



A3.D2.1 - S&D requirements for networks and devices

S. Gürgens, C. Rudolph, J. Porekar,, S. Holtmanns, R. Bonato

Document Number	A3.D2.1
Document Title	A3.D2.1 - S&D requirements for networks and devices
Version	V1.0
Status	final
Work Package	WP3.2
Deliverable Type	Report
Contractual Date of Delivery	30 September 2006
Actual Date of Delivery	26 October 2006
Responsible Unit	SET
Contributors	SIT, SET, NOK, DBL
Keyword List	
Dissemination level	PU

Executive summary

This report specifies a draft version of a specification language for the accurate description of S&D requirements for networks and devices. It also includes a first set of S&D requirements specified in this language.

The main goal of this work is to provide the means to formulate exact specifications of security and dependability requirements. As necessary foundations to achieve this goal a formal framework has been developed that provides formal semantics for many S&D requirements. General S&D requirements covered by this framework include authenticity, proof of authenticity (different flavours of non-repudiation), authenticity with respect to a phase (session management, etc), confidentiality, integrity, enforcement of particular behaviours, availability, reliability, and maintainability.

In order to facilitate the use of the framework, a language for S&D requirements specification is defined that provides simplified access to the (rather complex) formal S&D requirements specifications. This language can be applied to a model of a system in terms of sequences of actions. Such a model can be derived from other standard description languages (e.g. message sequence charts and action diagrams).

Two scenarios have been chosen to motivate the work and demonstrate the application of the S&D requirements specification language for networks and devices. The scenarios show different aspects of the S&D requirements in AmI systems:

- The communication scenario contains a small number of mobile devices. On these devices a shared application/service is started (online game) after a trust relationship has been established. The main topic with regard to networks and devices is the reaction to the change of the communication interfaces.
- In the smart items scenario the focus is on the heterogenous character of AmI systems combined with the particularly strong S&D requirements of eHealth applications.

Contents

1	Introduction	4
2	Formal specifications of S&D requirements	4
2.1	Preliminaries	4
2.1.1	<i>System behaviour specification and agents' knowledge about a system</i>	4
2.1.2	<i>Agents' knowledge about the global system behaviour</i>	5
2.1.3	<i>Agents' view of the global system behaviour</i>	6
2.2	Security requirements	7
2.2.1	<i>Authenticity</i>	7
2.2.2	<i>Proof of authenticity</i>	7
2.2.3	<i>Authenticity with respect to a phase</i>	8
2.2.4	<i>A remark to integrity</i>	9
2.2.5	<i>Confidentiality</i>	9
2.2.6	<i>Enforcing certain system behaviour</i>	16
2.3	Dependability requirements	16
2.3.1	<i>Modelling a system for the specification of dependability requirements</i>	17
2.3.2	<i>Availability</i>	17
2.3.3	<i>Reliability</i>	18
2.3.4	<i>Maintainability</i>	18
3	A language for the specification of S&D requirements for networks and devices	19
3.1	Security requirements	19
3.2	Dependability requirements	23
4	Guidelines for requirements elicitation	23
4.1	Initial questions for requirements elicitation	23
5	Examples based on SERENITY scenarios	24
5.1	Example: Communication scenario	24
5.1.1	<i>Agents</i>	24
5.1.2	<i>Some "state" parameters and preliminary assumptions</i>	24
5.1.3	<i>Actions (example sequence)</i>	25
5.1.4	<i>Security requirements</i>	26
5.2	Example: Smart Items scenario	28
5.2.1	<i>Agents</i>	28
5.2.2	<i>Actions</i>	29
5.2.3	<i>Security requirements</i>	31
	References	36

1 Introduction

Devices and networks constitute the basic building blocks of modern IT systems. Confidential data can be stored, processed and transmitted, particular actions might require authenticity, devices have to be identified, etc.; a large variety of security and dependability (S&D) requirements can be directly specified on the level of devices and networks. The variety is particularly high in the context of ambient intelligence (AmI) systems where different types of devices exist and these devices can move from trusted to untrusted environments while using different communication links. Furthermore, complex S&D requirements specified on the level of business processes or work-flows can induce S&D requirements on devices and network communication. This report describes a language for the accurate specification of S&D requirements for networks and devices, taking into account the particular requirements of AmI systems.

The report is structured as follows: First, a formal approach to system behaviour specification is described which is then used to define the semantic foundations of the S&D requirements. General notions of security and dependability properties are defined within this specification framework. These notions are subsequently used to define the semantics for the S&D requirements language. Section 3 introduces the language for S&D requirements specification. Guidelines for the use of this language are given in Section 4. Finally, two SERENITY scenarios are used to demonstrate the use of the language.

2 Formal specifications of S&D requirements

2.1 Preliminaries

2.1.1 System behaviour specification and agents' knowledge about a system

The *behaviour* S of a discrete system can be formally described by the set of its possible sequences of actions (traces). Therefore $S \subseteq \Sigma^*$ 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 $\text{pre}(u)$; $\varepsilon \in \text{pre}(u)$ holds for every word u . We denote the set of letters in a word u by $\text{alph}(u)$.

Formal languages which describe system behaviour have the characteristic that $\text{pre}(u) \subseteq S$ holds for every word $u \in S$. 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 $u \in S$ is formally expressed by the *left quotient* $u^{-1}(S) = \{y \in \Sigma^* \mid uy \in S\}$.

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' knowledge about the global system behaviour*

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 [2] and [5]). 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, we extend 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.

For any agent P its knowledge about the global system behaviour $W_P \subseteq \Sigma^*$ 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 S and the correct system behaviour by S_C , we assume $S_C \subseteq S \subseteq \Sigma^*$. We further assume $S \subseteq W_P$, i.e. every agent considers the system behaviour to be possible. This reflects the fact that an agent not having knowledge of any restrictions of the system considers all Σ^* to be possible. Security properties can now be defined relative to W_P . The relation between the system behaviour without malicious actions S_C , the system behaviour including malicious actions S , and W_P is graphically shown in Figure 1.

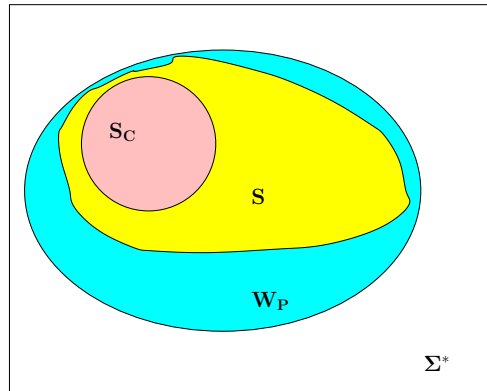


Figure 1: System behaviour and W_P

2.1.3 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 occurred. Satisfaction of security properties obviously also depends on what agents are able to learn. After a sequence of actions $\omega \in S$ has happened, every agent can use its *local view* of ω to determine the sequences of actions it considers to be possible. 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 $\pi_P : \Sigma^* \rightarrow \Sigma^*_{/P}$ defined by $\pi_P(x) = x$ if $x \in \Sigma_{/P}$ and $\pi_P(x) = \varepsilon$ if $x \in \Sigma \setminus \Sigma_{/P}$ formalizes 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 other agents' actions without seeing the messages sent and received, respectively. In this case, P 's local view of ω will be equal to $(send, P, m1)(rec, Q)$. 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' state.

Thus, we generally denote the local view of an agent P on Σ by $\lambda_P : \Sigma^* \rightarrow \Sigma^*_P$. The local views of all agents together contain all information about the system behaviour S .

For a sequence of actions $\omega \in S$ 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 example with the projection on P being P 's local view, $\lambda_P^{-1}(\lambda_P(\omega))$ consists of sequences each of which contains an action $(send, P, (Q, m1))$. For some agent R that does not take part in ω , $\lambda_R^{-1}(\lambda_R(\omega))$ consists of sequences of actions of other agents, i.e. is equal to $(\Sigma \setminus \Sigma_{/R})^*$.

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 knowledge to reduce this set: $\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$ describes all sequences of actions P considers to be possible when ω has happened. The set $\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$ is similar to the possible worlds semantics that have been defined for authentication logics in the context of cryptographic protocols [1, 6]. 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.

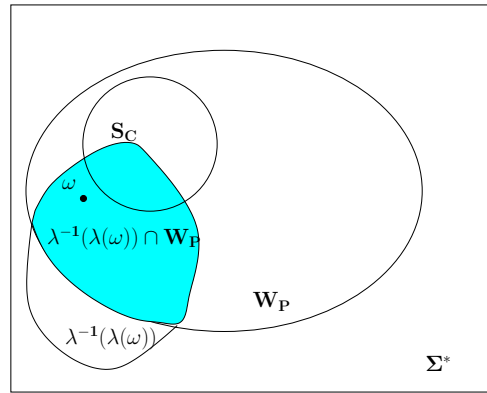


Figure 2: Local view of P

2.2 Security requirements

2.2.1 Authenticity

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

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 possible 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 $\omega \in S$ with respect to W_P if $\text{alph}(x) \cap \Gamma \neq \emptyset$ for all $x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$.

Input needed for specifying authenticity:

- The action after which authenticity shall hold.
- The agent for which authenticity shall hold.
- The action or set of actions that shall be authentic.

2.2.2 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

¹Please note that in order to achieve *authentication* one additionally needs assurance about the time of the occurrence of the action (see Section 2.2.3 *authenticity with respect to a phase*).

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 P 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 . If agent P has executed an action from ΓP then Γ is authentic for P and P can forward the proof to any other agent using actions in ΓS .

Definition 2 (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 S with respect to $(W_P)_{P \in \mathbb{P}}$ if for all $\omega \in S$ and for all $P \in \mathbb{P}$ with $\text{alph}(\pi_P(\omega)) \cap \Gamma P \neq \emptyset$ the following holds:

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

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

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

Input needed for specifying proof of authenticity

- The agent for which authenticity shall hold.
- The action or set of actions that shall be authentic.
- The action or set of actions after which the agent shall own a proof of authenticity.

2.2.3 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 [3]. 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 S .

