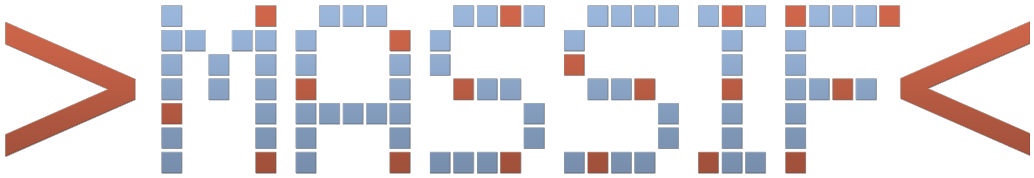

MAnagement of Security information and events in Service InFrastructures

MASSIF
FP7-257475

D4.2.1 - Formal Specification of Security Properties

Activity	A4	Workpackage	WP4.2
Due Date	Month 12	Submission Date	2011-09-30
Main Author(s)	Jürgen Repp (Fraunhofer) Roland Rieke (Fraunhofer)		
Contributor(s)	Andreas Fuchs (Fraunhofer), Nico Lincke (Fraunhofer)		
Version	v1.0	Status	Final
Dissemination Level	PU	Nature	R
Keywords	security modelling framework, security requirements, requirements elicitation		
Reviewers	Igor Kotenko (SPIIRAS) Valerio Formicola (CINI)		

Part of the Seventh
Framework Programme
Funded by the EC - DG INFSO

Version history

Rev	Date	Author	Comments
V0.1	2011-07-11	J. Repp, R. Rieke	initial version in BSCW
V0.2	2011-08-31	J. Repp, R. Rieke	review version
V0.9	2011-09-30	J. Repp, R. Rieke	final version
V1.0	2011-09-30	Elsa Prieto (Atos)	final review and official delivery

Glossary of Acronyms

Abbr	Abbreviation
BSCW	Be Smart - Cooperate Worldwide
CGS	Core Game System
CSS	Cascading style sheets
DoW	Description of Work
EC	European Commission
EU	European Union
FP7	Seventh Framework Programme
MASSIF	MAnagement of Security information and events in Service InFrastructures
MSS	Managed Security Service
MSSP	Managed Security Service Provider
PDC	Primary Data Centre
PU	Public Usage
R&D	Research & Development
RSS	Really Simple Syndication
RTU	Remote Terminal Unit
SeBB	Security Building Block
SeMF	Security Modeling Framework
SIEM	Security Information and Event Management

Executive Summary

This deliverable describes a method which can be used to identify *abstract security requirements* in order to reach generic security goals. Such a formal representation of the security requirements is necessary for different reasoning processes within the novel security information and event management concepts proposed by MASSIF. At configuration time of a SIEM, questions addressed can be:

1. "Which security goals are not covered by security requirements?",
2. "Which information must be provided by the SIEM sensors in order to verify the given requirements?", and "Which requirements cannot be measured by the given SIEM information sources?".

At runtime, we should be able to answer questions like:

3. "Which security requirements and associated security goals are broken by current security event measurements?".

In order to support the answering of theses questions, we show, how a complete set of security requirements for a given high-level security goals can be elicited, which helps to answer question 1. Furthermore, we show, how the derived security requirements from the analysed MASSIF scenarios can be formalised using the proposed modelling framework. This is the first step in the reasoning chain to answer question 2. Question 3 can then be answered by backward chaining of the reasoning in question 2 starting with the measured security information via the broken security requirement to the broken security goal. In MASSIF this analysis is based on a scenario description of a system, providing artifacts such as a scenario description, process models, generic domain knowledge, use cases, misuse cases and the respective security requirements (cf. D2.1.1 [12]). The missing steps in the reasoning chain, namely the derivation of measurement requirements from the abstract security requirements will be subject of a forthcoming deliverable.

Contents

List of Figures

1 Introduction

1.1 Deliverable Context

The activity A4 within MASSIF aims to design and implement new process/attack analysis and simulation techniques in order to be able dynamically to relate events from different execution levels, define specific level abstractions, evaluate them with respect to security issues and during runtime interpret them in context of specific security properties. Activity A4 comprises three main objectives which are covered by respective work packages: WP 4.1 provides dynamic abstraction techniques in order to adapt the actual specification level to the scale of the system to be managed, WP 4.2 evolves advanced methods in the context of predictive security monitoring for the evaluation of security-related events, and WP 4.3 develops a new approach to security evaluation and advanced techniques for attack modelling/simulation, threat analysis and risk evaluation.

This deliverable documents the outcome of task T4.2.1, the first task in WP 4.2. In order to relate security-related events and their interpretation with respect to the required security properties, this task identifies the abstract security properties which will be considered in the formal process analysis in WP4.2. The examples given are based on the MASSIF scenario descriptions in D2.1.1 [12]. In the upcoming deliverables of WP4.2 the identified monitoring rules have to be integrated in the predictive analyser.

1.2 Deliverable Content

The objective of task T4.2.1 is to identify the abstract security properties which will be considered in the formal process analysis. These properties have to be formalised and further they will be refined for each inspected process depending on the requirement analysis done in WP 2.1.

In this deliverable we base the formalisation of the abstract security properties on the Security Modeling Framework (SeMF) developed by Fraunhofer SIT. The predefined property instantiations (cf. Section 2.1.4) build the base for the requirements elicitation process and the refinement process used to identify necessities for system and process monitoring. The method for deriving assumptions in the SeMF terminology, which can be used for reasoning, from domain knowledge and available SIEM capabilities is sketched.

The most important contributions of this work with respect to the guidelines given in section 7 of D2.1.1 [12] are:

Coverage of Security Goals. The proposed requirements elicitation method uses a functional dependency graph which ensures that uncovered aspects of the high-level security goals are revealed.

Information Needs. A lack of SIEM monitoring capabilities would prevent the derivation of assumptions necessary for the reasoning process. This problem would also be detected in the proposed requirements elicitation process (cf. Figure 3.2).

Sufficiency of Monitoring Capabilities. Assumptions can be derived from the monitoring capabilities for reasoning whether the given requirements are fulfilled under these assumptions. This reasoning process can't be successful if monitoring capabilities are insufficient or can't be assigned to entities used in the current abstraction level of the system model.

Traceability. The derived relations between security event measurements, the associated security requirements and corresponding assumptions and the security goals can be used to identify the concrete high-level goals affected by the measurements.

We demonstrate the use of the proposed method by means of important use cases adapted from the deliverable D2.1.1 "Scenario Requirements". The examples are covering the main classical types of abstract security requirements and give an extensive overview of the application of the generic SeMF instantiations.

The remainder of the document is structured as follows: Chapter 2 describes the security modelling framework which is used as the basis for our security requirement definition in MASSIF. In Chapter 3 the application of a model based approach to systematically identify security requirements is presented. In Chapter 4 the elicitation of the classical security properties, authenticity, integrity, and confidentiality is demonstrated for selected use cases from the MASSIF scenarios. Finally, some conclusions and an outlook to further work is given in Chapter 5.

2 Security Properties

We will now describe in detail the Security Modelling Framework SeMF developed by Fraunhofer SIT that will be the basis for our security requirement definition in MASSIF. Exemplary on several MASSIF use cases we present a model-based approach to systematically identify security requirements for the MASSIF scenarios.

2.1 The Security Modelling Framework SeMF

Within SeMF, systems are specified in terms of sequences of actions, while the security properties of systems are specified as specific constraints regarding which sequences of actions can or can not occur. Security properties can be specified regardless of any specific abstraction level. Most basic SeMF elements are result of previous work, e.g. within the European projects SERENITY [1], EVITA [2] and TERESA.

In the following, we first give a brief overview of formal languages as a means to system specification and then introduce our main additional concepts, an agent's (acting entities of the system) *local view* and an agent's *initial knowledge*. We then give the formal definitions of the most important security properties based on these concepts (e.g. authenticity, confidentiality), and some instantiations relevant in MASSIF. It is forseen that further property definitions and instantiations will be necessary in order to capture all security and dependability properties relevant in MASSIF.

2.1.1 Formal Languages

Properties of a concurrent system in the sense of Alpern and Schneider [4] are defined as sets of sequences of states. Similarly, the *behaviour* B of a discrete system can be formally described by the set of its possible sequences of actions (traces). Therefore $B \subseteq \Sigma^*$ holds where Σ is the set of all actions of the system, and Σ^* is the set of all finite sequences of elements of Σ, including the empty sequence denoted by ε. This terminology originates from the theory of formal languages, where Σ is called the alphabet, the elements of Σ are called letters, the elements of Σ^* are referred to as words and the subsets of Σ^* as formal languages. Words can be composed: if u and v are words, then uv is also a word. This operation is called the *concatenation*; especially $\varepsilon u = u\varepsilon = u$. A word u is called a *prefix* of a word v if there is a word x such that $v = ux$. The set of all prefixes of a word u is denoted by $\mathrm{pre}(u)$; $\varepsilon \in \mathrm{pre}(u)$ holds for every word u. We denote the set of letters in a word u by $alph(u)$ and the number of occurences of any action of a set Γ in a word u by $card(\Gamma, u)$. If Γ consists of only one action a, we simply say $card(a, \omega)$.

Formal languages that describe system behaviour have the characteristic that $\mathrm{pre}(u) \subseteq B$ holds for every word $u \in B$. Such languages are called *prefix closed*. System behaviour is thus described by

prefix closed formal languages.

The set of all possible continuations of a word $u \in B$ is formally expressed by the *left quotient* $u^{-1}(B) = \{y \in \Sigma^* \mid uy \in B\}$.

Different formal models of the same application/system are partially ordered with respect to different levels of abstraction. Formally, abstractions are described by so called alphabetic language homomorphisms. These are mappings $h^* : \Sigma^* \longrightarrow \Sigma'^*$ with $h^*(xy) = h^*(x)h^*(y)$, $h^*(\varepsilon) = \varepsilon$ and $h^*(\Sigma) \subseteq \Sigma' \cup \{\varepsilon\}$. So they are uniquely defined by corresponding mappings $h : \Sigma \longrightarrow \Sigma' \cup \{\varepsilon\}$. In the following we denote both the mapping h and the homomorphism h^* by h. These homomorphisms map action sequences of a finer abstraction level to action sequences of a more abstract level.

Classical liveness and safety properties can easily be specified for such a system using well known formalisations. For security properties, we need to extend the system model by taking into account the agents' view of the system and agents' knowledge about the global system behaviour.

2.1.2 Agents' Local Views and Initial Knowledges

Security properties can only be satisfied relative to particular sets of underlying system assumptions. Examples include assumptions on cryptographic algorithms, secure storage, trust in the correct behaviour of agents or reliable data transfer. Relatively small changes in these assumptions can result in huge differences concerning satisfaction of security properties. Every model for secure systems must address these issues. However, most existing models rely on a fixed set of underlying assumptions (see for example [5] and [13]). Most of these assumptions are often implicitly given by particular properties of the model framework. Thus, it is very hard to verify whether a particular implementation actually satisfies all of these assumptions. Further, imprecise security assumptions might result in correct but useless security proofs and finally in insecure implementations. Therefore, a model for secure systems needs to provide the means to accurately specify underlying system assumptions in a flexible way.

