, \beta) \quad \Leftrightarrow \quad (\alpha \rightarrow \phi) \;\wedge\; (\alpha \wedge \psi) \rightarrow \beta \tag{24}$$

An example shall illustrate this proposition. We assume the following definitions for the conditions α, β, ϕ and ψ: $\phi \equiv x \geq 0$, $\psi \equiv y = x + 1$, $\alpha \equiv x = 1$, and $\beta \equiv y = 2$. Then, the two constraints formulated in the proposition are satisfied and we have a proper test case: the condition $\alpha \rightarrow \phi$ is satisfied since $x = 1 \rightarrow x \geq 0$, and the condition $(\alpha \wedge \psi) \rightarrow \beta$ is satisfied since $(x = 1 \wedge y = x + 1) \rightarrow y = 2$.

In order to deal with interaction between agents of a protocol, we expand our notion of test cases to interactions between two agents.

$$TC_{c_1|c_2}(\alpha_x, \beta_y) := \alpha_x \rightarrow [\overline{c_1}\langle x\rangle | c_2(y)]\ \beta_y$$

where α_x and β_y are properties of x and y, respectively. Properties of x are transferred to y if the interaction takes place. A test case for parallel compositions requires x and y to have the same type.

$$c_1|c_2 \sqsupseteq TC_{c_1|c_2}(\alpha_x, \beta_y) \quad \Leftrightarrow \quad \alpha_x \rightarrow \beta_y \tag{25}$$

[5] Most of the concepts here are motivated by [19], but formulated in a different semantical framework and extended to parallel composition.

The key construct to test concurrent non-deterministic agents is a scenario, i.e., a sequence of basic commands or interactions of basic commands. We define a *scenario* S for a specification as a sequence $(c_1; \ldots; c_n)$ of basic commands or interactions of basic commands. We assume an iterative non-deterministic choice to be the basic format of an intruder specification, see also (31). Scenarios abstract iterative choices $(c_1 + \ldots + c_n)^*$ if the scenario itself is executable, i.e., if the last state of the sequence can be reached.

Let c_i be specified by $\phi_i \rightarrow [c_i]\ \psi_i$ and assume $\psi_{i_k} \neq false$ (the last state should be reachable). Then

$$(c_1 + \ldots + c_n)^* \sqsupseteq (c_{i_1}; \ldots; c_{i_k}) \ \text{ if } \ \psi_{i_j} \rightarrow \phi_{i_{j+1}} \tag{26}$$

with $1 \leq i_j \leq n$ $(j = 1, .., k)$ and $\phi_{i_j} \rightarrow [c_{i_j}]\ \psi_{i_j}$ and $\phi_{i_{j+1}}$ being the precondition of the $(j+1)$-th element in the scenario sequence.

The next two propositions are corollaries based on the last proposition. They essentially state how to construct scenarios for specifications. The first one shows conditions that makes a basic scenario a test case for an iterative choice. Let $p \wedge \phi_i \rightarrow [c_i]\ \psi_i$ and $\phi_j \rightarrow [c_j]\ \psi_j$. If $p \neq false$ and $\psi_i \rightarrow \phi_j$ then

$$(c_1 + \ldots + c_n)^* \sqsupseteq (c_i; c_j) \tag{27}$$

for $i, j \in 1, .., n$. The next corollary shows how to combine basic scenarios into more complex ones. Let $c_a \wedge \phi_a \rightarrow [c_a]\ \psi_a$, $a \in \{i, j, k\}$. If $(c_1 + \ldots + c_n)^* \sqsupseteq c_i; c_j$ and $(c_1 + \ldots + c_n)^* \sqsupseteq c_j; c_k$ and $c_i \neq false$ and $\psi_i \rightarrow c_i \wedge \phi_j$ and $c_i \neq false$ and $\psi_i \rightarrow c_j \wedge \phi_j$ then

$$(c_1 + \ldots + c_n)^* \sqsupseteq (c_i; c_j; c_k) \tag{28}$$

4 Authentication Analysis

A central PKI service is authentication support through certificates. Therefore, our first analysis addresses a protocol implementing an authentication service.

Security analysis is mostly concerned with safety properties, i.e., something (bad) must never happen (e.g., that the intruder knows a secret – at any time), whereas the development of protocols is more involved with liveness properties, i.e., that something (good) will happen eventually (data arrives, keys are eventually in place). Our dynamic logic provides constructs for both aspects.