Definition 3 Let $S \subseteq \Sigma^*$ be a system. A prefix closed language $V \subset \Sigma^*$ is a phase in S if the following holds:

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

A phase can be a very complex part of the system (see for example the the definition of a phase by Grimm and Ochsenschläger [3]). However, often phases have well defined start and end actions. Therefore, for our purposes it is sufficient to consider only those phases that start with one specific action and end with one specific action. In other words, an element of such a phase begins with the start action and contains either no end action at all or contains just one end action at the end.

Definition 4 (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 $x \in S$ 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 $\text{alph}(v) \cap \Gamma \neq \emptyset$. Γ is currently authentic for P after x if $w = \varepsilon$.*

Input needed for specification of authenticity with respect to a phase

- Action with which P starts the phase between P and Q ,
- Action with which P ends the phase between P and Q ,
- The action after which authenticity shall hold.
- The agent for which authenticity shall hold.
- The action or set of actions that shall be authentic.

2.2.4 A remark to integrity

Usually, integrity is viewed as a very important security requirement. It is certainly true that data integrity is a central security requirement. However, consider the following example. Data integrity can be provided by a checksum. Change of the data would be detected because the checksum is not correct any more. But without authenticity, i.e. without knowing who computed the checksum, the checksum mechanism is useless: A malicious agent may change the data and simply compute a new checksum. Integrity is satisfied (meaning that data has not been changed since the checksum has been computed) but the actual security requirement is not.

Thus in order to make sense, integrity of data requires some kind of anchor fixing the origin of the data that shall remain unchanged. When considering distributed systems, this anchor can only be provided by some authentic action in the past. Taking into account that integrity is implied by authenticity and that integrity without authenticity is valueless we have omitted a definition for integrity.

2.2.5 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. 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 [4] provides this flexibility.

Various aspects are included in this definition. First, we have to consider an attacker C 's view of the sequence ω it has monitored and thus the set of sequences $\lambda_C^{-1}(\lambda_C(\omega))$ that are, from C 's view, identical. Second, C can discard some of the sequences from this set, depending on its knowledge of the system and the system assumptions, all formalized in W_C . For example, there may exist interdependencies between parameters in different actions, such as a credit card number remaining

the same for a long time, in which case C considers only those sequences of actions possible in which an agent always uses the same credit card number.

So the set of sequences C considers possible after having monitored ω is reduced to $\lambda_C^{-1}(\lambda_C(\omega)) \cap W_C$. Third, we need to identify the actions in which the respective parameter(s) shall be confidential. Usually many actions are independent from these and do not influence confidentiality, thus need not be considered.

Essentially, parameter confidentiality is captured by requiring that for the actions that shall be confidential for agent C with respect to some parameter p , all possible (combinations of) values for p occur in the set of actions that an attacker considers possible. What are the possible combinations of parameters is the fourth aspect that needs to be specified, as we may want to allow C to know some of the interdependencies between parameters (e.g. C may be allowed to know that a message that was received must have been sent before).

The definition of parameter confidentiality as described in the following section captures all these different aspects.

Definition of the property We want to formalize the following property: An agent R that monitors a sequence of actions ω of a system S cannot distinguish between the possible values of a certain parameter (a certain part of the message, the agent performing the action, etc.) of a specific action or set of actions of the sequence, even if it knows the set of possible parameter values. Consider for example an application consisting of the following actions: a user requests a price for a certain service, the request is received by a service provider and then an offer for this service is sent and received. In this example, one critical parameter might be the price. The service provider might have different rates for different users and these rates can change. We assume the price is supposed to be confidential, i.e. no other agent shall be able to tell which price has been offered. In the remainder of this section the external agent (the attacker) is denoted R , the user U and the service provider SP . The actions in the system are $send-price-request(U,SP)$, $rec-price-request(SP,U)$, $send-offer(SP,U,price)$ and $rec-offer(U,SP,price)$. The first parameter denotes the agent executing the particular action. We assume that R can see all actions but not the parameters, i.e. R 's local view $\lambda_R(send-price-request(U,SP))$ for example is equal to $send-price-request(U,SP)$, while $\lambda_R(rec-offer(U,SP,price)) = rec-offer(U,SP)$.

In the sequences of actions that R considers possible after having observed ω , only the actions where a price is sent and received are of interest for the example. Thus we disregard all other actions, i.e. we map them with a suitably chosen homomorphism μ onto the empty word. From those actions not mapped onto ε , μ extracts the confidential parameter that occurs in the action. Generally not only the parameter itself but also the "type" of its occurrence has to be considered. This type can be, for example, that a certain user U has received an offer. The parameter associated with this type is the price included in the offer. By considering only the type, actions from Σ are divided into classes the elements of which can be distinguished essentially by the parameter values. Each of these classes is represented by one type.

Hence $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is a set of sequences of actions that consist of the types of those actions that are of interest with respect to parameter confidentiality, paired with the respective parameter values being possible from R 's local view.

If Σ_t denotes the set of types of the parameter occurrences and if M denotes the set of parameter values then $\mu_{\Sigma_t, M} : \Sigma^* \rightarrow (\Sigma_t \times M)^*$ is a homomorphism. For simplicity we write μ if the related parameter set and the types are obvious. Such a homomorphism μ can be defined as follows:

$$\begin{aligned} \mu(send-offer(SP, U, price)) &= (Send_{SP}, price) \\ \mu(rec-offer(U, SP, price)) &= (Rec_U, price) \\ \mu(send-price-request(U, SP)) &= \mu(rec-price-request(SP, U)) = \varepsilon \end{aligned}$$

In order to explain our formalism, we use the price offer system S . Our aim is now to formalize that $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ “contains all possible parameter values”.

(L, M) –Completeness For the following, we assume that R monitors all sequences of actions. Let us consider as an example the following sequence of actions:

$$\begin{aligned} \omega = & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_1) \text{rec-offer}(U, SP, price_1) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_2) \end{aligned}$$

Let us further assume that for $R \neq U, SP$ it shall be confidential which price was sent and received, respectively. Let $\{price_1, price_2\}$ be the set of possible prices, and Σ the set of resulting possible actions. As described above, R 's local view of this sequence is the following:

$$\begin{aligned} \lambda_R(\omega) = & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U) \text{rec-offer}(U, SP) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U) \end{aligned}$$

Now if R does not know which of the possible two parameters was sent, but does know that the same parameter that was sent was also received (specified in his initial knowledge W_R), R considers four different sequences of actions possible: two in which SP sends and U receives twice the same parameter (either $price_1$ or $price_2$), one in which first $price_1$ is sent and received and then $price_2$, and one in which the parameters are sent and received in reverse order:

$$\begin{aligned} \lambda_R^{-1}(\lambda_R(\omega)) \cap W_R = & \{ \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_1) \text{rec-offer}(U, SP, price_1) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_1) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_1) \text{rec-offer}(U, SP, price_1) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_2) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_2) \text{rec-offer}(U, SP, price_2) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_1) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_2) \text{rec-offer}(U, SP, price_2) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, price_2) \} \end{aligned}$$

The function μ now maps these sequences of actions onto sequences with types for the send and receive actions, each one being paired with the respective parameter. All other actions are mapped onto ε . This results in

$$\begin{aligned} \mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = & \{ (\text{Send}_{SP}, price_1) (\text{Rec}_U, price_1) (\text{Send}_{SP}, price_1), \\ & (\text{Send}_{SP}, price_1) (\text{Rec}_U, price_1) (\text{Send}_{SP}, price_2), \\ & (\text{Send}_{SP}, price_2) (\text{Rec}_U, price_2) (\text{Send}_{SP}, price_1), \\ & (\text{Send}_{SP}, price_2) (\text{Rec}_U, price_2) (\text{Send}_{SP}, price_2) \} \end{aligned}$$

If we want to describe a situation where R does not know any correlation between the parameter of a send and the respective receive action, the μ -image of the sequence of actions monitored by R contains eight different sequences of pairs (*type, parameter*) with no order on the parameters $price_1$

and $price_2$:

$$\begin{aligned} \mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = & \{(Send_{SP}, price_1)(Rec_U, price_1)(Send_{SP}, price_1), \\ & (Send_{SP}, price_1)(Rec_U, price_1)(Send_{SP}, price_2), \\ & (Send_{SP}, price_1)(Rec_U, price_2)(Send_{SP}, price_1), \\ & (Send_{SP}, price_2)(Rec_U, price_1)(Send_{SP}, price_1), \\ & (Send_{SP}, price_2)(Rec_U, price_1)(Send_{SP}, price_2), \\ & (Send_{SP}, price_2)(Rec_U, price_2)(Send_{SP}, price_1), \\ & (Send_{SP}, price_1)(Rec_U, price_2)(Send_{SP}, price_2), \\ & (Send_{SP}, price_2)(Rec_U, price_2)(Send_{SP}, price_2)\} \end{aligned}$$

In general we have the requirement that in each group of actions that R knows to be correlated, it considers each of the parameters possible. In order to formalize this, we assign each group a number. We then built the μ -image of the sequences of actions that R considers possible after ω has happened, with the parameters being substituted by the number of the respective group they belong to. Then we check that when mapping these numbers arbitrarily onto possible parameters, this results in the μ -image of R 's inverse view of ω , i.e. we check that the μ -image is (L, M) -complete for a specific language L (containing the action types associated with numbers) and parameter set M .

For the formal definition of (L, M) -completeness, we need some notations: For $f : M \rightarrow M'$ and $g : N \rightarrow N'$ we define $(f, g) : M \times N \rightarrow M' \times N'$ by $(f, g)(x, y) := (f(x), g(y))$. The identity on M is denoted by $i_M : M \rightarrow M$, while $M^{\mathbf{N}}$ denotes the set of all mappings from \mathbf{N} to M .