In order to provide the required flexibility, SeMF extends the system specification by two components: *agents' knowledge* about the global system behaviour and *agents' view*. The knowledge about the system consists of all traces that an agent initially considers possible, i.e. all traces that do not violate any system assumptions, and the view of an agent specifies which parts of the system behaviour the agent can actually see. In the following paragraphs, these two components and their relations are explained in detail.

Agents' Knowledge about the Global System Behaviour

For any agent P its knowledge $W_P \subseteq \Sigma^*$ about the global system behaviour is considered to be part of the system specification.

We may assume for example that a message that was received must have been sent before. Thus an agent's W_P will contain only those sequences of actions in which a message is first sent and then received. All sequences of actions included in W_P in which a digital signature is received and verified by using some agent Q's public key will contain an action where Q generated this signature.

Care must be taken when specifying the sets W_P for all agents P in order not to specify properties that are desirable but not guaranteed by verified system assumptions. In a setting for example where we assume one time passwords are used, if P trusts Q, W_P contains only those sequences of actions in

which Q sends a certain password only once. However, if Q cannot be trusted, W_P will also contain sequences of actions in which Q sends a password more than once.

The specification of the desired system behaviour generally does not include behaviour of malicious agents which has to be taken into account in open systems. An approach which is frequently used for the security analysis of cryptographic protocols is to extend the system specification by explicit specification of malicious behaviour. However, in general malicious behaviour is not previously known and one may not be able to adequately specify all possible actions of dishonest agents. In our approach, the explicit specification of agents' knowledge about system and environment allows to discard explicit specification of malicious behaviour. Every behaviour which is not explicitly excluded by some W_P is allowed. Denoting a system containing malicious behaviour by B and the correct system behaviour by B_C, we assume $B_C \subseteq B \subseteq \Sigma^*$. We further assume $B \subseteq W_P$, i.e. every agent considers the system behaviour to be possible, as reasoning within SeMF primarily targets the validation and verification of security properties in terms of positive formulations, i.e. assurances the agents of the system may have. Other approaches that deal with malfunction, misassumptions and attacker models can not rely on this assumption.

Security properties can now be defined relative to W_P. The relation between the system behaviour without malicious actions B_C, the system behaviour including malicious actions B, and W_P is graphically shown in Figure 2.1.

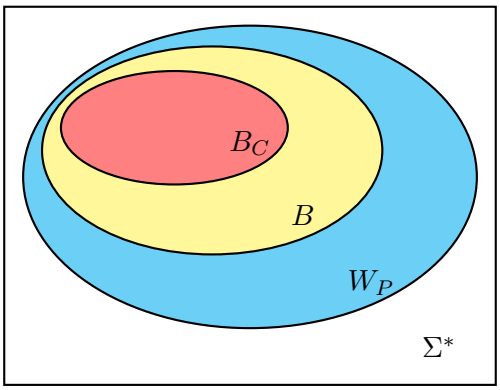

Figure 2.1: System behaviour and W_P

Agents' View of the Global System Behaviour

The set W_P describes what P knows initially. However, in a running system P can learn from actions that have occured. Satisfaction of security properties obviously also depends on what agents are able to learn. After a sequence of actions $\omega \in B$ has happened, every agent can use its *local view* of ω to determine the sequences of actions it considers to have possibly happened after ω has happened. In order to determine what is the local view of an agent, we first assign every action to exactly one agent. Thus $\Sigma = \dot{\bigcup}_{P \in \mathbb{P}} \Sigma_{/P}$ (where $\Sigma_{/P}$ denotes all actions performed by agent P, and $\dot{\bigcup}$ denotes the disjoint union). The homomorphism $\pi_P : \Sigma^* \to {\Sigma^*}_{/P}$ defined by $\pi_P(x) = x$ if $x \in \Sigma_{/P}$ and $\pi_P(x) = \varepsilon$ if $x \in \Sigma \setminus \Sigma_{/P}$ formalises the assignment of actions to agents and is called the *projection* on P.

The projection π_P is the correct representation of P's view of the system if all information about an action $x \in \Sigma_{/P}$ is available for agent P and P can only see its own actions. In this case P's local

view of the sequence of actions $\omega = send(P, m1)rec(Q, m1)$ for example is $send(P, m1)$. However, P's view may be finer. For example it may additionally note a message that was sent over a network bus without being able to see who sent it, in which case P's local view of $send(sender, message)$ e.g. is $send(message)$. P's local view may also be coarser than π_P. In a system the actions of which are represented by a triple (*global state, transition label, global successor state*), although seeing its own actions, P will not be able to see the other agents' states. Thus, we generally denote the local view of an agent P on Σ by $\lambda_P : \Sigma^* \to \Sigma_P^*$. The local views of all agents together contain all information about the system behaviour B.

For a sequence of actions $\omega \in B$ and agent $P \in \mathbb{P}$, $\lambda_P^{-1}(\lambda_P(\omega)) \subseteq \Sigma^*$ is the set of all sequences that look exactly the same from P's local view after ω has happened. In the above network bus example all actions in $\{send(sender_1, message), send(sender_2, message), \ldots\}$ look identical for P. Depending on its knowledge about the system S, underlying security mechanisms and system assumptions, P does not consider all sequences in $\lambda_P^{-1}(\lambda_P(\omega))$ possible. Thus it can use its initial knowledge to reduce this set: $\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$ describes all sequences of actions P considers to have possibly happened when ω has happened. This set is similar to the possible worlds semantics that have been defined for authentication logics in the context of cryptographic protocols [3, 14]. Our notion is more general because for authentication logics λ_P and W_P are fixed for all systems, whereas in our approach they can be defined differently for different systems. The knowledge of P relative to a sequence of actions ω is graphically shown in Figure 2.2.

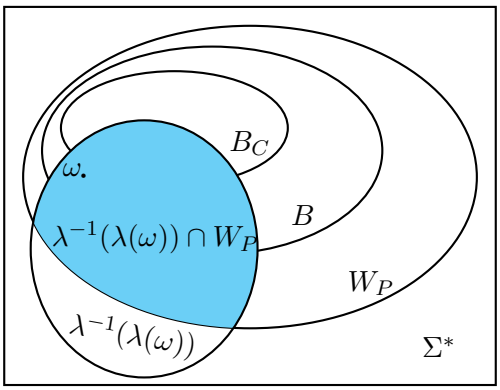

Figure 2.2: P's world after ω has happened

2.1.3 Basic Property Definitions

The concepts introduced in the previous section will now be used to define the most important security properties. Further definitions are given in Appendix A.

Authenticity

Authenticity can be seen as the assurance that a particular action has occurred in the past.[1]

[1] Please note that in order to achieve *authentication* one additionally needs assurance about the time of the occurrence of the action (see Section 6.1 *authenticity with respect to a phase*).

Since usually authenticity of some action for a certain agent is required, the definition has to refer in some way to the agent. Thus we call a particular action *a* authentic for an agent *P* if in all sequences that *P* considers to have possibly happened after a sequence of actions ω has happened, some time in the past a must have happened. By extending this definition to a set of actions Γ being authentic for P if one of the actions in Γ is authentic for P we gain the flexibility that P does not necessarily need to know all parameters of the authentic action. For example, a message may consist of one part protected by a digital signature and another irrelevant part without protection. Then, the recipient can know that the signer has sent a message containing the signature, but the rest of the message is not authentic. Therefore, in this case, Γ comprises all messages containing the relevant signature and arbitrary other message parts.

The formal definition for authenticity is as follows.

Definition 1 (Authenticity) *A set of actions $\Gamma \subseteq \Sigma$ is authentic for $P \in \mathbb{P}$ after a sequence of actions $\omega \in B$ with respect to W_P if $alph(x) \cap \Gamma \neq \emptyset$ for all $x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$.*

Integrity

Note that in most security-oriented frameworks data origin authenticity implies integrity.

Confidentiality

Confidentiality is required for data occurring in different types of actions. Among others these can be actions concerned with sending or receiving data or manipulating data stored on a particular device. We may also want to keep the agent acting confidential for privacy reasons. An adequate notion of confidentiality therefore has to provide the flexibility to define confidentiality for arbitrary parameters of the actions. The notion of *parameter-confidentiality* presented in [9] provides this flexibility (cf. Appendix A).

2.1.4 Specific Property Instantiations

The basic notions for the accurate specification of security requirements are very general, and using these notions to specify more concrete security requirements can result in very complex expressions. As such, they are difficult to handle when it comes to including them into standard approaches for systems modeling and verification. Consequently, in the SERENITY project [11], we started the definition of a requirements language using refined notions that describe concrete instantiations of the above security requirements. This language has been further extended in the EVITA project [2] and is further extended in the course of the TERESA project. In MASSIF we use these results to specify the security and dependability properties for the given scenarios.

Definition 2 (auth) *For $P \in \mathbb{P}$ and $a, b \in \Sigma$, $auth(a, b, P)$ holds in B if for all $\omega \in B$ that contain an action b, action a is authentic for agent P after ω corresponding to Definition 1.*

Note that in most cases, b will be in P's local view.

The following two security requirements are instantiations of *enforce-behaviour*. The first one describes that in all sequences of actions, whenever action b happens, action a must have happened before, the second one describes the opposite (whenever b has happened, a must not have happened before).

Definition 3 (precede) *Let Σ be a set of actions, $a, b \in \Sigma$. Let $B, L \subseteq \Sigma^*$ with L defined as follows:*

$$L = \Sigma^* \setminus ((\Sigma \setminus \{a\})^*\{b\}\Sigma^*)$$

Then $precede(a, b)$ holds if enforce-behaviour(L, B) *holds.*

Definition 4 (not-precede) *Let Σ be a set of actions, $a, b \in \Sigma$. Let $B, L \subseteq \Sigma^*$ with L defined as follows:*

$$L = \Sigma^* \setminus (\Sigma^*\{a\}\Sigma^*\{b\}\Sigma^*)$$

Then $not\text{-}precede(a, b)$ holds if enforce-behaviour(L, B) *holds.*

Definition 5 (not-happens) *Let Σ be a set of actions, $a \in \Sigma$. Let $B \subseteq \Sigma^*$*
Then $not\text{-}happens(a) := \forall \omega \in B : a \notin alph(\omega)$ holds.

For the specification of anonymity and privacy requirements, we can use the notion of *confidential* reflecting a specific confidentiality property according to Definition 15. Generally in order to specify confidentiality, we need to specify:

- the agents who are allowed to know the data to be confidential,
- the set M of possible values of the parameter that shall be confidential,
- the homomorphism μ that identifies all actions that contain the parameter that shall be confidential and that a malicious agent can use to gain knowledge about the parameter (μ maps these actions onto their "types" and at the same time extracts the parameter to be confidential: $\mu : \Sigma^* \longrightarrow (\Sigma_t \times M)^*$),
- the language L that characterizes the relations between actions that are allowed to be known by a malicious agent (e.g. the relation reflecting that the same message occurs in a specific send and receive action),
- the local views of malicious agents,
- the initial knowledges of malicious agents.