We will base our analysis on an accepted and successful methodology – used by most analysis techniques [8, 20, 21]: formal specification of the ideal behaviour, add the intruder or possible interfering features, state the properties to be guaranteed/analysed, analyse the ideal specification, and vary parameters and analyse again. This justifies to use refinement to add the adversary or new features, but also to vary parameters through repeated use of refinement.

4.1 The Protocol

The full specification of the desired behaviour of the Needham-Schroeder protocol with respect to *authentication* based on the specifications of A and B in

isolation, (12) and (13), is:

$$[snd_{A\to B}(K_B(A,N_a)); rcv_{A\leftarrow B}(K_A(N_a,N_b)); snd_{A\to B}(K_B(N_b)) \mid rcv_{B\leftarrow A}(K_B(A,N_a)); snd_{B\to A}(K_A(N_a,N_b)); rcv_{B\leftarrow A}(K_B(N_b))] \; Auth_A(B) \wedge Auth_B(A) \tag{29}$$

This is the *ideal* protocol. Any intruder behaviour will be analysed against this specification. The agents themselves behave deterministically, thus we have used the box operator. When the intruder I is integrated, behaviour becomes non-deterministic. The intruder might intercept at any time. Still, we would like to guarantee that A and B eventually authenticate each other, even under interference by an intruder:

$$\langle A|B|I\rangle \; Auth_A(B) \wedge Auth_B(A) \tag{30}$$

We want to prevent that an intruder can interfere with the authentication between A and B. A mutual authentication between A and B shall be achieved. This is an adaptation of Goguen and Meseguer's classical non-interference definition. Here, an intruder does not interfere with another group of agents, if the excution of intruder commands has no effect on the agent's security properties.

4.2 The Adversary

We assume intruders to have capabilities as formulated in the Dolev-Yao model [22]. The Dolev-Yao model is an accepted collection of assumptions about possible intruder behaviour. The intruder can read any message, block further transmission, decompose messages, remember messages, generate fresh data, and compose and send new messages[6]. In principle, the intruder can non-deterministically choose between these operations. The general difficulty with these analyses is to make the right assumptions about the intruder (or about new features in feature interaction analysis) in order to detect possible interferences. This problem can only be solved by the developer or analyser, but the specification and analysis technique should provide the possibility to vary assumptions explicitly.

The mechanism for an intruder to attack a protocol is to intercept the communication between the agents participating in the protocol. This can be modelled by allowing the intruder to be executed in parallel with the agents. Then, the intruder I can communicate with the agents A and B, receiving and sending messages. Let $A := snd_{A\to B}(y)$, $B := rcv_{B\leftarrow A}(y)$ and $I := rcv_{I\leftarrow A}(y); snd_{I\to B}(f(y))$. The parallel composition of A, B and intruder I, $A|B|I$, can result in one of the following executions based on synchronisations of non-deterministically chosen send- and receive-operations. A and B can communicate directly by transferring data from A to B, or A communicates first with I and then I communicates with B sending manipulated data $f(y)$ to B. The first is the desired case, the second is a successful intrusion, or interference, using a man-in-the-middle attack.

[6] This does not include the intruder's encryption capabilities.

Reducing the capabilities of the intruder to two operations here, the general behaviour of an intruder is:

$$(snd_{I \to X}(M_1, \ldots, M_n) + rcv_{I \leftarrow Y}(M_1, \ldots, M_m))^* \tag{31}$$

The intruder chooses repeatedly and non-deterministically between sending and receiving – we cannot make many assumptions about an intruder's behaviour. We assume that the intruder does not block communication between A and B.

4.3 The Analysis

In a concrete example, the intruder may execute the following command sequence

$$\begin{aligned} &rcv_{I \leftarrow A}(K_I(A, N_a)); snd_{I \to B}(K_B(A, N_a)); \\ &\quad rcv_{I \leftarrow B}(K_A(N_a, N_b)); snd_{I \to A}(K_A(N_a, N_b)) \end{aligned} \tag{32}$$

which satisfies (31). The intruder intercepts the communication between A and B. At the beginning, the intruder has to convince A to communicate with him instead of B, i.e., A needs to send data to him and to use his public encryption key. The intruder then forwards this to B imposturing as A, and B's answer to A is again intercepted and forwarded to A. A and B might not suspect an intrusion. In this scenario, A authenticates I, since A uses I's public key and receives N_A back. B authenticates A since N_B encrypted with A's public key is returned. This is where the protocol fails to work securelyfootnoteThis attack has originally been described in [23]..