Definition 5 Let $L \subseteq (\Sigma_t \times \mathbf{N})^*$ and let M be a set of parameters. A language $K \subseteq (\Sigma_t \times M)^*$ is called (L, M) -complete if

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

In this definition, the set L consists of sequences of pairs (*action type, number*). The functions $f \in M^{\mathbf{N}}$ map the numbers to parameter values in M . Therefore, $(i_{\Sigma_t}, f)(L)$ consists of sequences of pairs (*action type, parameter value*). These sequences are in accordance with the correlations between parameter values defined by L . Now, K is (L, M) -complete if it consists of all possible sequences of pairs (*action type, parameter value*) derived by applying (i_{Σ_t}, f) to L for all possible mappings f from \mathbf{N} to M .

This property allows the formalization of any of the above described situations. Let us consider as an example again the sequence of actions ω .

For the set $\{Send_{SP}, Rec_U\}$ of relevant action types we now choose a numbering that assigns the same number to those actions that are correlated:

$$L_1 = \{(Send_{SP}, 1)(Rec_U, 1)(Send_{SP}, 2)\}$$

Having chosen the language L_1 in this manner and considering the set $M = \{price_1, price_2\}$ of parameter values, $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is (L_1, M) -complete if and only if

$$\begin{aligned} \mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = & \{(Send_{SP}, price_1)(Rec_U, price_1)(Send_{SP}, price_1), \\ & (Send_{SP}, price_1)(Rec_U, price_1)(Send_{SP}, price_2), \\ & (Send_{SP}, price_2)(Rec_U, price_2)(Send_{SP}, price_1), \\ & (Send_{SP}, price_2)(Rec_U, price_2)(Send_{SP}, price_2)\} \end{aligned}$$

This exactly describes the situation in which R knows that the same price was received that was sent, but does not know which of the prices was sent. Note that if R considers more parameter values

possible (i.e. if $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ contains more than the above four sequences), R still does not know which parameter was sent.

If R shall not know that there is a correlation between send and receive actions, the action types have to be numbered differently: no actions are correlated. This results in

$$L_2 = \{(Send_{SP}, 1)(Rec_U, 2)(Send_{SP}, 3)\}$$

The requirement of (L_2, M) -completeness results in the above mentioned eight sequences of pairs (*type, parameter*). However, if R knows the correlation between *Send* and *Rec*, i.e. if $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ contains only the four different sequences above (in which the same parameter value is sent and received), then $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is not (L_2, M) -complete: Using $f(1) = price_1$, $f(2) = price_2$ and any $f(3)$ we obtain

$$(Send_{SP}, price_1)(Rec_U, price_2)(Send_{SP}, price_1) \notin \mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$$

Thus by appropriately numbering the action types, i.e. by appropriately choosing the language L , we can formalize which correlations between actions R is allowed to know, or in other words, which sequences of actions have to be included in W_R . This gives rise to the following definition:

Let M be a parameter set, Σ a set of actions, Σ_t a set of types, $\mu : \Sigma^* \rightarrow (\Sigma_t \times M)^*$ a homomorphism, and $L \subseteq (\Sigma_t \times \mathbf{N})^*$. Then M is parameter-confidential for R after ω with respect to (L, M) -completeness if there exists an (L, M) -complete language $K \subseteq (\Sigma_t \times M)^*$ with $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) \supseteq K$.

Instead of separately defining different languages L for different sequences ω and the resulting set of sequences $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$, it is sufficient to have an appropriate L and an (L, M) -complete language $K \subseteq (\Sigma_t \times M)^*$ serving for all $\omega \in S$. Using the function p_1 that denotes the projection on the first component of a tuple, we introduce the following definition:

Definition 6 Let M be a parameter set, Σ a set of actions, Σ_t a set of types, $\mu : \Sigma^* \rightarrow (\Sigma_t \times M)^*$ a homomorphism, and $L \subseteq (\Sigma_t \times \mathbf{N})^*$. 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 $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) \supseteq p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R))) \cap K$ for each $\omega \in S$.

Applying the projection p_1 and then the inverse p_1^{-1} to $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ results in sequences of actions where all parameter values occur and no grouping according to correlated actions has yet taken place. The intersection with the (L, M) -complete language K removes those sequences that do not match the respective grouping. Note that in the case of no correlation between actions, $p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R))) \cap K = p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)))$.

More generally, the above equation can be used to define, for an arbitrary language $K \subseteq (\Sigma_t \times M)^*$ with $K \supseteq \mu(W_R)$, K -completeness of $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$. This allows to capture more sophisticated correlations between parameters.

A different property: M -rich In some cases there is no adequate language L to describe that R cannot recognize the respective parameters used. We introduce a new example to motivate a weaker confidentiality property and we show that this weaker property may not be adequate for our previous price example. Let us consider a system which models an auction. In this system we look at the bidding phase. For simplicity we assume there are only two bidders $U1$ and $U2$. The only possible action for bidders is *bid* with the parameters *bidder* and *amount*. In the same manner as above, agent R can monitor the bidding actions. We want to model the property that R may monitor the amount which a bidder has made but is not allowed to know which bidder has made which bid. In contrast to (L, M) -completeness of the price in the previous section, R is allowed to know which bids have

been made by the same bidder. For example, R may know that bids have been made alternately by two agents, but it is not allowed to know which bidder has started and which bidder has placed the winning bid. The homomorphism μ for this example can be defined as follows (for simplicity we disregard the values of the amount):

$$\mu(\text{bid}(U, \text{amount})) = (\text{Bid}, U)$$

Now we consider the following sequence of actions:

$$\delta = \text{bid}(U2, \text{amount}_1) \text{bid}(U1, \text{amount}_2) \text{bid}(U2, \text{amount}_3) \text{bid}(U1, \text{amount}_4)$$

After having monitored δ , the following sequences of parameter values are possible in R 's view of the system:

$$\begin{aligned} \lambda_R^{-1}(\lambda_R(\delta)) \cap W_R = \\ \text{bid}(U2, \text{amount}_1) \text{bid}(U1, \text{amount}_2) \text{bid}(U2, \text{amount}_3) \text{bid}(U1, \text{amount}_4), \\ \text{bid}(U1, \text{amount}_1) \text{bid}(U2, \text{amount}_2) \text{bid}(U1, \text{amount}_3) \text{bid}(U2, \text{amount}_4) \end{aligned}$$

The μ -image now extracts types and parameters that shall be confidential:

$$\begin{aligned} \mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R) = \{ & (\text{Bid}, U2)(\text{Bid}, U1)(\text{Bid}, U2)(\text{Bid}, U1), \\ & (\text{Bid}, U1)(\text{Bid}, U2)(\text{Bid}, U1)(\text{Bid}, U2) \} \end{aligned}$$

This knowledge of R corresponds to the confidentiality property that R cannot tell which bidder has placed which bid while knowing that $U1$ and $U2$ bid alternately. However, it is not possible to find a language L such that $\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)$ is (L, M) -complete for $M = \{U1, U2\}$. Considering the correlations between actions known to R the following language L_3 seems to be appropriate at first sight, as R knows that bidders place their bids alternately:

$$L_3 = \{(\text{Bid}, 1)(\text{Bid}, 2)(\text{Bid}, 1)(\text{Bid}, 2)\}$$

However, for the language L_3 chosen in this manner, $\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)$ is not (L_3, M) -complete as $f(1) = f(2) = U1$ results in

$$(\text{Bid}, U1)(\text{Bid}, U1)(\text{Bid}, U1)(\text{Bid}, U1) \notin \mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)$$

Nevertheless R does not know which of the two parameter values occurred: for each of the bids it considers both bidders possible. To formalize this situation we need a property that does not consider the complete possible sequences of actions from R 's local view but that considers only a "cut" through all sequences at the respective interesting actions. The following property describes the fact that from R 's local view at each separate point in the sequence of actions, each of the parameter values is possible:

$$\begin{aligned} \forall u \in p_1(\text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R))) : \\ p_2(\text{suf}_1(p_1^{-1}(u) \cap \text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)))) \supseteq M \end{aligned}$$

Starting with the sequence of actions δ , we use λ_R , λ_R^{-1} and W_R to generate the set of possible sequences of actions that are identical to δ in R 's local view and which R considers possible. From these we extract, using the function μ , the relevant types with the respective parameter values, and pre generates all possible prefixes (i.e. we cut off the last action, the second last, etc.). With p_1 we disregard the parameters having been extracted by μ . Every u in a set of sequences generated in this manner is a sequence of types that correspond to those actions in δ that we are interested in, without the respective parameter values. p_1^{-1} again adds all parameter values in all possible combinations. The intersection with $\text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R))$ disregards those sequences that R does not consider possible because of its knowledge about correlation of actions. From each of the resulting sequences, we consider only the last element by applying suf_1 (where $\text{suf}_i(\omega)$ returns the suffix of ω with length i). p_2 then determines those parameter values that R considers possible in the respective action. The resulting set must include the complete set M of parameter values.

For the above example we get

$$p_1(\text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R))) = \{\varepsilon, \text{Bid}, \text{BidBid}, \text{BidBidBid}, \text{BidBidBidBid}\}$$

$u = \text{BidBid}$ for example results in

$$p_1^{-1}(u) = \{(Bid, U1)(Bid, U1), (Bid, U1)(Bid, U2), (Bid, U2)(Bid, U1), \\ (Bid, U2)(Bid, U2)\}$$

We now intersect this set with the set of sequences of types (with parameters) that R considers possible as described above.

$$p_1^{-1}(u) \cap pre(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)) = \{(Bid, U1)(Bid, U2), (Bid, U2)(Bid, U1)\}.$$

su_{f_1} reduces the resulting *(type, parameter)* sequences to the respective last *(type, parameter)* (that is, to (Bid, U_i)), and finally p_2 extracts the parameters that R considers possible in this *(type, parameter)*. If this set does not include M completely then R knows more about the parameters that are possible in this action than it should know after having monitored δ . In our example, $p_2(su_{f_1}(\{(Bid, U1)(Bid, U2), (Bid, U2)(Bid, U1)\})) = \{U1, U2\} = M$.