The parameters or the confidential property are instantiated as follows:

- We do not fix the set of agents that are allowed to know the parameter that shall be confidential. This set is denoted by who.
- We fix the "types" of actions the homomorphism μ maps onto: Let $action(p_1, \ldots, p_j)$ be an action in Σ with p_i being the parameter to be confidential. W.l.o.g. let $p_i = p_1$. Then the type of this action is $(action, p_2, \ldots, p_j)$ and $\mu(action(p_1, \ldots, p_j)) = ((action, p_2, \ldots, p_j), p_1)$. In other words, μ keeps all parameters of the action except the one to be confidential which it extracts to form the second component of the action's image under μ. (Note that μ maps those actions that can not be used to extract knowledge about the parameter value to ε.)

  Having fixed this, the only information that is still needed is which is the parameter that we want to be confidential (the message m being sent/received, the data d being stored, the agent P performing a specific action, etc.), and which are the actions that a malicious agent can use to gain knowledge about the parameter to be confidential. We denote the parameter by par and the set of actions by $\mathcal{A}(par)$.

- Regarding the relations between actions that are allowed to be known by a malicious agent, we will use languages $\mathcal{L}$ to denote all dependencies that must be assumed to be known. We should for example assume that the agents know the relation between encryption and decryption with the same shared secret. Later we may fix specific languages $\mathcal{L}$.

Definition 6 (confidential) *Let $who \subseteq \mathbb{P}$ be a subset of agents, par be the parameter whose value shall only be known by the agents in who, and $\mathcal{A}(par)$ be the set of actions from which a malicious agent can extract knowledge about the value of par. Let further $\mathcal{M}$ denote the possible values of par and $\mathcal{L}$ denote the language that captures all the allowed relation knowledge between actions in the sense explained above. Then $conf(\mathcal{A}(par), par, \mathcal{M}, \mathcal{L}, who)$ holds in B if $\mathcal{M}$ is parameter-confidential for all $P \in \mathbb{P} \setminus who$ with respect to $(\mathcal{L}, \mathcal{M})$–completeness according to Definition 15.*

Definition of availability: The definition of the availability property is not finalized and therefore not included in this deliverable.

3 Security Requirements Elicitation Process

Information flow between systems and system components is highly complex, especially given that a system can evolve via the replacement of its components. Consequently, an important aspect of security evaluation is the analysis of the potential information flows. We use the analysis of the potential information flows to derive the dependencies for the functional model.

For the description of the functional model from the use cases, an action-oriented approach is chosen. The approach is based on the work in [7]. For reasons of simplicity and readability the formal description of the model is omitted here and a graphical representation is used to illustrate the behaviour of the SIEM target. Actions in SeMF represent an abstract view on actions of the real system. Thereby the semantics of the actions of the real system is reduced to the interdependencies of these actions and ignores the functionality of the actions. necessary step of abstraction that lies outside of the SeMF semantics. Within SeMF we (currently) use the convention of a parameterized format consisting of the actions name, the acting agent and a variable set of parameters:

$$actionName(actingAgent, parameter1, parameter2, ...)$$

In order to document this step of abstraction from reality to SeMF model and to reduce ambiguity, the meaning of each of the SeMF actions should be defined accordingly.

3.1 Functional Model

A functional model can be derived from a use case description by identifying the atomic actions in the use case description. These actions are set into relation by defining the functional flow among them. This action oriented approach considers possible sequences of actions (control flow) and information flow (input/output) between interdependent actions. Functional models that describe only parts of the overall system behaviour will be called *functional component model*.

Figure 3.1 shows the functional component model for typical use cases in the dam scenario. To keep the example clear only templates for real actions (sense, action) are used. The following agents and their abbreviations are involved in the example:

- Control Station (CS)
- Monitoring Station (MS)
- Remote Terminal Unit (RTU)

The agents ($\mathbb{P} = \{CS, MS, RTU\}$) are able to perform the following set of actions:

- $send(x, data), receive(x, data)$ with $x \in \mathbb{P}$
  Sending and receiving of data by the entities of the system.
- $sense(RTU, S, v)$
  Reading of the current measurement v of the sensor S by the RTU.
- $read(CS, t_1, \ldots t_m)$
  Reading of the threshold values determining the execution of system actions.
- $action(t_1, \ldots t_m, (S_1, v_1), (S_2, v_2))$
  Execution of the system action $action$ depending on the sensor values v_1 and v_2 an the threshold values $t_1, \ldots t_m$.

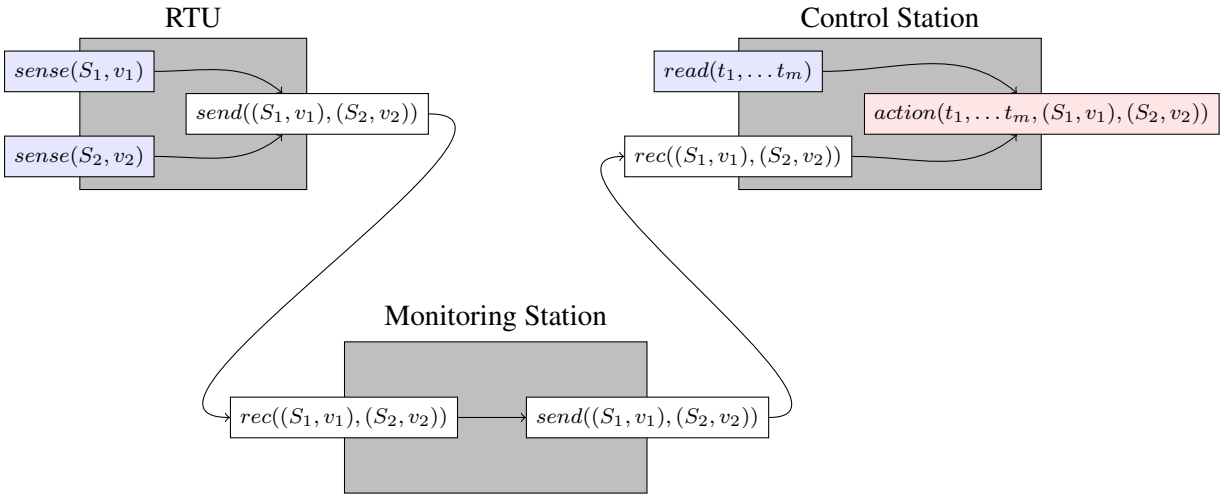

Figure 3.1: Dam use case 1

This model is derived from example use cases given in deliverable D2.1.1 [12]. The arrows outside of the components boundaries refer to functional flows between the components, whilst internal flow arrows refer to flows within the same instance of the component. For the given example, the external flows represent data transmission of one system to another, whilst the internal flows represent communication within a single system.

3.2 Functional Security Requirements Identification

The functional component model is used in a next step to derive security requirements. First, the boundary actions of the system model components are determined. Let the term *boundary action* refer to the actions that form the interaction of the internals of the system with the outside world. These are actions that are either triggered by occurrences outside of the system or actions that involve changes to the outside of the system.

With the boundary actions being identified, one may now follow the functional graph backwards. Beginning with the boundary actions by which the system takes influence on the outside, we may propagate backwards along the functional flow. These backwards references basically describe the functional

dependencies of actions among each other. From the functional dependency graph we may now identify the end points, namely the boundary actions that trigger the system behaviour that depends on them. Between these and the corresponding starting points, the requirement exists that without such an action happening as input to the system, the corresponding output action must not happen as well.

For the example depicted in Figure 3.1, the security requirement "The system must assure that all (safety critical) actions using sensor data must only get authentic sensor data" was given.

This is a specific form of a more general security goal of the system at stake:

Whenever a certain output action happens, the input actions that presumably led to it must actually have happened.

The formal derivation of security requirements from such kind of high-level goals is given in more detail in [7].

In our case, this leads to the following SeMF $precede$ property for the sensors (S), sensor values (V), the threshold values (T), and the control station (CS):

$$\forall S_i \in S, \forall v_i \in V, \forall t_j \in T :$$
$$precede(sense(RTU, S_i, v_i)\ \&\ read(CS, t_j), action(CS, (S_1, v_1), \ldots (S_n, v_n), t_1, \ldots t_m)$$

Coverage of Security Goals.

The original security goal was formulated related to the measured sensor data while the functional dependency graph suggests also to include threshold values into the security goals and the derived security requirements. This is one example for answer to the question whether uncovered security goals are revealed.

According to the model domain and SIEM capabilities monitoring rules have to be developed. Figure 3.2 illustrates this process.

Hard and Soft Facts.

Monitoring rules directly deduce from assumptions given by hard facts of the domain knowledge. "Soft facts" from the domain knowledge can be processes, guidelines etc.

In this case the monitoring rules are given by these facts and the assumptions have to be derived from these rules. These assumptions have to be checked against the hard facts from the domain knowledge. In either cases it has to proven that the combination of assumptions and monitoring rules fulfill the requirements. It has to be checked, whether the SIEM monitor capabilities match the monitoring rules. This process is explained step by step in section 4.1.

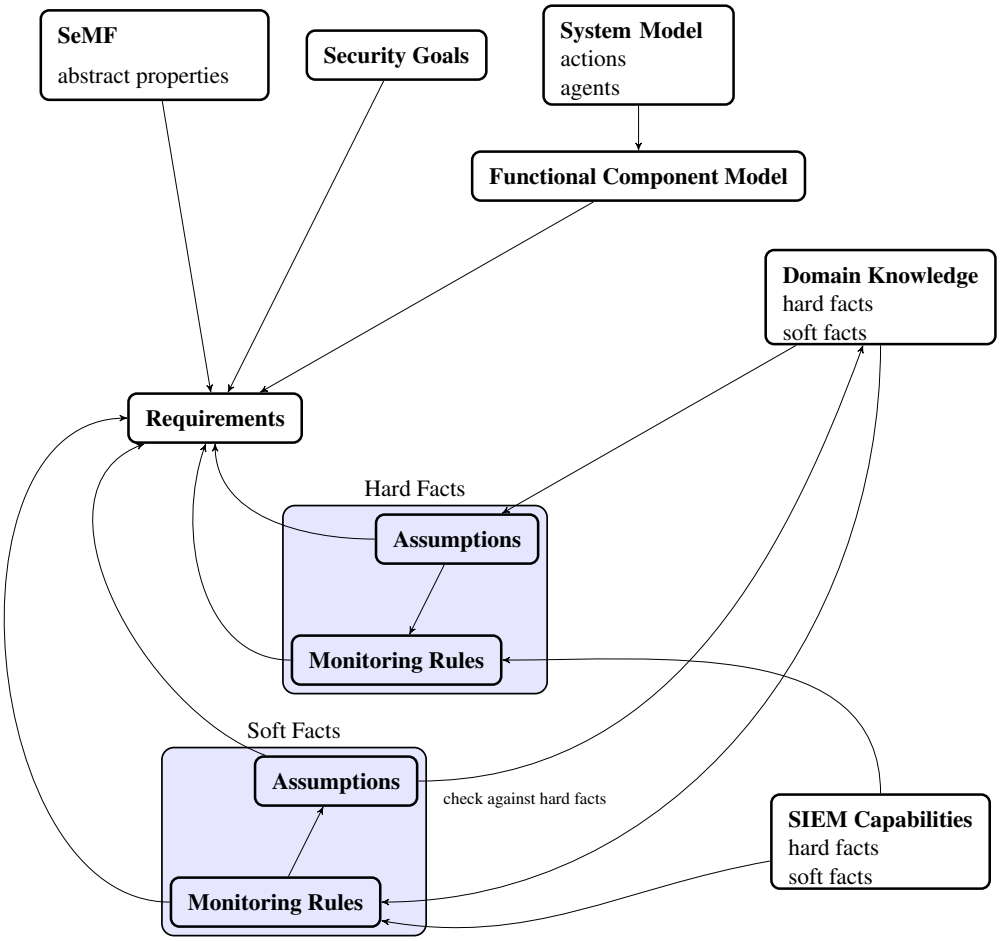

Figure 3.2: Workflow of the requirements elicitation process

4 Specification of Security Properties

In the following sections the elicitation of the classical security properties, authenticity, integrity, and confidentiality will be demonstrated for selected use cases from the deliverable "D2.1.1 - Scenario Requirements".