The intruder can be integrated via refinement. The ideal protocol specification, which specifies the expected secure behaviour, should be preserved. The inclusion of a successful intruder would violate the ideal specification, i.e., would not refine the ideal protocol specification. For instance, the confidentiality constraint could be violated by the intruder behaviour. Refinement is the tool to analyse, i.e., to prove or disprove, the security of a protocol. We would hope to prove that all possible extensions by intruder behaviours are refinements that preserve the properties specified for the protocol. Then, the protocol is secure.

The essential properties have already been discussed. Eventually the agents A and B have authenticated each other: $Auth_A(B) \wedge Auth_B(A)$. Including the intruder I as specified above in formula (32) will satisfy

$$\langle A|B|I\rangle \; Auth_A(I) \wedge Auth_B(A) \tag{33}$$

since the intruder intercepts the communication and is able to imposture as A for B, but this clearly violates the protocol specification (30) – $\langle A|B|I\rangle \; Auth_A(B) \wedge Auth_B(A)$ – which requires that A and B mutually authenticate each other. Seen as a refinement step, we get a violation of the constraint: the predicate $Auth_A(I)$ does obviously not imply $Auth_A(B)$. Besides being used for the stepwise development, refinement is also a tool for analysis – even though our aim now is to violate the refinement constraint in order to detect security flaws.

4.4 Testing

The testing concepts shall now be applied to the authentication analysis of the Needham-Schroeder protocol. Firstly, we would need to summarise the contracts of the basic commands such as $[snd_{A\to B}(K_B(N_A)); rcv_{A\leftarrow B}(K_A(N_A))]\ Auth_A(B)$ or $[snd_{A\to B}(A, N_a)]\ true$. Then, we list the possible interactions between the two agents of the ideal protocol in (34) and also some of the possible interactions between the two agents and the intruder for the extended protocol in (35).

$$\begin{array}{l} \iota_1 := \ snd_{A\to B}(A, N_a) \mid rcv_{B\leftarrow A}(A, N_a) \\ \iota_2 := rcv_{A\leftarrow B}(N_a, N_b) \mid snd_{B\to A}(N_a, N_b) \\ \iota_3 := \quad snd_{A\to B}(N_b) \mid rcv_{B\leftarrow A}(N_b) \end{array} \tag{34}$$

A variety of interactions is possible if the non-deterministic intruder is included. A few of them are:

$$\begin{array}{l} \iota_4 := snd_{A\to B}(A, N_a) \mid rcv_{I\leftarrow A}(A, N_a) \\ \iota_5 := \ snd_{I\to B}(I, N_a) \mid rcv_{B\leftarrow I}(I, N_a) \\ \iota_6 := rcv_{I\leftarrow B}(N_a, N_b) \mid snd_{B\to I}(N_a, N_b) \\ \iota_7 := rcv_{A\leftarrow I}(N_a, N_b) \mid snd_{I\to A}(N_a, N_b) \\ \iota_8 := \quad snd_{A\to B}(N_b) \mid rcv_{I\leftarrow A}(N_b) \\ \iota_9 := \quad snd_{I\to B}(N_b) \mid rcv_{B\leftarrow I}(N_b) \end{array} \tag{35}$$

The interactions $\iota_4, \ldots, \iota_9$ describe a successful intrusion that leads to the described authentication problem – see (32) in Section 4.3. Interactions are the basis of the scenario generation. Interactions are sequentially composed to simple scenarios in the first step. Simple two-element scenarios are $(\iota_4; \iota_5)$, $(\iota_5; \iota_6)$, $\ldots$, $(\iota_8; \iota_9)$. These simple scenarios are derived using proposition (27). The simple scenarios are combined to full scenarios based on proposition (28), e.g.,

$$\begin{array}{l} \sigma(\iota_4) := \iota_4 \\ \sigma(\iota_5) := \iota_4; \iota_5 \\ \sigma(\iota_6) := \iota_4; \iota_5; \iota_6 \end{array} \tag{36}$$