Analogously to definition 5 we give the following general definition:

Definition 7 For a given set M of parameter values and a set Σ_t of action types we call the language $K \subseteq (\Sigma_t \times M)^*$ M -rich if

$$\forall u \in p_1(pre(K)) : p_2(su_{f_1}(p_1^{-1}(u) \cap pre(K))) \supseteq M$$

Analogously to (L, M) -completeness, we can now define a different kind of parameter confidentiality:

Definition 8 Let M be a parameter set, Σ a set of actions, Σ_t a set of types, and $\mu : \Sigma^* \rightarrow (\Sigma_t \times M)^*$ a homomorphism. Then M is parameter-confidential for R with respect to M -richness if $\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)$ is M -rich for all $\omega \in S$.

For the price-offer example explained in Section 2.2.5, the property M -rich may be too weak. Consider for example a case with only two possible prices and SP offering alternately the two different prices to U . Now, if $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is M -rich, R cannot tell which price has been offered in which action. However, as R can observe that two prices have been offered alternately, R can calculate the average price offered to U , which may be undesirable.

For a given *type* sequence, (L, M) -completeness of a language $K \subseteq (\Sigma_t \times M)^*$ exactly determines the set of *(type, parameter)* sequences which have to be in K . This is not true in the case of M -richness: As mentioned above, the equation

$$\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = \{(Send_P, m1)(Rec_Q, m1)(Send_P, m2), \\ (Send_P, m2)(Rec_Q, m2)(Send_P, m1)\}$$

implies M -richness of $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$. But also the equation

$$\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = \{(Send_P, m1)(Rec_Q, m1)(Send_P, m1), \\ (Send_P, m2)(Rec_Q, m2)(Send_P, m2)\}$$

would imply M -richness of $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$. So two different languages K express the required parameter variety.

Therefore, to bridge the formal gap between (L, M) -completeness and M -richness we need to consider the family \mathcal{K} of all languages $K \subseteq (\Sigma_t \times M)^*$ which are M -rich. Now $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is M -rich if and only if $\exists K \in \mathcal{K} : \mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R))) \cap K$.

Specifying parameter-confidentiality In order to specify parameter-confidentiality, the following information is needed:

α : the action(s) for which some parameter(s) shall be confidential,

$par(\alpha)$: the parameter(s) that shall be confidential (some parameters of the action(s) may not require confidentiality and may be known by other agents),

\mathcal{P} : the set of possible values for the parameter(s),

\mathcal{Q} : a set of agents that is allowed to know the parameter value(s)

L: optional, a language that describes the interdependencies between parameter values.

2.2.6 Enforcing certain 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 9 (enforce-behaviour) For an alphabet Σ , let $L \subseteq \Sigma^*$ describe particular requirements and $S \subseteq \Sigma^*$ be the actual system. We say that L enforces its behaviour on S , denoted by $\text{enforce-behaviour}(L,S)$, if $S \subseteq L$.

This property is particularly useful to describe work-flows where certain sequences of actions have to occur. See Section 3.1 for instantiations of this property.

2.3 Dependability requirements

The term *dependability* has many different interpretation. In the context of IT systems, dependability requirements can include the following attributes:

- Availability (readiness for correct service),
- Reliability (continuity of correct service),
- Maintainability (restore a service within a given time after failure)
- Safety (absence of catastrophic consequences on the user(s) and the environment), and
- Security.

The notion of security used in the SERENITY project is much broader than what is usually subsumed under dependability. Therefore, security and dependability issues are separated in SERENITY. Security requirements are discussed in the previous section.

Safety in the sense of the absence of catastrophic consequences will not be considered on the level of security solutions for network and devices in SERENITY. It is assumed that the hardware of the devices is safe to use. One example of safety issues for AmI devices is the problem of defective rechargeable battery packs. For mobile phones as well as for laptops cases of exploding or burning battery packs are known. The focus of the work on networks and devices in SERENITY is not on such physical safety issues. Safety-related side effects of an application will also not be covered by the solutions on the network and devices level. It has to be assumed that such side effects are either known to the designer of AmI applications or have been identified on the levels of business processes and workflows and can therefore be prevented by more refined concrete security, availability or reliability requirements.

Thus, dependability issues of devices and networks in SERENITY concentrate on availability, reliability and maintainability.

2.3.1 Modelling a system for the specification of dependability requirements

Many dependability properties are concerned with the duration (time) or probability of particular actions or sequences of actions. Thus, in order to be able to reason about dependability, the system specification as presented in Section 2.1 needs to include information about time and probabilities.

Modelling time A discrete model of time is sufficient for the dependability requirements in SERENITY. Real-time issues will not be considered on the level of network and devices in SERENITY. At the level of requirements specifications the expected duration of particular actions is generally not known. In contrast, the duration of actions or sequence is part of the dependability requirements specification. Therefore, it would be not useful to explicitly include time and duration into the system specification. For the specification of timing requirements we need the following information:

- A discrete *time model* needs to be defined for each system. The time model consists of two sets:
 1. The set T_S describes the available time steps for a system S . This set can for example consist of natural numbers describing system ticks, seconds, etc. or a description of time in terms of hours, minutes, seconds.
 2. The set TI_S contains the values available for the description of time intervals.
- Further, a function $duration_S : \Sigma^* \rightarrow TI_S$ provides the duration of a particular sequence of actions. Please note that it may be that $duration_S(ab) \neq duration_S(a) + duration_S(b)$.

Modelling probabilities The model for probabilities is very similar to the time model explained above. Again, the probabilities of particular actions or sequences of actions are not previously known but are part of dependability requirements to be provided by dependability patterns. In contrast to the specification of time there is no need to specify a “probabilities model” for each system. We can define that probabilities are always numbers in the interval between 0 and 1. Thus, with $Pr_S : \Sigma^* \rightarrow [0, 1]$ we denote the probability for the occurrence of sequences of actions in S .

2.3.2 Availability

Availability is the property that a system is available for use when someone needs the service. We assume that in every sequence of actions we can identify the actions where a service is requested and the actions where successful access occurs. A service is *available* if the successful access occurs within a particular duration (time-out). Parts of sequences with access requests and no success within the defined time-out duration can be regarded as failure time where the user cannot access the service. We can now define availability in terms of the percentage of time in which the service is available:

$$Availability(S, Service) = \frac{available(S, Service)}{(available(S, Service) + failure(S, Service))} \%$$

Here, *available* is the sum of all durations where the Service is available in all sequences in S while *failure* is the sum of all failure times in all sequences in S . The concrete representation of these functions depends on the particular system model.

Now, availability of a service in a system S may be specified as $Availability(S, Service) > 99,9\%$. Another aspect of availability is concerned with the *system interface robustness*. A system interface should not cause a system failure when wrong input is provided. This includes user interfaces as well as communication interfaces.

2.3.3 Reliability

Reliability is the property that a system continuously provides the correct service. One important reliability requirement can be expressed as the mean time between failures. Other reliability requirements (e.g. failure rate or failure probability) can be formalized in a similar way.

Mean time between failures (MTBF) Mean time between failures (MTBF) expresses the time a system runs without any failures that affect the use of the system (note that failures can occur that are not noticed by any user of the system and that do not directly influence the functional behaviour of a system. These failures can be neglected for the MTBF but might, nevertheless, be important for the security of the system. Consider for example Trojan horses, key-loggers, bot-nets and other malicious software that does not directly influence the reliability of the system).

For a time duration t we define the mean time between failures as

$$MTBF(S, Service, t) = \frac{\text{operating-time}(S, Service, t)}{\text{number-of-failures}(S, Service, t)}$$

where *operating-time* expresses the total sum of all possible sequences with duration t and *number-of-failures* counts the failures that occur in these sequences. This definition can be refined by taking into account the probabilities of the sequences.

2.3.4 Maintainability

Maintainability is the property that a system can be restored to provide the services within a given time after failure. Maintainability can be specified by the following requirements. Other requirements considering physical failures, recover time by repair, etc. are out of the scope of a software framework.

Recover time after failure We assume that after a system failure occurred, a particular action a cannot be executed. The *recover time after failure* is the time that is needed in the maximum to achieve a system state where a can occur without any further delay. Thus, the recover time is specified in relation to a particular action. Please note that this is not a general requirement on the occurrence of a . There may be ordinary (non-failure) system states where a cannot occur for a time that is longer as the required recover time after failure.

Impact of system reset System reset frequently occurs in modern computing devices. Many causes for system reset exist, e.g. power failure, software or hardware errors, or system maintenance and security updates. System reset can occur suddenly without user interaction or might be induced by a user of the system. In any case, the impact of the system reset on the applications and services running on the system needs to be small. The following general requirements can be identified.

1. System reset shall not result in the loss of information about ongoing services.
2. System reset shall not result in the loss of data.
3. System reset shall not cause system function degradation.

For item 1 and 2 services and data that should not be affected by the system reset can be identified relative to sequences of actions. Item 3 is more difficult, because it is concerned with the possible continuations of the system behaviour after a particular sequence of actions with a system reset. Formally this is a liveness requirement.

3 A language for the specification of S&D requirements for networks and devices

The previous chapter has introduced the basic notions that can be used for the accurate specification of S&D requirements. However, these notions are very general. Therefore, using these notions to specify more concrete S&D requirements as needed for SERENITY pattern specifications can result in very complex expressions. Such expressions can be difficult to understand and the automatic processing of these expressions by the SERENITY framework would require complex reasoning mechanisms. Consequently, the S&D requirements language for networks and devices to be used in SERENITY needs to be based on refined notions of security and dependability, describing concrete requirements. The following two sections introduce these refined notions.

In the following it is assumed that every action is associated with exactly one agent executing this action. Further it is assumed that some actions are characterized as *send* and *receive* actions to be used for communication between agents. Names are used for actions, parameters or agents in the descriptions as placeholders for arbitrary actions, parameters and agent names.

3.1 Security requirements

1. **authentic-send – Authenticity of sending for the recipient**

For a send action **send** by agent A and receive action **rec** by agent B, **authentic-send(send,rec,B)** expresses the requirement that whenever B executes **rec** the action **send** is authentic for B corresponding to Definition 1.