4.1 Data Integrity in the Dam Scenario

Step 1: The security goals have to be identified.

- The system must assure that all (safety critical) actions using sensor data must only get authentic sensor data.

Step 2: Identification of safety critical actions.

- Open/close penstock gate according to the parameters of the command received by the control station.

Step 3: Identification of functional dependencies.
All actions related to information flow corresponding to the safety critical actions and their parameters have to be gathered. In our example these dependencies are extracted from misuse case 6 of the dam scenario. Figure 4.1 shows the dependency graph. The term $sense(PP_{state})$ represents the measurement of voltage and current in the power grid. The power plant sends commands to the control station depending on these measurements. The term $sense(SDC, vdc)$ describes the measurement of the water discharge on the penstock gates. The dashed line indicates that there is no direct functional dependency, but the decision of the operator, which operation shall be triggered, depends on the displayed measurements.

Step 4: Now the actions that form the interaction of the internals of the system with the outside world (blue nodes) have to be identified. In the following, these actions will be called boundary actions. As an extension to the graphical representation, the actors will be included in all action statements in textual form as first parameter.

- sense: discharge sensor ($sense(RTU, SDC_{val})$)
- sense: Power plant state ($sense(PP, PP_{state})$)
- display of the control station:
  - discharge level on the penstocks (SDC_{val})
  - hydroelectric power plant operative (PP_{state}: current, voltage,...)

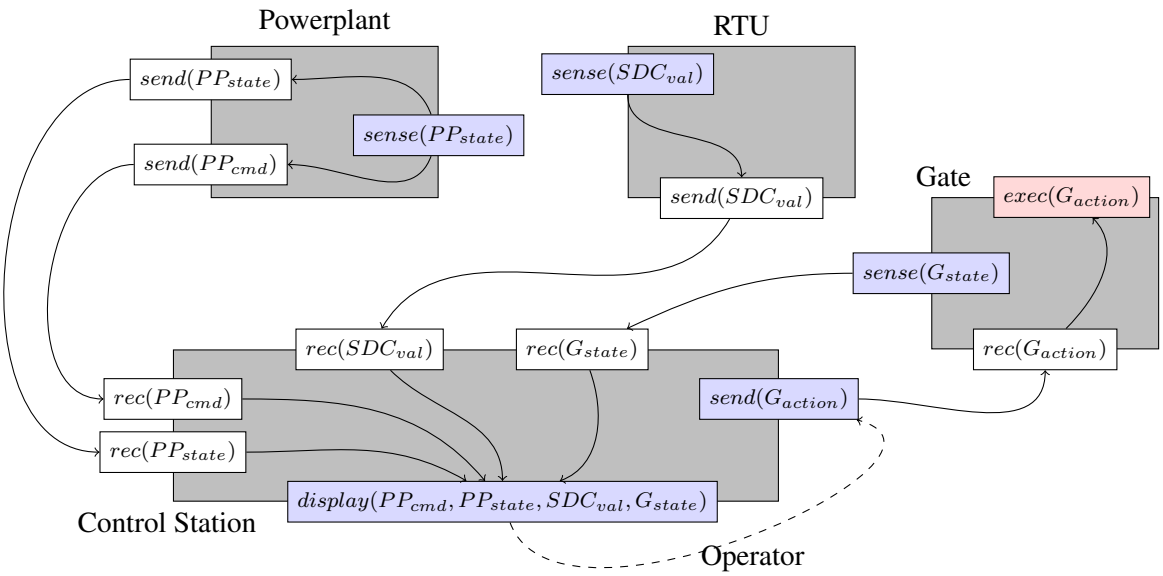

Figure 4.1: Functional Dependencies

Step 5: The following parameters (assets) are used in our boundary actions:

- discharge sensors (SDC)
- discharge values (SDC_{val})
- power plant (PP)
- power plant operation PP_{state}
- penstock gate (G)

Step 6: Now in the next step the security requirements can be derived from the security goal related to the identified boundary actions:

- The measurements displayed for the $CS\_Administrator$ on the control station have to be authentic for the gate system:

$$auth(\{sense(RTU, SDC_{vdc}), sense(PP, PP_{state}), sense(G, G_{state}\},$$
$$display(CS, PP_{cmd}, PP_{state}, SDC_{val}, G_{state}), CS\_Administrator)$$

- The sending of the command to the $GATE$ has to be authentic:

$$auth(send(CS, G_{action}), exec(G, G_{action}), G)$$

Step 7: These requirements have to be matched with existing SIEM monitoring capabilities. It has to be checked whether there is a domain knowledge, which can be set into relation with the security requirements. It is assumed that the following SIEM monitoring capabilities related to the parameters of our model are given:

1. state of the penstock gate
2. discharge sensor
3. power plant operation state (voltage, current...)

The following assumptions follow from the domain knowledge:

- discharge level is related to current produced by the power plant
- discharge level is related to penstock gate state

Step 8: Assessment of the developed monitoring capabilities must follow:

- Risk assessment related to the fulfillment of derived assumptions for the given monitoring capabilities
- Evaluation whether further refinements are necessary

A refinement of the developed monitoring rules is always necessary, if the chosen abstraction level prevents assignment of monitoring possibilities to assumption related to the given requirements. In other cases depending on the underlying security measures this decision on the refinement process has to be taken. For instance refinement will be necessary if hop-by-hop security measures are used, while end-to-end security measures do not require further refinements.

Technical Integration. Technical integration of the presented approach has to be done. The specification of the SeMF models will be integrated into the Eclipse development environment. Figure 4.2 shows a screenshot of the specification of our current example. The further development of this Eclipse plugin and of the interfaces to other MASSIF components will be done in the following work packages. This editor will be part of the model management GUI.

4.2 Access Control in Dam Scenario

In the example presented in this section the requirement elicitation process will be skipped. The formal requirement definition is directly derived from the textual definition. The main emphasis will be put on the elicitation of monitoring rules and assumptions. Additional to the scenario deliverable the answers of the scenario providers to the following questions build the basis for the presented analysis.

Question: Is there any role concept in the dam scenario for the execution of any actions?

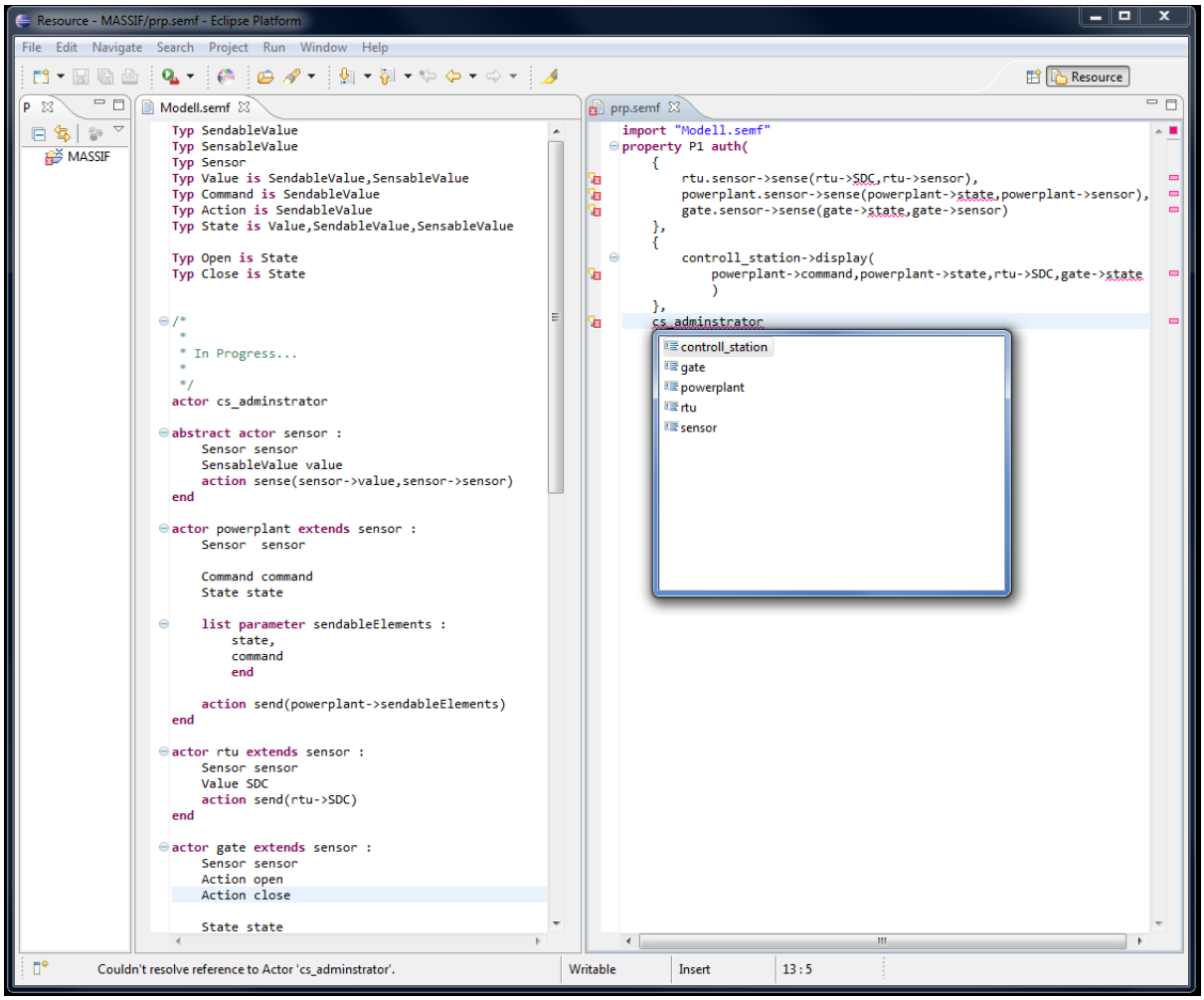

Figure 4.2: Screenshot of a SeMF specification in Eclipse environment

Answer: A dam control system is usually provided with different operational roles. Usually these roles include: Administrators of the control station (allowed to perform both monitoring and control operations and also to modify settings and parameters), monitoring operators (allowed only to access the monitoring data), and visualization roles provided with different levels of visualization rights over the collected data.

Question: What actions are executed automatically and are there any actions which need human participation?

Answer: This depends on the country that we take into account: while in Italy the law compels that all the dams must be always supervised by a human operator and also that all the operations (i.e. opening the gates) are carried out or at least supervised by a human operator, in other countries the law is less strict and allows actuators to be controlled remotely, from the control station, allowing the creation of totally unmanned dams.