Finally, we can show that the scenario $\sigma(\iota_9) := \iota_4; \ldots; \iota_9$ is an abstraction of the protocol including the intruder.

$$A|B|I \ \sqsupseteq \ \iota_4; \ldots; \iota_9 \tag{37}$$

We can derive the intruder behaviour for this scenario by projecting onto the intruder in the overall sequence of interactions. This sequence is clearly an abstraction of the general intruder behaviour:

$$\begin{array}{l} (snd_{I\to X}(M_1) + rcv_{I\leftarrow Y}(M_2))^* \ \sqsupseteq \\ \qquad rcv_{I\leftarrow A}(A, N_a); snd_{I\to B}(I, N_a); rcv_{I\leftarrow B}(N_a, N_b); \\ \qquad snd_{I\to A}(N_a, N_b); rcv_{I\leftarrow A}(N_b); snd_{I\to B}(N_b) \end{array} \tag{38}$$

The inclusion of the intruder does not satisfy the constraint that A and B mutually authenticate each other, see (30). A security flaw is detected.

5 Confidentiality and Integrity Analysis

Besides authentication support, a PKI also provides services concerned with the secure distribution of keys and other secrets. In order to show the versatility of our method, we shall look at confidentiality and integrity issues relating to a key establishment service based on a simplified Diffie-Hellman protocol [18].

5.1 The Protocol

The protocol assumes a common number g. The names a and b denote random values generated by A and B, respectively.

$$snd_{A\to B}(g^a); rcv_{A\leftarrow B}(g^b) \tag{39}$$

A sends g^a to B, and receives g^b from B. Here is B:

$$rcv_{B\leftarrow A}(g^a); snd_{B\to A}(g^b) \tag{40}$$

Here are the access control properties concerning A and B:

$$\begin{array}{l}[snd_{A\to B}(g^a); rcv_{A\leftarrow B}(g^b)]\ Knows_A(a,b)\\ [rcv_{B\leftarrow A}(g^a); snd_{B\to A}(g^b)]\ Knows_B(a,b)\end{array} \tag{41}$$

Thus, we get for the ideal protocol – the parallel composition:

$$\begin{array}{c}[snd_{A\to B}(g^a); rcv_{A\leftarrow B}(g^b)\ |\ rcv_{B\leftarrow A}(g^a); snd_{B\to A}(g^b)]\\ Knows_A(a,b) \wedge Knows_B(a,b)\end{array} \tag{42}$$

Two values are exchanged. A knows value b of B, and B knows value a of A. This is the full specification of the desired behaviour – the protocol without intruder – based on the specifications of A and B in isolation, (41). Including intruder I

$$[A|B|I]\ \neg Knows_I(a,b) \tag{43}$$

we would not like I to interfere, but would like to achieve

$$\langle A|B|I\rangle\ Knows_A(a,b) \wedge Knows_B(a,b)\ . \tag{44}$$

We want to prevent that an intruder will ever get hold on the secret (a safety condition) and that A and B will eventually share a secret (a liveness condition).

5.2 The Adversary

We reduce the capabilities of the intruder to the send- and receive-operations: $(snd_{I\to X}(M_1,\ldots,M_n) + rcv_{I\leftarrow Y}(M_1,\ldots,M_m))^*$. The intruder may proceed as specified by the following command sequence:

$$rcv_{I\leftarrow A}(g^a); snd_{I\to B}(g^a); rcv_{I\leftarrow B}(g^b); snd_{I\to A}(g^b) \tag{45}$$

The intruder intercepts the communication between A and B. He will know about the secret shared between A and B – the values g^a and g^b – and he will even about a and b if he knows g or can infer it.

5.3 The Analysis

The intruder is again integrated via refinement. A successful intruder inclusion would violate the security conditions of the ideal specification, i.e., would not refine the ideal protocol specification. We hope that eventually secrets a and b are in place and nobody else knows about them, i.e., $Knows_A(a,b) \wedge Knows_B(a,b)$ and $\neg Knows_I(a,b)$ for any other agent I. Including the intruder I as specified above in formula (45) and assuming that I can infer g will satisfy

$$[A|B|I]\ Knows_I(a,b) \tag{46}$$

since the intruder intercepts the communication and has access to g^a and g^b and can calculate a and b from them, but it also violates the specification $[A|B|I]\ \neg Knows_I(a,b)$, see (43). Assuming that I knows g, we get again a violation of the refinement constraint: $Knows_I(a,b)$ does not imply $\neg Knows_I(a,b)$.