2. **authentic-send-all(A,B)**

For all send actions **send** by agent A and receive actions **rec** by agent B shall the send action be authentic for the recipient B, i.e. **authentic-send(send,rec,B)** shall hold. Analogously, for all send actions **send** by agent B receive actions **rec** by agent A shall the send action be authentic for the recipient A, i.e. **authentic-send(send,rec,A)** shall hold.

3. **authentic-rec – Authenticity of receipt for the sender**

For a receive action **rec** by agent B and some action **confirm-rec** by agent A, **authentic-rec(rec,confirm-rec,A)** expresses the requirement that whenever A executes **confirm-rec** the action **rec** is authentic for A corresponding to Definition 1.

4. **authentic-local – Authenticity of local actions**

For actions **local-action** and **confirm-action** both executed by the same agent A, **authentic-local(local-action,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** the action **local-action** is authentic for A corresponding to Definition 1.

5. **authentic-remote – Authenticity of remote actions**

For action **remote-action** by agent B and action **confirm-action** by agent A, **authentic-remote(remote-action,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** the action **remote-action** is authentic for A corresponding to Definition 1.

6. **non-rep-origin – Non-repudiation of origin**

Remark: Different non-rep req. have to be distinguished, because the solutions have to be different. E.g. non-rep of origin can be achieved by dig.signatures, non-rep of receipt might require a TTP, while non-rep for local actions is difficult to realise (possibility: logging).

For a send action **send** by agent A and receive action **rec** by agent B, **non-rep-origin(send,-rec,B)** expresses the requirement that whenever B executes **rec** the action **send** is authentic for B and there exist proof actions by B for action **send** corresponding to Definition 2.

7. **non-rep-rec – Non-repudiation of receipt**

For a receive action **rec** by agent B and some action **confirm-rec** by agent A, **non-rep-rec(rec,confirm-rec,A)** expresses the requirement that whenever A executes **confirm-rec** the action **rec** is authentic for A and there exist proof actions by A for action **rec** corresponding to Definition 2.

8. **non-rep-local – Non-repudiation of local actions**

For actions **local-action** and **confirm-action** both executed by the same agent A, **non-rep-local(local-action,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** the action **local-action** is authentic for A and there exist proof actions by A for action **local-action** corresponding to Definition 2.

9. **non-rep-remote – Non-repudiation of remote actions**

For action **remote-action** by agent B and action **confirm-action** by agent A, **non-rep-remote(remote-action,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** the action **remote-action** is authentic for A and there exist proof actions by A for action **remote-action** corresponding to Definition 2.

10. **auth-phase/non-rep-phase – Authenticity or non-repudiation in a particular phase**

For actions **start-phase,end-phase** and **phase-action** by arbitrary agents plus **confirm-action** by A, **auth-phase(phase-action,start-phase,end-phase,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** then **phase-action** is authentic in the phase starting with **start-phase** and ending with **end-phase** corresponding to Definition 4.

Remark: This requirement does not imply any information about the agent executing **phase-action**.

Note that it may be necessary to refine this requirement with respect to authenticity of send or receive actions or local or remote actions. More examples will provide more insight here.

11. **auth-phase/non-rep-phase – Authenticity or non-repudiation in a particular phase**

In addition to authenticity in a phase, **non-rep-phase(phase-action,start-phase,end-phase,confirm-action,A)** adds the requirement that A needs to be able to prove the occurrence of **phase-action** in the particular phase. Thus, adequate proof actions in accordance with Definition 2 are required.

12. **“General indistinguishability”** This is a very strong requirement, that is useful for password or any other authorisation data. This implies that this particular data has to be indistinguishable while stored on a device as well as during transmission over any network links. This is probably only useful for the design phase. During runtime more concrete requirements taking into account the actual system are necessary.

13. **End-to-end confidentiality** This requirement is a requirement on data transfer: If data is confidential before the transfer it remains confidential (except for the recipient). No agents except those in **who** may learn the value of **data-to-be-conf** from the actions in **actions-to-learn-from** although knowing that the possible values of the data to be confidential are all in the set **possible-values**.

As explained in Section 2.2.5, dependencies between the parameters of the actions from which an observer can learn are expressed by choosing an adequate language L . Thus end-to-end-confidentiality has various different characteristics:

- **end-to-end-conf(actions-to-learn-from,data-to-be-conf,possible-values,who)**
No dependencies are known, thus no language L is needed.
- **end-to-end-conf-L(actions-to-learn-from,data-to-be-conf,possible-values,who,L)**
Some dependencies are known, expressed by the language L .
- One dependency that agents will often know is that the same parameter must have been sent that was received. Thus a requirement addressing this specifically is useful:
end-to-end-conf-LSendRec(actions-to-learn-from,data-to-be-conf,possible-values,who, set-of-send-rec-pairs)

14. **end-to-end-conf-all(A,B)**

For all send actions **send** by agent A and receive actions **rec** by agent B , the data sent shall be confidential for all other agents, i.e. **end-to-end-conf**({ **send**($A,B,\dots,data-sent,\dots$), **rec**($B,A,\dots,data-sent,\dots$)}, $data-sent,possible-values, \{A,B\}$) shall hold.

Analogously, for all send actions **send**(B,A,\dots) and receive actions **rec**(A,B,\dots), the data sent shall be confidential for all other agents, i.e. **end-to-end-conf**({ **send**($B,A,\dots, data-sent,\dots$), **rec**($A,B,\dots,data-sent,\dots$)}, $data-sent,possible-values, \{A,B\}$) shall hold.

This security requirement can only be used if there are no dependencies between parameters. Since different actions will induce different dependencies, hence different languages L , it seems too complex to address them all in one requirement. Consequently, if such dependencies exist the more refined requirements described above have to be used.

15. **Confidentiality of data stored on a device**

Nobody but the agents in the set **who** shall learn from the actions in **actions-to-learn-from** which values the data in **data-to-be-conf** stored on **device** have although knowing that the set of possible values is **possible-values**. Again, depending on existing or not existing dependencies between the parameters, we have three different characteristics of this requirement:

- No dependencies exist: **device-conf(actions-to-learn-from,device,data-to-be-conf,possible-values,who)**
- The existing dependencies are described by L : **device-conf-L(actions-to-learn-from,device,data-to-be-conf,possible-values,who,L)**
- The only dependency known by agents is that the same parameter is sent that was received: **device-conf-LSendRec(actions-to-learn-from,device,data-to-be-conf,possible-values,who, set-of-send-rec-pairs)**

16. **access-allowed(who,device,data,access-type)** is the requirement that only agents in the set **who** can have access of the type **access-type** to **data** on **device**.

This implies that whoever tries to have access to the data has to authenticate himself as being member of **who**. Thus in the refined system that describes how this is achieved, **access-allowed**

is an instantiation of **enforce-behaviour**: Whenever an access action by any agent occurs, some action must have occurred before that authenticates the agent as being member of **who**. What is considered access to a specific device depends on the **access-type**. For example, read and write access have to be distinguished and will require different solutions that comply with the requirement. We may even distinguish between different types of read access, e.g. between just copying a certain file without understanding its content and reading a file with the consequence of understanding its content.

Definition 10 (access-allowed) Let P be an agent, Σ a set of actions, and $access(P, device, data, access-type) \in \Sigma$ the action of P to access data on device with some access type. Let $auth(P, who) \in \Sigma$ be the action that authenticates P as being a member of **who**. Let further $S \subseteq \Sigma^*$ be a system. We define a language L as follows:

$$L = \Sigma^* \setminus (\Sigma \setminus \{auth(P, who)\})^* \{access(P, device, data, access-type)\} \Sigma^*$$

Then **access-allowed**($P, device, data, access-type$) holds if **enforce-behaviour**(L, S) holds, i.e. if $S \subseteq L$.

17. **sequence(a,b,c)**

This requirement describes that if action a is followed by action c (denoted by $a < c$), an action b must have occurred in between, i.e. $a < b < c$ must hold. This can be used for example to describe the requirement that when having installed a program, before using it some license agreement must be acknowledged.

Definition 11 (sequence(a,b,c)) Let Σ be a set of actions, $a, b, c \in \Sigma$. Let $S, L \subseteq \Sigma^*$. with L defined as follows:

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

Then **sequence**(a, b, c) holds if **enforce-behaviour**(L, S) holds.

18. **no-sequence(a,b)**

This requirement describes that after action a , action b must not occur anymore. This can be used for example to describe the requirement that when having sold an upgrade to a program, the upgrade can not be used anymore.

Definition 12 (no-sequence(a,b)) Let Σ be a set of actions, $a, b \in \Sigma$. Let $S, L \subseteq \Sigma^*$ with L defined as follows:

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

Then **no-sequence**(a, b) holds if **enforce-behaviour**(L, S) holds.

3.2 Dependability requirements

This section provides language constructs for the specification of relevant dependability requirements as defined in Section 2.3.

1. **High-availability(S,a)** holds if $Availability(S, a) > 99,9\%$. It is assumed that a models a particular service.
2. **High-MTBF(t)** holds if the MTBF for a system is higher than t for an average running time of the system.
3. **recover-time(a,time)** holds if after every failure in all continuations of the system behaviour, action a is possible after duration t .

4 Guidelines for requirements elicitation

4.1 Initial questions for requirements elicitation

Questions 1-4 are concerned with a slightly refined view on a scenario, Questions 4 and 5 cover general requirements for devices and networks, and the remaining question covers requirements related to particular actions/parameters in the scenarios. Mainly, plain English can be used to describe the requirements.

1. **What are the ACTORS of the scenario?** Actors can be humans, devices or any other entity that can “act” in the scenario. If concrete devices and network components can be named at this level of scenario description, requirements can be formulated for these.
2. **What are the ACTIONS in the system?** For every ACTOR name the main actions in the scenario. Proposed notation: **action-name**(ACTOR, parameters). The parameters can express data, configuration, current context, network addresses, etc..
3. **What are typical sequences of actions?** One sequence might already provide a good understanding of the scenario.
4. **Are there any generally satisfied assumptions for the scenario?** These can describe properties of the scenario, devices, actors’ behaviour etc. that generally can be assumed. (Note that the examples in Sections 5.1 and 5.2 are just for the sake of the example and might not reflect the real situation in the scenarios.)
5. **Are there any general requirements for devices?** This should **not** include requirements covered by the usual baseline protection (i.e. regular backup, frequent security updates, etc). Examples for “good” requirements:
 - Corporate data on John’s laptop has always to be confidential for John, i.e. nobody else is allowed to learn any information on this data.
 - A service on a particular server has to be always available (might induce a general requirement on the network connection to the server).