Based on this information from the scenario partners, we analyse the following security goal:

Security Goal:

The system must assure that all (safety critical) actions are carried out or at least supervised by a human operator. (G1)

Semi-formal description:

Beside the dam operators also other staff which might carry out other tasks in the control center is modelled. $\mathcal{P}(set)$ defines the powerset of a set. $Trigger$ defines the set entities which can cause an action. The control center administrator will be an element of this set.

$Supervisors = CS\_Administrators \cup Monitoring\_Operators$
$Staff_{Supervisors} = \mathcal{P}(Supervisiors)$
$Staff_{all} = \mathcal{P}(Supervsiors \cup Other\_Staff)$
$Trigger = \{Other\_Entity\} \cup CS\_Administrators$
$Actions = Actions_{supervised} \cup Actions_{exec\_by\_operator}$
Agents and SeMF Actions:
$\mathbb{P} = Staff_{all}$
$\Sigma = \{execute(A, T, X) | A \in Actions, T \in Trigger, X \in Staff_{all}\}$
The parameter T stands for the entity starting the action A, X is the set of persons present in the control center.

Requirements:

$$not\_happens(\{execute(Actions_{supervised}, X, S) | (S \cap Supervisors) = \{\}\} \quad \text{(R1)}$$

$$not\_happens(\{execute(Actions_{exec\_by\_operator}, X, Y) \mid X \notin \{CS\_Administrator\} \wedge X \in Y\}) \quad \text{(R2)}$$

The conditions for the fulfillment of R2 will be investigated in the following part of this section.

SIEM monitoring possibilities (related to parameters of the security requirements):

Enter / leave control center using a RFID badge.
Login / logout control center workstation

Domain Knowledge (related to parameters of the security requirements):

According to the model domain assumptions related to the requirements have to defined. The domain knowledge can be classified as follows:

- Hard facts with low risk.
- Soft facts with medium risk, e.g. processes controlled by technical measures, policies, etc.
- Workflows, processes, guidelines, e.g. not enforced by technical measures with high risk

These facts together with the requirements define properties of our system model. Hard facts directly imply assumptions for our system model, while assumptions for soft facts need to be derived, which will be explained on our example. If hard facts can be monitored, violation of these facts implies that related assumptions derived from soft facts and the corresponding requirements are broken. SeMF provides techniques for reasoning on these system properties to check whether the

requirements are fulfilled under the given assumptions. The classification of the domain knowledge does not act as a part in this reasoning. A separate risk assessment will be needed, to decide whether facts will be classified as hard or soft facts.

A possible classification of the domain knowledge to the given requirements would be:

Hard fact: Actions to be executed by $CS\_Administrators$ are only possible after login by $CS\_Administrator$.

Hard fact: Only people with valid RFID Badge can enter the control center.

Soft fact: Login of $CS\_Administrators$ is only possible with physical presence in the control center.

Workflow/guideline: $CS\_Administrators$ will logout before leaving the control center.

The assumptions have to be formalised using SeMF predicates:

$$not\_happens(\{execute(Actions_{supervised}, X, S) \mid X \in Staff_{all} \wedge X \notin CS\_Administrators \wedge (S \cap CS\_Administrators) \neq \{\}\}$$

This formula expresses the fact that if a $CS\_Operator$ is logged in and is physically present in the control center no one else can execute actions with his permissions. Further assumptions (here only given informal) are necessary:

1. Actions to be executed can only be executed after successful login of a $CS\_Administrator$
2. Login is only possible in the phase of entering and leaving the control center.
3. The administrator logs out before leaving the control center.

The guideline for the administrator could be given by the UML state diagram Figure 4.3.

For the formal definition of these assumptions the phase definition provided in [8] can be used. The proof of the requirements with the given assumptions will be possible. But a monitor automaton constructed on the base of the UML diagram 4.3 would also produce false positives, because the assumptions are only sufficient conditions for the requirements. To exclude false positives and get necessary conditions, a second proof is necessary:

$$\neg Monitoring\_Rule \implies \neg Requirement$$

which can be transformed to:

$$Requirement \implies Monitoring\_Rule$$

The monitor automaton Figure 4.4 describes the assumptions necessary to prove.

State S_4 and S_5 are not part of the monitor automaton and are used to illustrate the possible exceptions and possible mitigation strategies.

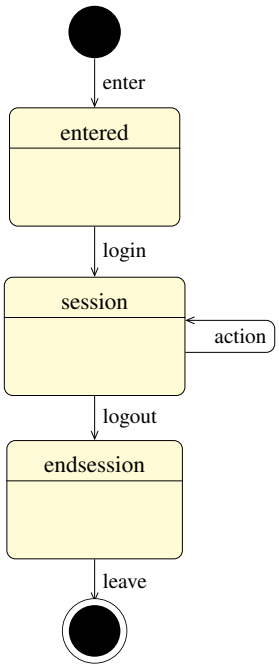

Figure 4.3: UML state diagram of the behaviour of the control station administrator (CS_A)

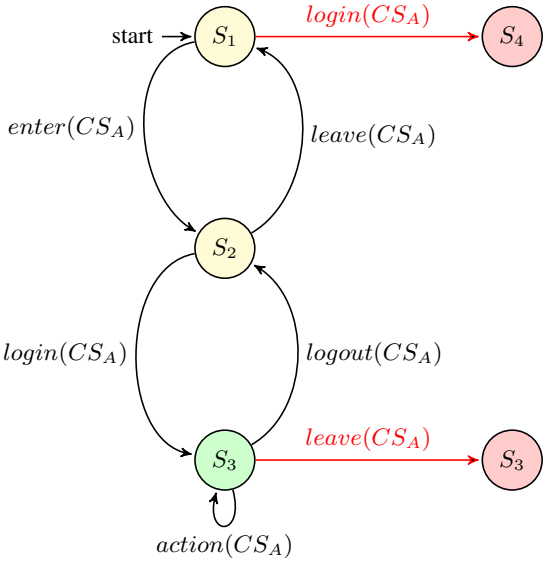

Figure 4.4: Monitor automaton for the control station administrator (CS_A)

Traceability.

These exceptions would cause security events and are the answer to the question "Which security requirements and associated security goals are broken by current security event measurements ?". The linkage between these events, the security requirement R2 with the corresponding assumptions the security goal G1 was developed in the presented process.

So the requirements and assumptions imply monitoring rules given by state $S_1 \ldots S_3$ of the automaton. The violation of a monitoring rule implies that a requirement does not hold or an assumption is broken. The proof could also be conducted with the following weaker assumption instead of assumption 3:

- No one except the control station administrators is allowed to enter the control center if an administration session is active but no administrator is present.
- The control center administrator is not allowed to leave the control center during active session when other staff is present in the control center.

The following monitor automaton (Figure 4.5) can be derived from these assumptions:

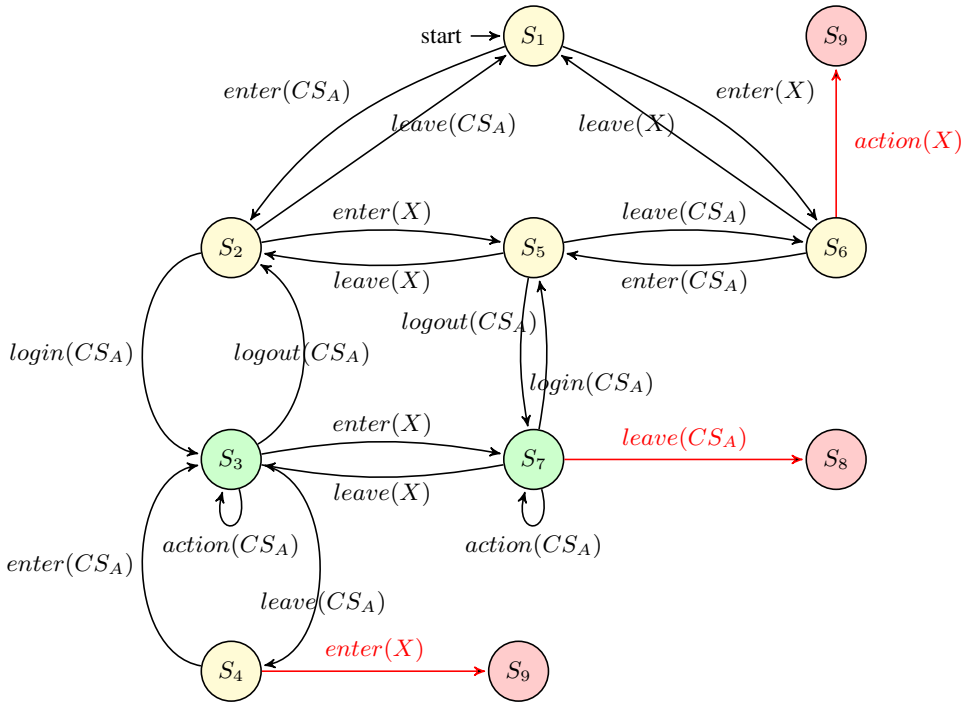

Figure 4.5: Monitor automaton w.r.t. the control station staff

This automaton monitors the necessary conditions for the given requirements. The red states $s8$ and $s9$ are not part of the monitor automaton and are used to illustrate some possible exceptions. An assessment whether appropriate mitigation techniques are available for exceptions of this monitor automaton is necessary. The implementation of technical measures to prevent exceptions or

defining “stronger” assumptions could be the consequence if mitigation possibilities are not adequate.

Information Needs.
The question “Which information must the SIEM sensors provide and which additional assumptions to the environment are necessary in order to verify the given requirements ?” is answered in this section. The lack of SIEM monitoring capabilities would prevent the derivation of assumptions necessary for the reasoning process.

4.3 Data Access Isolation in the Olympic Games Scenario

Data access isolation between users is a security goal with high priority in the Olympic Games scenario. Figure 4.6 shows the information flow and the functional dependencies assigned to the components presented in the high level architecture diagram from the scenario description in [12]. Data access isolation is formalised using the SeMF properties $auth$ and $conf$ with the agents, Core Game System (CGS) user (CGS_U), other CGS other user (CGS_O) remote info user (RI_U), other remote info users (RI_O), and the Primary Data Centre (PDC) provider (A).