6 Related Work

Most approaches to security systems analysis are essentially adaptations of general frameworks to the security context. Durgin and Mitchell [8] use conventional logics and analysis methods to analyse security protocols. Their specification approach is based on multisets of first-order formulas (called facts). A rewriting technique is used to develop and analyse specifications. State are described by multisets of facts. State transitions are given by rules, essentially relations on multisets of facts. This treatment of states and state transitions is the essential difference between their approach and our framework. We believe that a formal specification framework closer to techniques such as pre/postconditions and refinement is more suitable for a general approach to dependable systems engineering. Common characteristics include the aim to reduce implicit assumptions and to make them explicit, and the use of the explicit intruder method.

The spi-calculus [20] is based on the π-calculus and includes additional cryptographic primitives. Process calculi such as the π-calculus are suitable to model and develop infrastructures for distributed and mobile systems. The key difference to our approach is that the intruder behaviour is not modelled explicitly in the spi-calculus. Security properties of process definitions such as confidentiality (secrecy) and authentication (essentially integrity) are expressed via equivalences to a process specification. Consider the following example. Two processes shall be defined: a process A sending a message M on channel c_{AB} and a process $B := c_{AB}(x).F(x)$ receiving on c_{AB} and then processing the input x. The protocol is the parallel composition of A and B, i.e., $P = \nu c_{AB}.A(M)|B$ with a channel c_{AB} restricted to A and B. We expect B to process M internally, which can be expressed by $B_{spec} := c_{AB}(x).F(M)$. The overall protocol specification is $P_{spec} := \nu c_{AB}.A(M)|B_{spec}$.

- Secrecy: if $F(M) \simeq F(M')$ then $P(M) \simeq P(M')$ for all M, M'. Whatever the message is, an observer cannot distinguish between messages. If B does not leak M within F, then the protocol should not leak M in order to guarantee secrecy.

– Authentication (integrity): $P(M) \simeq P_{spec}(M)$ for all M. The protocol P should behave (under observation) like its specification P_{spec}, i.e., if M is sent, then M arrives unchanged and is processed subsequently.

The equivalence $\simeq$ is testing equivalence. It formalises the idea of observation (by an intruder). Compared to our approach, the spi-calculus is more abstract. It assumes restricted channels to be secure. Our approach does not make this assumption. We offer the possibility of more fine-granular and explicit analyses.

Paulson's Inductive Method [21] uses induction over protocol traces (a trace is a list of events that occur in some run of a protocol). Paulson introduces a specialised notation for security protocols. Standard operators to construct, deconstruct and remember messages are used in the specification of traces. The overall set of traces describing a protocol is defined inductively. Focardi, Ghelli and Gorrieri [24] apply a non-interference approach for the analysis of the Needham-Schroeder protocol. To keep our approach suitable for all forms of dependability aspects and integrated development and analysis, we have included security-specific aspects into a general-purpose framework, providing flexibility and configurability by combining different command features.

Butler [25] describes an approach to security systems analysis similar to ours. Butler bases his framework on a combination of the abstract machine notation AMN (the B method) and CSP. He also uses refinement to introduce the intruder. The correctness of an abstraction invariant AI needs to be checked: $S \sqsubseteq_{AI} T$ if $AI \Rightarrow [T]\langle S\rangle AI$. Butler's approach is based on iterative refinement, i.e., an initial abstraction might need to be strengthened iteratively until suitable. Butler's and our approach are similar in that both use an explicit intruder model and use refinement to introduce encryption and the intruder. Both provide safety and liveness operators and make parallel composition available. The key difference is that Butler's approach is based on data refinement with explicit state, whereas we use an implicit, observation-based notion of state, which creates a more abstract framework. We think that developing our refinements results into refinement laws giving templates for e.g. confidentiality-preserving refinements is a more suitable way. The combination of CSP and FDR, a model checking tool, has been been very fruitful for security analyses, e.g., to detect Lowe's attack [23]. This work has been carried further; [26] is a recent example. However, a proof-theoretic approach can give more insight into why a protocol works or fails than model checking.