6. **Are there any general requirements for network communication?** Example: Every wireless communication between two devices needs message confidentiality (i.e. nobody but sender and recipient should get any additional information on the content of the message).
7. **What are the requirements for single actions or particular sequences of actions or *phases* beginning and ending with a particular action?** Security requirements can be stated by using the notions of authenticity, proof of authenticity, authenticity with respect to a phase, and confidentiality as described below. If possible, all information in the paragraphs “Input needed. . .” should be included. Dependency requirements and any other requirements that do not fit into these notions should be provided in plain English. Please try to give explanations and maybe reasons for the requirement and avoid to use only keywords (like “must be dependable”).

5 Examples based on SERENITY scenarios

5.1 Example: Communication scenario

5.1.1 Agents

This scenario comprises two human agents, namely Sally and Olli and their devices, Sally’s Internet Tablet and Olli’s Laptop. The SERENITY framework can only control the behaviour of the devices. Therefore, in the following actions denoted with *Sally* occur on Sally’s Internet Tablet and actions denoted with *Olli* occur on Olli’s Laptop. Thus, we have the following agents:

Sally models Sally’s Internet Tablet
Olli models Olli’s Laptop

In addition to the agents we need the groups of agents with a particular reputation in the context of the game.

5.1.2 Some “state” parameters and preliminary assumptions

We assume that the SERENITY framework will receive reliable information on the state of a particular device. Further we assume that this state cannot change in an arbitrary way, i.e. the SERENITY framework is aware of any changes. This is particularly relevant for active communication interfaces, executable code running on the platform, etc.

In the communication scenario the following parameters are relevant for Olli’s laptop:

Active network device (ND): For simplicity we assume that only one network device can be active on Olli’s laptop. Possible values for ND are: company network (LAN), local wireless hotspot (WLAN), cellular connection (CC) or no active device (NONE).

Office application (OA): This parameter shows the status of the office application on Olli’s laptop. OA can be active (ON) or inactive (OFF).

For Sally, only the active network connection is relevant.

We assume that the value of ND can only be changed by the action *change-connection*, while OA can only be changed by *start-OA* and *end-OA*. Please note that we assume that these properties are not provided by the SERENITY framework. However, it might still be possible to use the framework for checking the satisfaction of these requirement during run-time.

5.1.3 Actions (example sequence)

Remark on the notation: For every action the *name* of the action is followed by parameters in parentheses. These parameters are local to this action and are not global variables. The main goal of this specification is to be able to provide precise security requirements. Behaviour (and states) cannot be derived from this set of actions.

The following represents one possible (and expected) sequence of actions. Other sequences are possible as well.

1. **game-invitation**(Sally, ND=WLAN)
2. **change-connection**(Olli, ND=LAN, OA=ON,office-device, office-data, new-ND=WLAN)
3. **end-OA**(Olli, ND=WLAN, OA=ON,office-device,office-data)
4. **check-for-invitation**(Olli, ND=WLAN, OA=OFF)
5. **accept-invitation**(Olli, ND=WLAN, OA=OFF, Sally, Sally-ND=WLAN, reputation-Sally=good,session)
6. **start-game**(Sally, ND=WLAN, Olli, Olli-ND=WLAN, reputation-Olli = good,session)
7. **play-game**(Olli, ND=WLAN, OA=OFF, Sally, Sally-ND=WLAN, upgrade=anti-sub, reputation-Sally=good,session)
8. **play-game**(Sally, ND=WLAN, Olli, Olli-ND=WLAN, upgrade=logos,reputation-Olli =good,session)
9. **change-connection**(Olli, ND=WLAN, OA=OFF, office-device, office-data,new-ND=CC)
10. **play-game**(Olli, ND=CC, OA=OFF, Sally, Sally-ND=WLAN, upgrade=anti-sub, reputation-Sally = good, session)
11. **play-game**(Sally, ND=WLAN, Olli, Olli-ND=CC,upgrade=logos,reputation-Olli=good,session)
12. **suggest-trade**(Sally, ND=WLAN, Olli, Olli-ND=CC, reputation-Olli=good, logos, anti-sub,session)
13. **successful-trade**(Olli, ND=CC, OA=OFF, Sally, Sally-ND=WLAN, reputation-Sally =good, logos, anti-sub,session)
14. **accept-trade**(Olli, ND=CC, OA=OFF, Sally, Sally-ND=WLAN, reputation-Sally=good, logos, anti-sub,session)
15. **successful-trade**(Sally, ND=WLAN, Olli, Olli-ND=CC, reputation-Olli=good, logos, anti-sub,session)
16. **play-game**(Olli, ND=WLAN, OA=OFF, Sally, Sally-ND=WLAN,upgrade=logos, reputation-Sally=good,session)
17. **play-game**(Sally, ND=WLAN, Olli, Olli-ND=WLAN,upgrade=anti-sub, reputation-Olli=good,session)
18. **end-game**(Olli, ND=WLAN, OA=OFF, Sally, Sally-ND=WLAN, reputation-Sally=good,session)

19. **end-game**(Sally, ND=WLAN, Olli, Olli-ND=WLAN, reputation-Olli=good,session)

Remark: All actions are considered to be local. Example: **accept-invitation** by Olli indeed only represents Olli's part. The receipt of an accept message by Sally would be part of Sally's **start-game** action rather than being part of **accept-invitation**.

5.1.4 Security requirements

1. Req. 3.2.4.3 If one of the players' identity fails to be verified the trade should be denied.

A successful trade can only occur if either **suggest-trade** or **accept-trade** is authentic. I.e. for all sequences of actions containing action **successful-trade**(Olli,...,Sally,...,logos, anti-sub,session), **suggest-trade**(Sally,ldots,Olli,...,logos,anti-sub,session) is authentic for Olli and for all sequences of actions containing **successful-trade**(Sally,...,Olli,...,logos,anti-sub,session), **accept-trade**(Olli,...,Sally,...,logos,anti-sub,session) is authentic for Sally.

authentic-remote(**successful-trade**(Olli,...,Sally,..., logos,anti-sub,session),**suggest-trade**(Sally,...,Olli,...,logos,anti-sub,session),Olli)

authentic-remote(**successful-trade**(Sally,...,Olli,..., logos,anti-sub,session),**accept-trade**(Olli,...,Sally,...,logos,anti-sub,session),Sally)

2. The following requirement is related to

Req. 3.2.1.3 Every device should have a policy that defines which application can run and under which constraints (if applicable).

Req. 3.2.1.4 If the application's properties do not match the properties allowed by the device's policy, the application shall not be allowed to run.

Req.3.2.3.1 Changes in the environment shall be notified to the Serenity Framework enforcing the restrictions and policies.

Req. 3.2.3.2 If a change in the environment does not conform to the device's policy, the affected application shall be closed.

For all actions of Olli it is required that OA=A only holds if ND=LAN.

This represents a local security policy and can (in a stronger sense) not be considered a "security requirement". One security requirement could be that office application data has to be confidential inside the company:

device-conf({**change-connection**(Olli, ND=LAN, OA=ON, office-device,office-data, new-ND=WLAN), **end-OA**(Olli, ND=WLAN, OA=ON,office-device,office-data),device, office-data,possible-values?,{Olli})

However, such a security policy can of course be enforced by the framework.

3. Req. 3.2.4.1 The sensitive data transmitted between the players shall be protected from eavesdropping and modification by external non-trusted parties.

— The parameters *logos* and *anti-sub* in actions **suggest-trade**, **accept-trade** and **successful-trade** shall be confidential. Possible values for these items can be all data that can be traded within the context of the game. Only Olli and Sally are allowed to know the values of these parameters:

end-to-end-conf ({**suggest-trade**(Sally,...,Olli,...,logos,anti-sub,session),

successful-trade(Olli,...,Sally,...,logos,anti-sub,session),

accept-trade((Olli,...,Sally,...,logos,anti-sub,session),

successful-trade(Sally,...,Olli,...,logos,anti-sub,session)}, {logos,anti-sub},game-data,{Olli,Sally})

- The integrity of *logos* and *anti-sub* in actions **suggest-trade**, **accept-trade** and **successful-trade** shall be provided. Since integrity alone without knowing who generated the data is useless (see remark in Section 2.2.4), this translates to

authentic-send(**suggest-trade**(Sally,...,Olli,...,logos,anti-sub,session), **successful-trade**(Olli,...,Sally,...,logos,anti-sub,session), Olli)

authentic-send(**accept-trade**(Olli,...,Sally,...,logos,anti-sub,session), **successful-trade**(Sally,...,Olli,...,logos,anti-sub,session), Sally)

4. Req. 3.2.2.6 All game actions shall be authentically executed by the player with the unique ID that was verified before.

- For agent Olli, all actions **start-game**(Sally,...,Olli,...,session) and **play-game**(Sally,...,Olli,...,session) shall be authentically executed by Sally within the current phase started with **accept-invitation**(Olli,...,Sally,...,session) (assuming that **accept-invitation** includes the check of Sally's identity by Olli).

- For Sally, all actions **play-game**(Olli,...,Sally,..., session) shall be authentically executed by Olli within the current phase started with **start-game**(Sally,...,Olli,...,session) (assuming that **start-game**(Sally,...,Olli,...,session) includes the check of Sally's identity by Olli).

5. Req. 3.2.2.7 No game actions from parallel or previous game sessions shall be accepted in the current session of the game.