The set of agents is defined as follows:

$$\mathbb{P} := \{CGS_U,\ CGS_O,\ RI_U,\ RI_O, A\}$$

Data delivered by customer CGS_U has to be delivered to remote info user RI_U. $\mathcal{M}(d)$ is the set which is indistinguishable related to customer data d, while $\mathcal{A}(d)$ describes the actions out of the alphabet Σ where customer data does occur as a parameter:

$$\mathcal{A}(d) := \{a \in \Sigma \mid d \in params(a)\}$$

The trigger actions for the predicate authenticity property $auth$ are the provision of data by the PDC provider A and the delivery of user data to the PDC system:

$$\forall d \in \mathcal{M}(d) : auth(\{send(CGS_U, d), provide(A, (CGS_U, data)\}, rec(CGS_U, d), CGS_U)$$
$$\forall d \in \mathcal{M}(d) : auth(\{send(CGS_U, d), provide(A, (CGS_U, data)\}, rec(RI_U, d), RI_U)$$

These actions have to be authentic for the CGS user CGS_U and for the remote info system user RI_U. The boundary action are shown in blue in Figure 4.6. The $auth$ predicate defines that in each trace of actions one of the trigger actions must have happened before the customer or remote info user receives data. For the definition of confidentiality the set who

$$who := \mathbb{P} \setminus \{CGS_O, RI_O\}$$
$$conf(\mathcal{A}(d), d, \mathcal{M}(d), \mathcal{L}_{max}, who)$$

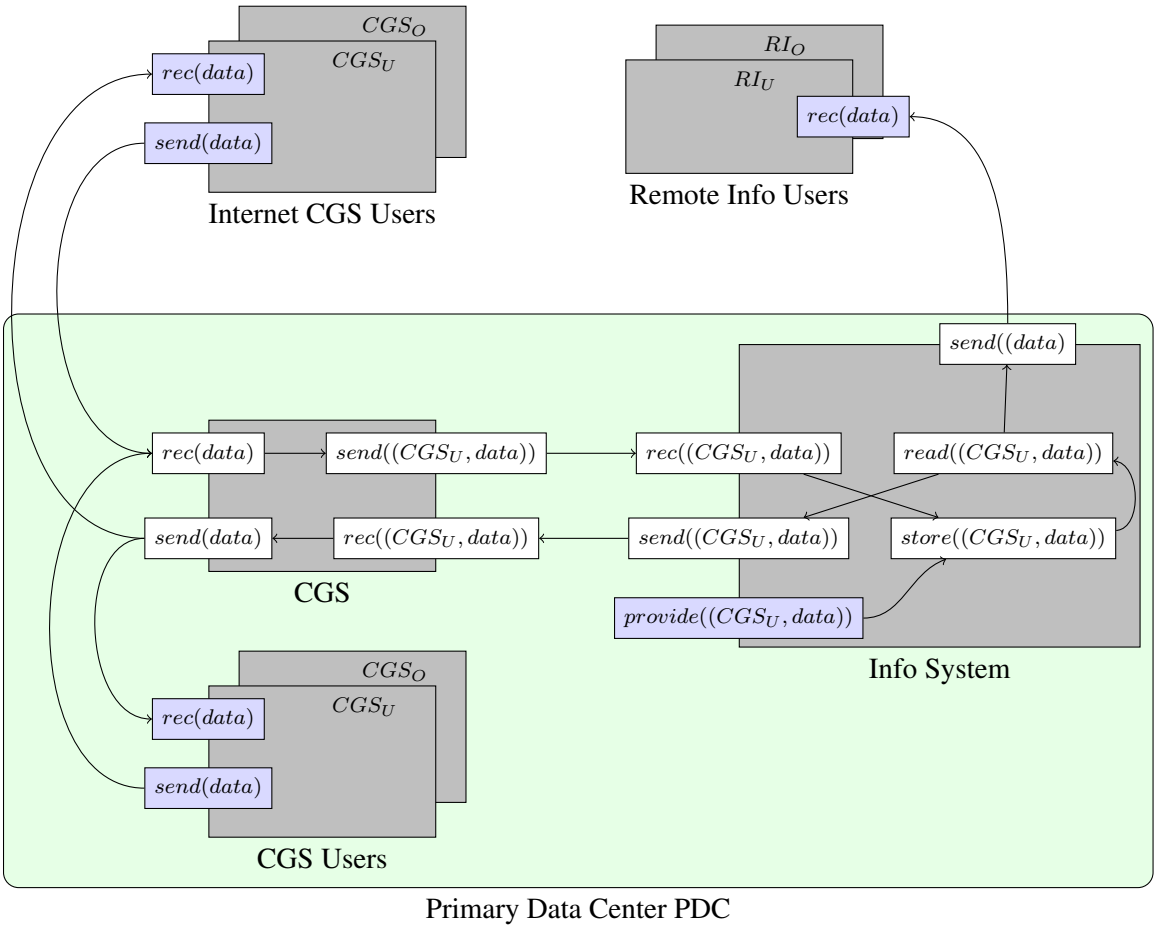

Figure 4.6: Data flow user data

$\mathcal{L}_{max}$ defines the language that allows malicious agents to know about all relations between occurrences of the confidential parameter d. Note that this language will differ for every property instance.

The next step would be to analyse monitoring capabilities to derive the necessary assumptions for the reasoning process. Further refinement of the coarse high level description of the Olympic Games scenario is necessary for this process. As a first step used assets from the use case should be mapped to the architecture. E.g. it is not possible to locate the central database system (GMS) in the architecture. In order to define suitable assumptions, monitoring would be necessary for:

- Database activity
- Identity monitoring
- Monitoring of user activity
- Monitoring of accreditation and workforce management

Sufficiency of Monitoring Capabilities
Assumptions have to be derived from monitoring capabilities for reasoning whether the given requirements are fulfilled under these assumptions. This reasoning process can't be successful if monitoring capabilities are insufficient or can't be assigned to entities used in the current abstraction level of the system model. The requirements cannot be measured by the given SIEM information sources.

5 Conclusion

In this deliverable we described a method to identify *abstract security requirements* from given high-level generic security goals. We showed that this formal representation of security requirements is necessary for different reasoning processes in MASSIF. We addressed questions like "Which security goals are not covered by security requirements ?", "Which information must the SIEM sensors provide in order to verify the given requirements ?", and "Which requirements cannot be measured by the given SIEM information sources ?" during configuration time, as well as runtime questions like "Which security requirements and associated security goals are broken by current security event measurements ?".

In order to support the answering of theses questions, we demonstrated, how a complete set of security requirements for a given high-level security goals can be elicited. Furthermore, we showed, how the derived security requirements from the analysed MASSIF scenarios can be formalised using the proposed property-based characterisation of security requirements in the security modelling framework SeMF. This is the first step in the reasoning chain to answer the above questions.

For runtime analysis, a backward chaining of the reasoning starting with the measured security information via the broken security requirement to the broken security goal is proposed. The missing steps in the reasoning chain, namely the derivation of measurement requirements from the abstract security requirements will be subject of a forthcoming deliverable.

The application of the proposed concepts was performed for several MASSIF use cases based on the respective scenario descriptions taken from D2.1.1 [12]. For these use cases the elicitation of monitoring rules and derivation of corresponding assumptions was sketched. This builds the base to support reasoning whether the combination of monitoring rules and assumptions fulfill these requirements.

The next steps will b to apply so called security building blocks in the monitoring rule elicitation process and reasoning. The dashed lines in Figure 5.1, which is a refined version of Figure 3.2, illustrate these steps of consecutive work. The main steps will be:

- Continuation in development of sketched approach for monitoring rule elicitation.
- Definition of building blocks for reasoning.
- Derivation of an executable model from the system model and domain description to enable predictive analysis.
- Transformation of monitoring rules to a representation usable for monitoring of predictive analysis results.

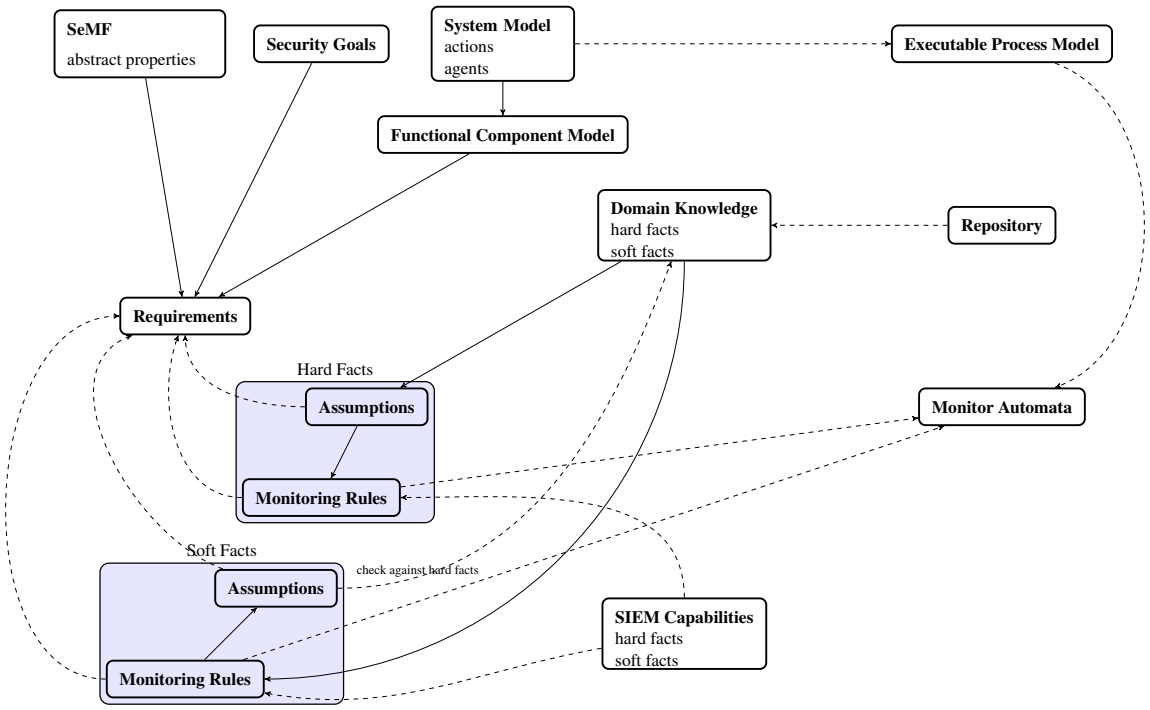

Figure 5.1: Workflow of the monitoring rule elicitation process

Bibliography

[1] Serenity, system engineering for security & dependability. www.serenity-project.org, 2006.

[2] Evita,e-safety vehicle intrusion protected applications. www.evita-project.org/, 2010.

[3] M. Abadi and M.R Tuttle. A Semantics for a Logic of Authentication. In *Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Canada*, pages 201–216, August 1991.

[4] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 7 October 1985.

[5] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8, 1990.

[6] Andreas Fuchs, Sigrid Gürgens, and Carsten Rudolph. A Formal Notion of Trust – Enabling Reasoning about Security Properties. In *Preceedings of Fourth IFIP WG 11.1 International Conference on Trust Management*, 2010.

[7] Andreas Fuchs and Roland Rieke. Identification of Security Requirements in Systems of Systems by Functional Security Analysis. In Antonio Casimiro, Rogério de Lemos, and Cristina Gacek, editors, *Architecting Dependable Systems VII*, volume 6420 of *Lecture Notes in Computer Science*, pages 74–96. Springer, 2010.

[8] R. Grimm and P. Ochsenschläger. Binding Cooperation, A Formal Model for Electronic Commerce. *Computer Networks*, 37:171–193, 2001.

[9] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Parameter confidentiality. In *Informatik 2003 - Teiltagung Sicherheit*. Gesellschaft für Informatik, 2003.

[10] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Abstractions preserving parameter confidentiality. In *European Symposium On Research in Computer Security (ESORICS 2005)*, pages 418–437, 2005.

[11] K. Dolinar, A. Fuchs, S. Gürgens, C. Rudolph. A3.D2.2 - S&D requirements for networks and devices. Technical report, SERENITY-Project, 2008.

[12] Marc Llanes, Elsa Prieto, Rodrigo Diaz, , Luigi Coppolino, Antonio Sergio, Rosario Cristaldi, Mohammed Achemlal, Said Gharout, Chrystel Gaber, Andrew Hutchison, and Keiran Dennie. Scenario requirements. Technical Report Deliverable D2.1.1, MASSIF Project, 2011.