We see refinement as an interference analysis tool, not restricted to security analysis, but also suitable for other forms of interference detection. Feature interaction in telephony systems poses a similar problem [14, 15]. If a new feature has to be added to an existing system, the main question is whether there are unexpected or undesirable interferences with existing features. Refinement can answer this question. The principle of our analysis method – state the ideal properties, add new behaviour, and analyse possible interferences – is not limited to security analysis. In [15], an investigation into common simple telephone systems and advanced features such as call waiting and call forwarding is carried out. Certain properties (invariants) are proven for the specification of the basic

system. A refined specification including advanced features needs to preserve the properties. There is an interference, if this is not possible. Feature interaction is defined as the violation of proof obligations in a refinement.

Our work is based on testing approaches developed in [27, 28, 29, 19]. We have in particular based parts of our test case generation on ideas developed by Aichernig in [19]. He presented his work in a general purpose context, with semantics based on weakest preconditions – essentially based on [3]. We have improved this semantic framework towards a more flexible and expressive modal logic framework. Additionally, we have provided an improved, process-algebra style command language including an explicit parallel composition.

7 Conclusions

Our approach to integrated development and analysis of dependable systems is based on a refinement mechanism for both purposes. Using refinement as an analysis tool is not restricted to security analyses where various intruder behaviours can be analysed. The analysis of any kind of interference such as feature interaction can be carried out. An essential technique for the analysis of interferences is to vary the behaviour. Elements have to be added or removed in a flexible way. We have provided two ways to control this flexibility: firstly, by using refinement to add new elements while preserving properties, and, secondly, by providing a framework where the command language for the communication primitives itself is not fixed, but can be influenced through the introduction and axiomatisation of new commands (or variants of existing ones). This flexibility in reflecting different assumptions about the underlying technology and the intruder is crucial. Another key element is the compositionality of the approach, which supports the required flexibility in modelling and analysing various scenarios through composition and decomposition.

Our main objective has been to illustrate the concepts needed to address reliability and security problems in dependable systems engineering. Two different aspects have been looked at to show the versatility of the approach. Here, we have illustrated concepts using aspects from the well-known security protocols Needham-Schroeder and Diffie-Hellman. In [30], we have investigated a specialised protocol – the Online Certificate Status Protocol OCSP. Due to the compositionality of the dynamic logic framework, the approach is scalable and can be applied to larger systems. We have addressed mechanised analysis support through test case generation, necessary for large systems analysis. An approach to further simplify reasoning, and enable automated or mechanised reasoning in particular, is to reduce the complexity to equational reasoning by defining a refinement between modal formulas where the condition can be reduced to an implication between simple non-modal formulas. Suitable environments for proof support could be tools supporting pre- and postcondition based specification or tools such as tools for the B specification language, which have also been used in [25] and in the feature interaction analysis [15] discussed earlier on.

We have used abstraction-based testing to verify security properties. Since we have used a dynamic logic similar to the modal μ-calculus [31], the question arises whether model checking is another alternative. The modal μ-calculus is a branching time temporal logic that forms the basis of several model checking approaches, see [32]. With a finite state space and finite set of properties, model checking becomes an alternative. If a given model satisfies the ideal protocol specification, then the model also has to satisfy the protocol with intruder. Otherwise, there is a security violation. In [30], we have given semantics to a similar specification notation based on Kripke transition systems and the μ-calculus, enabling model checking as an alternative approach to the automation of the security analysis.

Acknowledgements

The author would like to thank the anonymous reviewers for their valuable comments.

References

[1] IETF PKIX Working Group. Internet X.509 Public Key Infrastructure, 2000. http://www.ietf.org/internet-drafts/draft-ietf-pkix-roadmap-06.txt.

[2] C. Morgan. *Programming from Specifications 2e.* Addison-Wesley, 1994.

[3] R.J.R. Back and J. von Wright. *The Refinement Calculus: A Systematic Introduction.* Springer-Verlag, 1998.

[4] Dexter Kozen and Jerzy Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 789–840. Elsevier Science Publishers, 1990.

[5] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 995–1072. Elsevier Science Publishers, 1990.

[6] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[7] K.M. Chandy and J. Misra. *Parallel Program Design.* Addison-Wesley, 1988.

[8] N.A. Durgin and J.C. Mitchell. Analysis of Security Protocols. In M. Broy and R. Steinbruggen, editors, *Calculational System Design*, pages 369–395. IOS Press, 1999.