This is an instantiation of **enforce-behaviour**: Either **sequence**(**end-game**(Olli,...,Sally,...,session1), **start-game**(Olli,...,Sally,...,session2), **play-game**(Olli,...,Sally,...,session2)) or **sequence**(**end-game**(Olli,...,Sally,...,session1), **accept-invitation**(Olli,...,Sally,...,session2), **play-game**(Olli,...,Sally,...,session2)) must hold with $session1 \neq session2$.

Analogously, **sequence**(**end-game**(Sally,...,Olli,...,session1), **start-game**(Sally,...,Olli,...,session2), **play-game**(Sally,...,Olli,...,session2)) or **sequence**(**end-game**(Sally,...,Olli,...,session1), **game-invitation**(Sally,...,Olli,...,session2), **play-game**(Sally,...,Olli,...,session2)) must hold with $session1 \neq session2$.

6. Req. 3.2.4.7 After the trade is successful, only the receiving entity shall be able to use the upgrade. The previous owner shall not be able to use it anymore.

Again, this is an instantiation of **enforce-behaviour**. Here we have the requirement that **no-sequence**(**successful-trade**(Olli,...,upgrade,...), **play-game**(Olli,...,upgrade,...)) holds. Analogously,

no-sequence(**successful-trade**(Sally,...,upgrade,...), **play-game**(Sally,...,upgrade,...)) must hold.

5.2 Example: Smart Items scenario

5.2.1 Agents

The scenario can be modelled with the following agents:

Bob	models the person Bob
Bob-eh	models Bob's e-health terminal which is a PDA phone with a particular e-health application for patients running on it
SmartShirt	Bob's Smart T-Shirt with the sensors that measure Bob's blood pressure etc. The Smart T-Shirt is connected to Bob's PDA by wire (e.g. USB) or wireless (bluetooth, infrared).
SensorNetCentral	The sensor network's central station
Charlie	the doctor Charlie
groupDoctors	a group of doctors principally being able to take care of Bob, containing in particular Bob's family doctor Andrew and Andrew's substitute Charlie
Doc-eh	doctor Charlie and his e-health terminal which is a PDA phone with a particular e-health application for doctors running on it
ERC	the Emergency Response Center (i.e. the complete system with devices, databases, applications etc.)
Alison	The social worker Alison
groupWorkers	a group of social workers, in particular containing Alison
Alison-eh	The social worker Alison's e-health terminal which is a PDA phone with a particular e-health application for social workers running on it
Pharmacy	the Pharmacy and their PC
medTeam	the medical team
medTeam-eh	the medical team's e-health terminal which is a PDA phone with a particular e-health application for the medical team running on it
LIC	Location Information Centre

It seems necessary to distinguish between Bob and his e-health terminal since some of the actions of Bob's e-health terminal below are triggered by Bob, others are triggered by the Smart T-Shirt. Furthermore, we want to model the explicit agreement of the doctor Charlie to take care of Bob's health problems in scene 1, so we need to distinguish between Charlie and his e-health terminal as well. The same holds for the medical team: We want to model the explicit agreement of the medical team that they will take care of Bob in the situation of urgency (scene 4). Finally, since Requirement 2.2.2.6 says that Alison shall not be allowed to access the prescription itself, we need to distinguish between her and her e-health terminal.

In some actions listed in the following, we denote by **data = xxx** that the data of this action in this particular scene is equal to xxx. This shall support the understandability of the description. Furthermore the following not only lists the actions necessary to model the four scenes but repeats an action whenever it is performed more than once. This shall give an idea of what is the desired sequence of actions.

We note that in a bigger model containing more actors there is the possibility that Bob's e-health terminal contacts an institution other than ERC. However, the smart T-Shirt can not be connected to some device other than Bob's e-health terminal.

5.2.2 Actions

Scene 1 - Faintness alert In scene 1 we have the following actions:

1. **requ-assistance**(Bob,Bob-eht,data=ERC)
2. **requ-data**(Bob-eht,SmartShirt)
3. **send-request**(Bob-eht,ERC,data=(assistance,Bob,Bob-data))
4. **check-availability**(ERC,groupDoctors,data)
5. **available**(Doc-eht,ERC,data=Bob)
6. **take-over**(ERC,Doc-eht,data=(task,Bob))
7. **ackn-task**(Charlie,Doc-eht,data=(task,Bob))

With the above actions we model a situation where Charlie's e-health terminal has access to Charlie's calendar so that the e-health terminal can notify the ERC of Charlie's availability. However, Charlie has explicitly to acknowledge taking over the task of taking care of Bob.

8. **requ-info**(Doc-eht,ERC,data=Bob)
9. **send-info**(ERC,Doc-eht,data=(Bob,Bob-data,...))
10. **send-prescription**(Doc-eht,Bob-eht,data)
11. **rec-prescription**(Bob-eht,Doc-eht,data)

Scene 2 - Delivery of the medicine

1. **requ-med-del**(Bob,Bob-eht,data)
2. **requ-med-del**(Bob-eht,ERC,data=(Bob,prescription,...))
3. **check-availability**(ERC,groupWorkers,data)
4. **available**(Alison-eht,ERC,data)
5. **take-over**(ERC,Alison-eht,data=(task,Bob,prescription,...))
6. **ackn-task**(Alison,Alison-eht,data=(task,Bob,prescription,...))

Again the above models that Alison explicitly acknowledges to take over the task of delivering the medication to Bob.

7. **requ-medication**(Alison-eht,Pharmacy,data)

8. **process-medication**(Pharmacy,data=(Bob,prescription,authorization))

This action includes probably validating whatever **Alison-eh** provides as data to be validated and the actual delivery of the medication to **Alison**.

9. **send-ackn**(Alison-eh,Pharmacy,data)

10. **send-ackn**(Pharmacy,Alison-eh,data)

The last two actions model the successful message exchange between the pharmacy computer and Alison's PDA mentioned in A7.D2.1. The related security requirements are Req. 2.2.2.10 Req. 2.2.2.13 (see below).

11. **ackn-delivery**(Alison,Alison-eh,data=(Bob,prescription,...))

12. **invoice**(Alison-eh,Bob-eh,data)

13. **ackn-receipt**(Bob-eh,Alison-eh,data)

Scene 3 - False alarm

1. **alert**(SmartShirt,Bob-eh,channel,data)

2. **alert**(Bob-eh,ERC,channel,data)

The channel parameter in the action **alert** enables to distinguish between different solutions using different communication media.

3. **query-position**(ERC,LIC,data=Bob)

4. **locate**(LIC,data=(Bob-eh,ERC,...))

Bob's e-health terminal is actually a mobile phone with a GPS, thus the LIC can determine Bob's location by using the GPS.

5. **send-position**(LIC,ERC,data=(locationBob,...))

6. **rec-position**(ERC,LIC,data=(locationBob,...))

7. **query-info**(ERC,SensorNetCentral,data=Bob)

8. **rec-info**(ERC,SensorNetCentral,data)

Scene 4 - emergency

1. **alert**(Bob-eh,ERC,channel,data)

2. **query-position**(ERC,LIC,data=Bob)

3. **locate**(LIC,data=(Bob-eh,ERC,...))

4. **rec-position**(ERC,LIC,data=(locationBob,...))

5. **check-availability**(ERC,groupMedTeam,data)

6. **available**(medTeam-eh,ERC,data)
7. **take-over**(ERC,medTeam-eh,data=(task,Bob,...))
8. **ackn-task**(medTeam,medTeam-eh,data=(task,Bob,...))
9. **requ-info**(medTeam-eh,ERC,data=Bob)
10. **rec-info**(medTeam-eh,ERC,data)
where data contains all information about Bob that ERC has collected in the scenes.
11. **requ-info**(medTeam-eh,Bob-eh,data)
12. **rec-info**(medTeam-eh,Bob-eh,data)

5.2.3 Security requirements

General requirements Many of the requirements for the Smart Items scenario listed in A7.D2.1 are similar. They all require authenticity, confidentiality and integrity for all communication between two agents. As an example, we show how to transform these into the adequate requirements listed in Section 3 for one of them.

— Req. 2.2.1.2 Each communication between the patient's e-health terminal and the ERC shall guarantee messages delivery, integrity and confidentiality of the data exchanged, and mutual authentication.

- Each send action involving **Bob-eh** and **ERC** shall be authentic for the respective recipient.

authentic-send-all(Bob-eh,ERC)

- For each action involving **Bob-eh** and **ERC**, the data exchanged shall be confidential for all other agents.

end-to-end-conf-all(Bob-eh,ERC)

- Integrity of the data exchanged in each action involving **Bob-eh** and **ERC** shall be provided.

Since for communication, integrity of the data only makes sense if one knows who generated it (see Section 2.2.4), this requirement is actually a requirement of authenticity of the send actions, which is already expressed above.

— The following requirements can be expressed analogously:

- Req. 2.2.1.13 referring to communication between **Bob-eh** and **SmartShirt**,
- Req. 2.2.1.14 referring to communication between **ERC** and **Doc-eh**,
- Req. 2.2.1.20 referring to communication between **Bob-eh** and **Doc-eh**,
- Req. 2.2.2.4 referring to communication between **Alicon-eh** and **ERC**,
- Req. 2.2.2.7 referring to communication between **Alison-eh** and **Pharmacy**,
- Req. 2.2.2.14 referring to communication between **Bob-eh** and **Alison-eh**,

- Req. 2.2.3.9 referring to communication between **ERC** and **SensorNetCentral**,
- Req. 2.2.4.5 referring to communication between **ERC** and **medTeam-eh**,
- Req. 2.2.4.6 referring to communication between **Bob-eh** and **medTeam-eh**

— Req. 2.2.1.6 Only authorized agents may access patient's data. Authorized agents are in this scenario Doc-eh, ERC, medTeam-eh (the medical staff of ERC), and Pharmacy. Alison herself is not authorized to access medical data (see Req. 2.2.2.6).

Note: *access* means both read and write access, thus this requirement is not covered by requiring confidentiality for the data.

The implications of Req. 2.2.1.6 for the communication are covered by the requirements listed above. As to accessing the patients' data on the devices, it is still open which type of access we will have to consider in the requirements listed in the following.