[13] L.C. Paulson. Proving Properties of Security Protocols by Induction. Technical Report 409, Computer Laboratory, University of Cambridg, 1996.

[14] G. Wedel and V. Kessler. Formal Semantics for Authentication Logics. In *Computer Security - Esorics 96*, volume 1146 of *LNCS*, pages 219–241, 1996.

6 Appendix A

6.1 SeMF Property Definitions

Authenticity with Respect to a Phase

In many cases it is not only necessary to know *who* has performed a particular action, but also the specific *"time"* of the action is important. As our specification is discrete and does not model any real time properties, time is modeled in terms of relations between actions in a sequence. However, an explicit model of discrete time can be easily included in the model.

We use the definition of a $phase$ provided in [8]. A phase $V \subset \Sigma^*$ is a prefix closed language consisting only of words which, as long as they are not maximal in V, show the same continuation behaviour within V as within B. Maximal words in a phase are those that lead out of the phase, i.e. those $v \in V$ for which exists $\omega, u \in B$ with $\omega = uv$ such that for all $a \in \Sigma$ with $\omega a \in B$ holds $va \notin V$.

Definition 7 *Let $B \subseteq \Sigma^*$ be a system. A prefix closed language $V \subset \Sigma^*$ is a phase in B if the following holds:*

1. $V \cap \Sigma \neq \emptyset$
2. $\forall \omega \in B$ *with* $\omega = uv$ *and* $v \in V \setminus (max(V) \cup \{\varepsilon\})$ *holds:* $\omega^{-1}(B) \cap \Sigma = v^{-1}(V) \cap \Sigma$

Thus a phase as defined above is essentially a part of the system behaviour that is closed with respect to concatenation. In analogy to the maximal words of a phase we define the minimal words of a phase as all $v \in V$ with $|alph(v)| = 1$.

A phase can be a very complex construct. However, in many cases phases are of interest that can be defined by their starting and ending actions. Since an action can occur more than once in a word, it is not sufficient to identify the starting and terminating actions for determining where a particular phase starts and where it ends. The following definition takes this into account.

Definition 8 *Let $s_1, \ldots, s_k, t_1, \ldots, t_l \in \Sigma$ be actions. Then $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$ (with $j_1, \ldots, j_l \in \mathbb{N}$) defines a phase in B that starts with actions $s_1, \ldots, s_k$ and terminates with actions $t_1, \ldots, t_l$ in the following sense:*

- *For all $\omega \in B$ for which exists $s_j \in \{s_1, \ldots, s_k\}$ such that $\omega s_j \in B$ follows $s_j \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$ (i.e. s_j is minimal in $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$).*
- *For all $\omega \in B$ for which exists $t_i(j_i) \in \{t_1(j_1), \ldots, t_l(j_l)\}$ and $u, v \in \Sigma^*$ with $\omega = uvt_i$, $vt_i \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$, and $card(t_i, vt_i) = j_i$ follows that vt_i is maximal in $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$.*

In the above definition, the starting action(s) need to be fixed so that starting from these the terminating actions occuring in the phase can be counted in order to identify those ones that actually terminate the phase. In the following definition, we conversely fix the termination action(s) and count the number of occurences of the starting action(s) backwards in the phase to identify the actual start(s) of the phase:

Definition 9 *Let $s_1, \ldots, s_k, t_1, \ldots, t_l \in \Sigma$ be actions. Then $V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\}) \subseteq B$ (with $i_1, \ldots, i_l \in \mathbb{N}$) defines a phase in B that starts with actions $s_1, \ldots, s_k$ and terminates with actions $t_1, \ldots, t_l$ in the following sense:*

- *For all $\omega \in B$ for which exists $s_j(i_j) \in \{s_1(i_1), \ldots, s_k(i_k)\}$ such that $\omega s_j \in B$ follows $s_j \in V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$ (i.e. s_j is minimal in $V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$) and $card(s_j, v) = i_j$ for all maximal words $v \in V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$.*
- *For all $\omega \in B$ for which exists $t_i \in \{t_1, \ldots, t_l\}$ and $u, v \in \Sigma^*$ with $\omega = uvt_i$, $vt_i \in V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$, follows that vt_i is maximal in $V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$.*

In some cases we are not interested in how often each of the ending actions occurs within the phase but we want to fix the number of occurences of any of them.

Definition 10 *Let $s_1, \ldots, s_k, t_1, \ldots, t_l \in \Sigma$ be actions. Then $V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j)) \subseteq B$ (with $j \in \mathbb{N}$) defines a phase in B that starts with actions $s_1, \ldots, s_k$ and terminates with actions $t_1, \ldots, t_l$ in the following sense:*

- *For all $\omega \in B$ for which exists $s_j \in \{s_1, \ldots, s_k\}$ such that $\omega s_j \in B$ follows $s_j \in V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j))$ (i.e. s_j is minimal in $V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j))$).*
- *For all $\omega \in B$ for which exists $t_i \in \{t_1, \ldots, t_l\}$ and $u, v \in \Sigma^*$ with $\omega = uvt_i$, $vt_i \in V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j))$, and $card(\{t_1, \ldots, t_l\}, vt_i) = j$ follows that vt_i is maximal in $V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j))$.*

While in many cases we are able to identify the last action(s) of a phase, in some cases we may know only the first action(s) that occur outside the phase. The next definition allows to specify a phase using these actions.

Definition 11 *Let $s_1, \ldots, s_k, t_1, \ldots, t_l \in \Sigma$ be actions, and let $p_1, \ldots, p_l$ be parameters with values $\in \{\text{in}, \text{ex}\}$. Then $V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\}) \subseteq B$ (with $j_1, \ldots, j_l \in \mathbb{N}$) defines a phase in B that starts with actions $s_1, \ldots, s_k$ and terminates with actions $t_1, \ldots, t_l$ in the following sense:*

- *For all $\omega \in B$ for which exists $s_j \in \{s_1, \ldots, s_k\}$ such that $\omega s_j \in B$ follows $s_j \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\})$ (i.e. s_j is minimal in $V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\})$.*
- *For all $\omega \in B$ for which exists $t_i(j_i, in) \in \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\}$ and $u, v \in \Sigma^*$ with $\omega = uvt_i$, $vt_i \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\})$, and $card(t_i, vt_i) = j_i$ follows that vt_i is maximal in $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$.*
- *For all $\omega \in B$ for which exists $t_i(j_i, ex) \in \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\}$ and $u, v \in \Sigma^*$ with $\omega = uvt_i$, $v \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\})$, and $card(t_i, v) = j_i$ follows that v is maximal in $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$.*

In other words, if the number of occurences of a terminating action is accompanied by the parameter in, the action is part of the phase. If it is accompanied by ex, it is the first action after the phase's termination.

Definition 12 (Authenticity with respect to a phase) *A set of actions $\Gamma \subseteq \Sigma$ is authentic for agent $P \in \mathbb{P}$ after a sequence of actions $x \in B$ with respect to W_P and a phase V if it is authentic for P after x and for all $y \in \lambda_P^{-1}(\lambda_P(x)) \cap W_P)$ exists $u, v, w \in \Sigma^*$ such that $y = uvw$ and $v \in V$ and $alph(v) \cap \Gamma \neq \emptyset$. Γ is currently authentic for P after x if $w = \varepsilon$.*

The above definition can be used to specify the security property provided by e.g. an SSL channel: data origin authenticity and freshness. The concept of a phase is used to model the duration of the SSL channel: The phase starts with the establishment of the channel by receiving the last handshake message (from the server's point of view), and ends by terminating the SSL channel (e.g. by deleting the respective session key). Each message exchanged on the channel is both performed within the phase represented by the channel and authentically generated by the server and the client, respectively.

Proof of Authenticity

Some actions do not only require authenticity but also need to provide a proof of authenticity (non-repudiation of receipt etc.). Usually, some evidence of the occurrence of a particular action is provided as "proof". Different types of proofs are possible: transferable, non-transferable, proofs that can get lost, etc. The following describes transferable proofs with the additional assumption that one cannot lose the proofs. Other requirements can be defined in a similar way.

If agent Q owns a proof of authenticity for a set Γ of actions we assume it can send this proof to other agents, which in turn can receive the proof and be convinced of Γ's authenticity. In the following definition the set ΓP denotes actions that provide agents with proofs about the authenticity of Γ. An agent Q can use the forwarding actions from ΓS in order to present the proof to other agents, that in turn can use the actions from ΓP to validate the authenticity of Γ.

Definition 13 (Proof of authenticity) *A pair $(\Gamma S, \Gamma P)$ with $\Gamma S \subseteq \Sigma$ and $\Gamma P \subseteq \Sigma$ is a pair of sets of proof actions of authenticity for a set $\Gamma \subseteq \Sigma$ on B with respect to $(W_Q)_{Q \in \mathbb{P}}$ if for all $\omega \in B$ and for all $Q \in \mathbb{P}$ with $alph(\pi_Q(\omega)) \cap \Gamma P \neq \emptyset$ the following holds:*

1. *For Q the set Γ is authentic after ω and*
2. *for each $R \in \mathbb{P}$ there exist actions $a \in \Sigma_{/Q} \cap \Gamma S$ and $b \in \Sigma_{/R} \cap \Gamma P$ with $\omega ab \in B$.*

Agent $Q \in \mathbb{P}$ can give proof of authenticity of $\Gamma \subseteq \Sigma$ after a sequence of actions $\omega \in B$ if 1 and 2 hold.

In addition to agent Q and set of actions Γ we have to specify the set of actions after which proof of authenticity for Q shall hold.

Note that we do not specify actions for forwarding of proofs and receiving of forwarded proofs in our example because such actions might happen outside the specified system. Thus these actions are not explicitly included in the properties described below. However, in order to prove that these properties hold, the forwarding and receiving of proofs can easily be added.

Parameter Confidentiality

Parameter confidentiality essentially captures the following: Consider agent R has monitored a sequence of actions ω, and in some of these actions a parameter p occurs with a specific value. Then R must not be able to distinguish this sequence from any other sequence in which the parameter occurs with different values, even if knowing all possible values.

Consider as an example a system consisting of three active nodes and an end user. $Sensor_1$ and $Sensor_2$ are nodes deployed somewhere in the system. They perform measurements (e.g. measure the water level in the dam scenario) and send the resulting data over the network. MS is the third node of the system. It receives data from the network and displays them to the end user *User* (e.g. the operator of the monitoring station MS) . Let us assume that the communication channel between the monitoring stataion and the sensors is secured using a digital signature scheme. Hence we may want to require that a sensor's signature key shall be confidential to all other agents. Now assume that $Sensor_1$ generates and sends a signature and MS receives, verifies and accepts this signature, modeled for example by $send(Sensor_1, data, privK_1, sig_j(data))\ recv(MS, data, pubK_1, sig_j(data))$. When monitoring this sequence, $Sensor_2$ and the display shall not be able to deduce the actual private key that was used even if knowing the key's length and thus all possible values. This property can be expressed with our notion of parameter confidentiality and we will explain it further using the requirement of $privK_1$ to be confidential to the display.