[9] G.T. Leavens and A.L. Baker. Enhancing the Pre- and Postcondition Technique for More Expressive Specifications. In R. France and B. Rumpe, editors, *Proceedings 2nd Int. Conference UML'99 - The Unified Modeling Language.* Springer Verlag, LNCS 1723, 1999.

[10] Bertrand Meyer. Applying Design by Contract. *Computer*, pages 40–51, October 1992.

[11] J.B. Warmer and A.G. Kleppe. *The Object Constraint Language – Precise Modeling With UML.* Addison-Wesley, 1998.

[12] M. Büchi and E. Sekerinski. Formal Methods for Component Software: The Refinement Calculus Perspective. In *Proceedings 2nd International Workshop on Component-Oriented Programming WCOP '97.* Turku Center for Computer Science, General Publication No.5-97, Turku University, Finland, 1997.

[13] C. Pahl. Components, Contracts and Connectors for the Unified Modelling Language. In *Proc. Symposium Formal Methods Europe 2001, Berlin, Germany.* Springer-Verlag, LNCS-Series, 2001.
[14] B. Mermet and D. Méry. Incremental Specification of Telecommunication Services. In M. Hinchey, editor, *International Conference on Formal Engineering Methods ICFEM.* IEEE Press, 1997.
[15] J.-P. Gibson, G. Hamilton, and D. Méry. Integration Problems in Telephone Feature Requirements. In A. Galloway and K. Taguchi, editors, *Proc. IFM'99 Integrated Formal Methods.* Springer-Verlag, 1999.
[16] R. Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, 1999.
[17] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.
[18] W. Stallings. *Cryptography and Network Security.* Prentice Hall, 1999.
[19] B.K. Aichernig. Test-case calculation through abstraction. In J.N. Oliveira and P. Zave, editors, *Proc. FME'2001 Symposium Formal Methods Europe.* Springer-Verlag, LNCS Series No. 2021, 2001.
[20] M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: the spi Calculus. *Information and Computation*, 148:1–70, 1999.
[21] L.C. Paulson. Proving Properties of Security Protocols by Induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83. 1997.
[22] D. Dolev and A. Yao. On the Security of Public-key Protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.
[23] G. Lowe. An attack on the Needham-Schroeder public-key protocol. *Information Processing Letters*, 56:131–133, 1995.
[24] R. Focardi, A. Ghelli, and R. Gorrieri. Using non interference for the analysis of security protocols. In H. Orman and C. Meadows, editors, *DIMACS Workshop on Design and Formal Verification of Security Protocols.* DIMACS, Rutgers University, 1997. http://dimacs.rutgers.edu/Workshops/Security.
[25] M. Butler. On the Use of Data Refinement in the Development of Secure Communications Systems. Technical Report DSSE-TR-2001-1, University of Southampton Declarative Systems and Software Engineering, 2001.
[26] I. Zakiuddin, J. Woodcock, M. Goldsmith, and J. Hulance. Formal Verification for Survivable Key Management Systems. In *Proc. IEEE Information Survivability Workshop.* http://www.cert.org/research/isw/isw2000/, 2000.
[27] J. Peleska. Test automation for safety-critical systems: Industrial applications and future developments. In M.-C. Gaudel and J. Woodcock, editors, *Proc. FME'96 Symposium Formal Methods Europe.* Springer-Verlag, LNCS Series, 1996.
[28] R. Back, A. Mikhajlova, and J. von Wright. Reasoning about interactive systems. In J.M. Wing, J. Woodcock, and J. Davies, editors, *Proc. FME'99 Symposium Formal Methods Europe.* Springer-Verlag, LNCS Series No. 1709, 1999.
[29] J. Derrick and E. Boiten. Testing Refinements of State-based Formal Specifications. *Software Testing, Verification and Reliability*, 9:27–50, 1999.
[30] C. Pahl. Analysing Security Properties using Refinement. In *Proc. International Workshop on Refinement of Critical Systems RCS'02*, 2002. (to appear).
[31] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
[32] M. Müller-Olm, D. Schmidt, and B. Steffen. Model Checking – a Tutorial Introduction. In *Proc. 6th Static Analysis Symposium.* Springer-Verlag, LNCS 1694, 1999.