- Nobody but **Bob-eh** may have access to **SmartShirt**:
access-allowed(Bob-eh,SmartShirt,Bob-data,access-type).
- Nobody but **Bob** and **ERC** may have access to the data stored in **Bob-eh**. (It may be assumed that Bob is allowed to access his own data (at least to read it). Furthermore, although in no action the ERC actually requests any data from Bob's PDA, since the data is sent to the ERC we may assume that ERC is allowed to do so.)
access-allowed({Bob,ERC},Bob-eh,Bob-data,access-type)
- Since in our model only **ERC** is allowed to access the data on **ERC**, we have the requirement that only **ERC** may have access to Bob's data stored at **ERC**:
access-allowed({ERC},ERC,Bob-data,access-type)
- Since the prescription stored on **Alison-eh** and **Pharmacy** is considered patient's data (see Req. 2.2.2.6), we have the requirements that only authorized agents may have access. Thus only **Pharmacy** may have access to the patient's prescription data stored on the pharmacy PC:
access-allowed({Pharmacy},Pharmacy,prescription,access-type)

We assume that the Pharmacy does not directly access the prescription on Alison's e-health terminal but that the prescription is sent from one device to the other. Now Req. 2.2.2.6 says that Alison herself shall not have access to (shall not be able to read and understand) the patient's prescription, on the other hand Alison is obliged to verify that the medication delivered by the pharmacy is correct (Req. 2.2.2.11). So parts of the prescription have to be accessible (understandable) by Alison. Thus we model the prescription data as two parts **prescription = (conf-presc-data,presc-data)**. **presc-data** enables Alison to verify the correctness of the medication. It follows that Alison and nobody else is allowed to read and understand this part, while Alison is not allowed to read and understand **conf-presc-data**. However, Alison is allowed to read **prescription** and send it to **Pharmacy** where reading in this context does not include understanding the data:

access-allowed({Alison},Alison-eh,presc-data,access-type=understand)

access-allowed({ },Alison-eh,conf-presc-data,access-type=understand)

The latter requirement means that nobody is allowed to read and understand the confidential prescription data on Alison's e-health terminal.

- Req. 2.2.1.21 Only authorized agents may have access to patient data on **Doc-ehT**. The only authorized agent that may have access is Charlie:

access-allowed({Charlie},Doc-ehT,Bob-data,access-type)

Note: The data gathered by the sensor network is not considered medical data.

- Req. 2.2.1.17 Electronic prescriptions shall be used only once.

In some actions, *using prescription data* actually means *processing prescription data* on a specific device. This is the case for example for the action **process-medication**(Pharmacy, data=(Bob,prescription,authorization)). In such a case, the above requirement can be expressed by **no-sequence** as a particular instantiation of **enforce-behaviour**(L,S) where *L* denotes a language that does not contain sequences of actions with an action **process-medication**(Pharmacy1,data=(X,prescription,authorization1)), and another action **process-medication**(Pharmacy2,data=(Y,prescription,authorization2)), both containing the same prescription data.

no-sequence(**process-medication**(Pharmacy1,data=(X, prescription,authorization1)), **process-medication**(Pharmacy2,data=(Y, prescription,authorization2)))

- Req. 2.2.2.3 The social worker discovery process shall guarantee the non repudiation of the final decision.

This is **non-repudiation of remote action**: Alison may not be able to repudiate having acknowledged the task of delivering the medicine:

non-rep-remote(**ackn-task**(Alison,Alison-ehT,data=(task,Bob,prescription,...)), **ackn-task**(Alison,Alison-ehT,data=(task,Bob,prescription,...)),ERC)

We assume that ERC is able to notice the action **ackn-task** of Alison (for example because some ERC employee talks to Alison directly).

- No A3: Req. 2.2.2.5 The messages sent by the ERC to the e-health terminal of the selected social worker shall provide evidence that the selected social worker is authorized to get the prescribed medicine in behalf of the patient.

- Req. 2.2.2.6 The messages sent by the ERC to the e-health terminal of the selected social worker shall not provide means to the selected social worker to access to patient's medical information except those really necessary to accomplish his task (e.g., the identifier of medicine prescribed).

This means that Alison herself is not authorized to access the patient's prescription, thus this requirement is covered by Req. 2.2.1.6 Only authorized agents may access patient's data.

- No A3: Req. 2.2.2.13 If the selected social worker accepts the medicine from the pharmacist, **Alison-ehT** shall send a proof of receipt to **Pharmacy** to provide evidence that the selected social worker has got the medicine.

- Req. 2.2.2.17 If the patient accepts the medicine from the selected social worker, the patient's e-health terminal shall send a proof of receipt to the e-health terminal of the selected social worker to provide evidence that the patient has received the medicine.

This is a requirement for A1 or A2 but might result in some A3 requirement (integrity of log data, e.g.)

Requirements for one action only

- Req. 2.2.1.8 The doctor discovery process shall guarantee the non repudiation of the final results.

Note: the selected doctor cannot repudiate to have provided his acknowledgement to accomplish the patient's request and the ERC cannot repudiate to have discovered and entitled this doctor.

This is **non-repudiation of remote action**: Charlie may not be able to repudiate having acknowledged the task of taking care of Bob:

non-rep-remote(ackn-task(Charlie,Doc-eh,task=task,Bob)), send-info(ERC,Doc-eh,data=(Bob,Bob-data,...)), ERC)

- Req. 2.2.1.19 The electronic prescription shall insure that its integrity is not altered and that the issuer cannot repudiate that he has released the prescription.

The part about non-repudiation seems not to be A3. The requirement for integrity seems to be detached from any device the prescription might be stored on. We can of course require integrity of the prescription on any device it is stored on but it is not clear whether this is sensible. Integrity of the prescription is a property that is attached to the prescription on any device.

- Req. 2.2.2.10 When the pharmacist hands the medicine to the selected social worker, the pharmacy computer shall send to the e-health terminal of the selected social worker an electronic invoice that gives evidence of the medicine provided and of the pharmacist's identity.

This means the pharmacist shall not be able to repudiate to have provided the medicine.

On device level, this could be interpreted as the requirement that it can not be repudiated that **Pharmacy** has processed Bob's prescription:

non-rep-remote(process-medication(Pharmacy,data), send-ackn(Alison-eh,Pharmacy,data), Alison-eh)

- Req. 2.2.2.11 The selected social worker shall accept the medicine if and only if the medicine corresponds to the prescribed one.

Note: Usually this requires a human validation but solutions based on RFID can be envisaged (e.g., the selected social worker checks that the number on the medicine pack corresponds to the medicine identifier he was expecting for). However, for the moment we disregard this kind of solution, thus this is not a requirement on A3 level.

- Req. 2.2.2.15 When **Alison** hands the medicine to **Bob**, **Alison-eh** shall send to **Bob-eh** an electronic invoice that gives evidence of the medicine delivered and of **Alison's** identity.

Note: This means Alison shall not be able to repudiate to have delivered the medicine to the patient.

On device level, this could be interpreted as the requirement that it can not be repudiated that **Alison-eh** has processed Bob's prescription.

non-rep-remote(ackn-delivery(Alison,Alison-eh,data=(Bob,prescription,...)), invoice(Alison-eh,Bob-eh,data), Alison-eh)

Now Alison-eh has the proof that Alison has confirmed to have delivered the medicine. The assumption is that ERC is allowed to collect all proofs stored in e-health terminals of their staff and thus is able to control the work-flow.

— Req. 2.2.3.7 The delivery of the patient's location by the LIC shall ensure non repudiation, integrity and confidentiality of the data exchanged, and mutual authentication:

- We view the non-repudiation requirement not to be a requirement for communication and device level.
- Mutual authentication of the delivery of the patient's position means that the the location data shall be authentic for **ERC**, and that on the other hand the LIC does only provide the ERC with data about Bob's position.

authentic-send(**query-position**(ERC,LIC,data=Bob), **locate**(LIC,data=(Bob-eh,ERC,...), LIC)

authentic-send(**rec-position**(ERC,LIC,data=(locationBob,...)), **query-info**(ERC, SensorNetCentral,data=Bob), ERC)

- The integrity of the data sent in **rec-position** is covered by requiring authenticity of the send action.
- **end-to-end-conf** ({**send-position**(LIC,ERC,data=(locationBob,...)), **rec-position**(ERC,LIC,data=(locationBob,...)}, locationBob,possible-values,{LIC,ERC})
where *possible-values* denotes the set of values that are possible for Bob's location.

Dependability requirements We give some examples of dependability requirements. Other dependability requirements can be expressed in a similar way.

— Req. 2.2.1.1 and Req. 2.2.3.6 The ERC and the LIC shall be available 24 hours a day, i.e. for all actions x of the EHC and of the LIC shall hold **High-availability**(**S**, x).

— Req. 2.2.1.9 The Smart T-Shirt shall be available 24 hours a day (except for the times Bob takes it off, notifying his PDA.). Let *shirt-active*(S) be the set sequences of actions where Bob has not notified the PDA the the Smart T-Shirt is not available. Then, for all actions x of the Smart T-Shirt shall hold **High-availability**(*shirt-active*(S), x).

References

- [1] M. Abadi and M.R Tuttle, 1991. A Semantics for a Logic of Authentication. In *Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Canada*, pages 201–216.
- [2] M. Burrows, M. Abadi, and R. Needham, 1990. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8.
- [3] R. Grimm and P. Ochsenschläger, 2001. Binding Cooperation, A Formal Model for Electronic Commerce. *Computer Networks*, 37:171–193.
- [4] S. Gürgens, P. Ochsenschläger, and C. Rudolph, 2003. Parameter confidentiality. In *Informatik 2003 - Teiltagung Sicherheit*. Gesellschaft für Informatik.
- [5] L.C. Paulson, 1996. Proving Properties of Security Protocols by Induction. Technical Report 409, Computer Laboratory, University of Cambridge.
- [6] G. Wedel and V. Kessler, 1996. Formal Semantics for Authentication Logics. In *Computer Security - Esorics 96*, volume 1146 of *LNCS*, pages 219–241.