Various aspects are included in this definition. First, one has to consider MS's view of the sequence ω it has monitored and thus the set $\lambda_{MS}^{-1}(\lambda_{MS}(\omega))$ of sequences that are, from MS's view, identical. We assume a network bus connection which allows all agents to see their own actions and the data and signature of actions performed by other agents, but not the sender and the signature key used. The display's local view λ_{MS} of the sequence above is then $\lambda_{MS}(send(Sensor_1, data, privK_1, sig_j(data))\ recv(MS, data, pubK_1, sig_j(data))) = send(data, sig_j)\ recv(MS, data, pubK_1, sig_j(data))$. The set $\lambda_{MS}^{-1}(\lambda_{MS}(send(Sensor_1,\ da\ ta,\ privK_1, sig_j(data))\ recv(MS, data, pubK_1, sig_j(data))))$ of the sequences of actions that are from the display's view identical consists of sequences $send(P, data, privK_i,\ sig_j(data'))\ \ recv(MS, data, pubK_1,\ sig_j(data))$ with $privK_i$ denoting possible key values, P denoting any possible sender, and $data, data'$ denoting possible values of the parameter $data$.

Second, MS can discard some of the sequences from this set, depending on its knowledge of the system and the system assumptions, all formalised in W_{MS}. There may for example exist interdependencies between parameters in different actions, such as the public key used for a signature's verification determining the private key used for its generation. In consequence, MS considers only those sequences of actions possible in which its own receive action, including verification and acceptance of the signature, is always preceded by a signature generation and send action which uses the matching signature key and data. So the set of sequences MS considers to have possibly happened after ω has happened is reduced to $\lambda_{MS}^{-1}(\lambda_{MS}(\omega)) \cap W_{MS}$. In the example this set consists of sequences $send(P, data, privK_i, sig_j(data))\ recv(MS, data, pubK_1, sig_j(data))$, with MS knowing that there is a relation between the public key $pubK_1$ used for verification of the signature and $privK_i$ but not being able to deduce the actual value of $privK_i$.

Third, those actions have to be identified in which the respective parameter(s) shall be confidential, or in other words, the actions from which a monitoring agent can possibly learn the parameter's value. Usually many actions are independent from these and do not influence confidentiality, thus need not be considered. In the example, it is the send action that contains $privK_i$ and from which MS can learn its value but the sensing and display actions are not relevant regarding the confidentiality of $Sensor_1$'s

signature key. We formalise this by using a homomorphism μ that maps all actions of interest for the particular confidentiality property onto what we call the *action type* while at the same time extracting the parameter to be confidential, and that maps all other actions onto the empty word. In the example, we define $\mu(send(Sensor_i, data, privK_i, sig_j(data))) = (send(Sensor_i, data, sig_j), privK_i)$ and $\mu(action) = \varepsilon$ for all other actions $action$.

Essentially, parameter confidentiality is captured by requiring that for actions that shall be confidential for an agent with respect to some parameter p, all possible (combinations of) values for p occur in this set. What are the possible combinations of parameters is the fourth aspect that needs to be specified, as one may want to allow the agent to know some of the interdependencies between parameters (e.g. it may be allowed to know the relation between signature and verification key). The notion of (L, M)–Completeness captures which are the allowed dependencies within a set of sequences of actions.

We define L to be a formal language that consists of sequences of pairs *(action type,number)* where *action type* are the types of actions that are kept by μ and the numbers are used to identify those actions whose relation between the parameters is allowed to be known. We extend the example and add a further send action by $Sensor_1$, resulting in
$\omega = send(Sensor_1, data', privK_1, sig_i(data'))\ send(Sensor_1, data, privK_1, sig_j(data))$
$recv(Displ,\ data', pubK_1, sig_i(data'))\ \ recv(Displ, data, pubK_1, sig_j(data))$. Now MS owns the sensor's public key and thus knows (and is allowed to know) that $Sensor_1$ always uses the same private key. On the other hand, $Sensor_2$ does not own $pubK_1$, and if we assume that it does not know that $Sensor_1$ always uses the same key, even if $data$ indicates who is the message's sender, $Sensor_2$ should not be able to distinguish one send action from another. Hence for describing the confidentiality requirement of the display we assign all send actions the same number, hence L_{MS} contains sequences $(send(Sensor_1, data, sig_i), k)(send(Sensor_1, data, sig_j), k)$. Addressing the confidentiality requirement of $Sensor_2$ we assign different numbers to the send actions which results in a different language L_{Sensor_2} that contains sequences such as $(send(Sensor_x, data, sig_i), k)(send(Sensor_x, data, sig_j), l)$. Then functions f are used to map these numbers to the set M of possible parameter values. The resulting set of action sequences contains all possible combinations of parameter values with the constraint that actions related with respect to the the parameter contain the same parameter value. Such a set of action sequences is called (L, M)–complete.

For the formal definition of (L, M)–completeness, some additional notations are needed: For
$f : M \longrightarrow M'$ and $g : N \longrightarrow N'$ we define
$(f, g) : M \times N \longrightarrow M' \times N'$ by $(f, g)(x, y) := (f(x), g(y))$. The identity on M is denoted by $i_M : M \longrightarrow M$, while $M^{\mathbb{N}}$ denotes the set of all mappings from $\mathbb{N}$ to M, and $p_1 : (\Sigma_t \times M) \longrightarrow \Sigma_t$ is a mapping that removes the parameters.

Definition 14 *Let $L \subseteq (\Sigma_t \times \mathbb{N})^*$ and let M be a set of parameters. A language $K \subseteq (\Sigma_t \times M)^*$ is called (L, M)–complete if*

$$K = \bigcup_{f \in M^{\mathbb{N}}} (i_{\Sigma_t}, f)(L)$$

The definition of parameter confidentiality captures all the different aspects described above:

Definition 15 (Parameter Confidentiality) *Let M be a parameter set, Σ a set of actions, Σ_t a set of types, $\mu : \Sigma^* \to (\Sigma_t \times M)^*$ a homomorphism, and $L \subseteq (\Sigma_t \times \mathbb{N})^*$. Then M is parameter-confidential for agent $R \in \mathbb{P}$ with respect to (L, M)-completeness if there exists an (L, M)–complete language $K \subseteq (\Sigma_t \times M)^*$ with $K \supseteq \mu(W_R)$ such that for each $\omega \in B$ holds*

$$\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) \supseteq p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R))) \cap K$$

Here $p_1^{-1} \circ p_1$ first removes and then adds again the parameters that shall be confidential, i.e. constructs all possible value combinations. (L, M)–completeness of K captures exactly that we require R to consider all combinations of parameter values possible except for those that we allow to be disregarded. Hence the right hand side of the inequality constitutes all sequences of actions we require R to consider to have possibly happened after ω has happened, while the left hand side constitutes those sequences R actually does consider to have possibly happened. For further explanations we refer the reader to [9, 10].

Enforcing system behaviour

Some security requirements are concerned with enforcing specific system behaviour, i.e. with not allowing certain other system behaviour. Requiring authenticity of action a whenever action b has happened is one particular instantiation of enforcing system behaviour. However, other required behaviour needs a more general definition.

Definition 16 (enforce-behaviour) *For an alphabet Σ, let $L \subseteq \Sigma^*$ describe particular requirements and $B \subseteq \Sigma^*$ be the actual system.*
We say that L enforces its behaviour on B, denoted by enforce-behaviour(L,B)*, if $B \subseteq L$.*

Trust

Trust assumptions play an important role for the verification of system properties. When for example using a PKI we need to trust in the authenticity of the Trusted Third Party's public key when verifying certificates. When using trusted computing technology we need to trust that the TPM indeed provides certain security properties, etc. Hence our Security Modelling Framework includes the concept of trust. Within SeMF, we will use the following trust concept:

The term ***trust*** refers to a relation from one agent in the system to another agent with respect to a property, or from an agent directly to a property in the system. Thus, agents can have three slightly different types of trust:

1. Agents can trust that some (global) property holds in a system.
2. Agents can trust that another agent behaves in a certain way, i.e. that a property concerning the behaviour of this other agent is satisfied.
3. Agents can trust that another agent has a particular trust.

Being a relation, this notion of trust cannot be used to express different degrees of trust. Agents can either trust or not trust. See [6] for a more detailed discussion on our trust concept.

In order to provide the formal definition, we first introduce the definition of a system and a trusted system, both using the concepts introduced in Section 2.1.2.

Definition 17 (System) *A system $S = (\Sigma, \mathbb{P}, B, \mathbb{W}, \mathbb{V})$ consists of a set $\mathbb{P}$ of agents acting in the system, a language $B \subseteq \Sigma^*$ over an alphabet of actions Σ describing the system behaviour in terms of sequences of actions, a set $\mathbb{V} = \{\lambda_X : \Sigma^* \to (\Sigma_X)^* | X \in \mathbb{P}\}$ of agents' local views, and a set $\mathbb{W} = \{W_X \subseteq \Sigma^* | X \in \mathbb{P}\}$ of agents' initial knowledges.*

Here $(\Sigma_X)^*$ denotes the image of the homomorphism λ_X which has to be individually specified for each system. Which part of an action an agent can see depends on the specific system to specify and can contain any part of it, as indicated in the previous section.

An agent P's conception and understanding of a system S, denoted by S_P, may differ from the actual system. P may not know all about the system's behaviour, thus from P's point of view the system's behaviour consists of P's initial knowledge W_P. Further, P may not have all information with respect to the other agents' initial knowledges and local views, so P's conception of agents' initial knowledges (W_{XP}) and local views (λ_{XP}) may differ from the actual initial knowledges and local views of the system S. This motivates the following definition.

Definition 18 (Trusted System) *Agent P's conception of system S is defined by $S_P = (\Sigma, \mathbb{P}, W_P, \mathbb{W}_P, \mathbb{V}_P)$. Σ and $\mathbb{P}$ are the alphabet and set of agents, respectively, of both S and S_P, whereas P's initial knowledge (conception) $W_P \subseteq \Sigma^*$ of system behaviour B constitutes the behaviour of S_P. It further contains a set $\mathbb{V}_P = \{\lambda_{XP} : \Sigma^* \to (\Sigma_{XP})^* | X \in \mathbb{P}\}$ of agent P's conception of agents' local views of S, and a set $\mathbb{W}_P = \{W_{XP} \subseteq \Sigma^* | X \in \mathbb{P}\}$ of agent P's conception of agents' initial knowledges in S. We say that P* trusts *in system S_P (since it represents P's knowledge about system S).*

The definition of an agent's trusted system gives rise now to the definition of an agent's *trust in a property* holding in a system:

Definition 19 (Trusted Property) *Let* prop *be any property that refers to a system as defined in Definition 17. An agent $P \in \mathbb{P}$ trusts in* prop *to hold in a system S, denoted by* trust(P, prop)*, iff* prop *is fulfilled in S_P.*

This notion of trust follows naturally from the different aspects that constitute the model of a system. If a property holds in the system as P perceives it (i.e. in S_P), then from P's point of view the property holds, i.e. P trusts in the property to hold in S. Further the notion of trust allows to specify precisely what it is an agent trusts in. An agent may have trust in one property but not in another. Of course, trust itself is a property of a system as well. Therefore the trust concept allows to model arbitrarily long trust chains such that e.g. the trust of an agent in another agent's trust in a property can be expressed.