Identification of Security Requirements in Systems of Systems by Functional Security Analysis

Andreas Fuchs and Roland Rieke

Fraunhofer Institute for Secure Information Technology (SIT)
Rheinstrasse 75, 64295 Darmstadt, Germany
{andreas.fuchs,roland.rieke}@sit.fraunhofer.de

Abstract. Cooperating systems typically base decisions on information from their own components as well as on input from other systems. Safety critical decisions based on cooperative reasoning however raise severe concerns to security issues. Here, we address the security requirements elicitation step in the security engineering process for such systems of systems. The method comprises the tracing down of functional dependencies over system component boundaries right onto the origin of information as a functional flow graph. Based on this graph, we systematically deduce comprehensive sets of formally defined authenticity requirements for the given security and dependability objectives. The proposed method thereby avoids premature assumptions on the security architecture's structure as well as the means by which it is realised. Furthermore, a tool-assisted approach that follows the presented methodology is described.

Key words: security requirements elicitation, systems of systems security engineering, security analysis for vehicular communication systems

1 Introduction

Architecting novel mobile systems of systems (SoS) poses new challenges to getting the dependability and specifically the security requirements right as early as possible in the system design process. Security engineering is one important aspect of dependability [1]. The security engineering process addresses issues such as how to identify and mitigate risks resulting from connectivity and how to integrate security into a target architecture [2]. Security requirements need to be explicit, precise, adequate, non-conflicting with other requirements and complete [13].

A typical application area for mobile SoS are vehicular communication systems in which vehicles and roadside units communicate in ad hoc manner to exchange information such as safety warnings and traffic information. As a co-operative approach, vehicular communication systems can be more effective in avoiding accidents and traffic congestion than current technologies where each vehicle tries to solve these problems individually. However, introducing dependence of possibly safety-critical decisions in a vehicle on information from other systems, such as other vehicles or roadside units, raises severe concerns to security issues. Security is an enabling technology in this emerging field because without security some applications within those SoS would not be possible at all. In some cases security is the main concern of the architecture [22].

The first step in the design of an architecture for a novel system of systems is the requirements engineering process. With respect to security requirements this process typically covers at least the following activities [17, 16, 15]

– the identification of the target of evaluation and the principal security goals and the elicitation of artifacts (e.g. use case and threat scenarios) as well as risk assessment
– the actual security requirements elicitation process
– a requirements categorisation and prioritisation, followed by requirements inspection

In this paper we address the security requirements elicitation step in this process. We present a model-based approach to systematically identify security requirements for system architectures to be designed for cooperative applications in a SoS context. Our contribution comprises the following distinctive features.

Identification of a Consistent and Complete Set of Authenticity Requirements. We base our method on the following general assumption about the overall security goal with respect to authenticity requirements:

> *For every safety-critical action in a system of systems, all information that is used in the reasoning process that leads to this action has to be authentic.*

To achieve this, we first derive a functional model of a system by identification of atomic actions and functional dependencies in a use case description. From this model we generate a dependency graph with the safety-critical function under consideration as root and the origins of decision relevant information as leaves. Based on this graph, we deduce a set of authenticity requirements that is comprehensive and defines the maximal set of authenticity requirements from the given functional dependencies.

Security Mechanism Independence. The most common problem with security requirements is, that they tend to be replaced with security-specific architectural constraints that may unnecessarily constrain the choice of the most appropriate security mechanisms [4].

In our approach we avoid to break down the overall security requirements to requirements for specific components or communication channels prematurely. So the requirements identified by this approach are independent of decisions not only on concrete security enforcement mechanisms to use, but also on the structure, such as hop-by-hop versus end-to-end security measures.

Throughout this paper we use the following terminology taken from [1]: A *system* is an entity that interacts with other entities, i.e., other systems. These other systems are the *environment* of the given system. A *system boundary* is the common frontier between the system and its environment. Such a system itself is composed of *components*, where each component is yet another system. Furthermore, in [1] the *dependence* of system A on system B represents the extent to which system A's dependability is affected by that of system B. Our work though focuses on purely functional aspects of dependence and omits quantitative reasoning. For the approach proposed, we describe the *function* of such a system by a *functional model* and treat the components as atomic and thus we do not make preliminary assumptions regarding their inner structure. Rather, the adaption to a concrete architecture is considered to be a task within a follow-up refinement and engineering process.

The subsequent paper is structured as follows. Section 2 gives an overview of the related work on security engineering and requirements identification methodologies. In Sect. 3 we introduce a scenario from the automotive domain that will serve as use case throughout the rest of this text. Section 4 introduces the proposed approach to requirements identification, exemplified by application on the given use case. Section 5 presents an tool-assisted methodology that follows this approach utilising the scenario. Finally, the paper ends with conclusions and an outlook in Sect. 6.

2 Related Work

The development of new security relevant systems that interact to build new SoS requires the integration of a security engineering process in the earliest stages of the development life-cycle. This is specifically important in the development of systems where security is the enabling technology that makes new applications possible. There are several common approaches that may be taken, depending on the system architect's background.

In order to design a secure of vehicular communication system, an architect with a background in Mobile Adhoc Networks (MANETs) would probably first define the data origin authentication [27] of the transmitted message. In a next step he may reason about the trustworthiness of the transmitting system. An architect with a background in Trusted Computing [7] would first require for the transmitting vehicle to attest for its behaviour [25]. Advanced experts may use the Trusted Platform Module (TPM) techniques of sealing, binding, key restrictions and TPM-CertifyKey to validate the trustworthiness and bind the transmitted data to this key [24]. A distributed software architect may first start to define the trust zones. This would imply that some computational means of composing slippery wheels with temperature and position happen in an untrusted domain. Results may be the timestamped signing of the sensor data and a composition of these data at the receiving vehicle.

This shall only illustrate a few different approaches that might be taken in a security engineering process for new SoS. Very different types of security requirements are the outcome. Some of these leave attack vectors open, such as the manipulation of the sending or receiving vehicle's internal communication and computation.

Another conclusion that can be derived from these examples is related to premature assumptions about the implementation. Whilst in one case the vehicle is seen as a single computational unit that can be trusted, in another case it has to attest for its behaviour when sending out warnings. The trust zone based analysis of the same use cases however requires for a direct communication link and cryptography between the sensors and the receiving vehicle and the composition of data is moved to the receiver side. A direct result of falsely defined system boundaries typically are security requirements that are formulated against internal subsystems rather than the system at stake itself, To overcome these problems several methods for security requirements elicitation have been proposed.

A comprehensive concept for an overall security requirements engineering process is described in detail in [16]. The authors propose a 9 step approach called SQUARE (Security Quality Engineering Methodology). The elicitation of the security requirements is one important step in the SQUARE process. In [15] several concrete methods to carry out this step are compared. These methods are based on misuse cases (MC), soft systems methodology (SSM), quality function deployment (QFD), controlled requirements expression (CORE), issue-based information systems (IBIS), joint application development (JAD), feature-oriented domain analysis (FODA), critical discourse analysis (CDA) as well as accelerated requirements method (ARM). A comparative rating based on 9 different criteria is also given but none of these criteria measures the completeness of the security requirements elicited by the different methods.

A similar approach based on the integration of Common Criteria (ISO/IEC 15408) called SREP (Security Requirements Engineering Process) is described in [17]. However the concrete techniques that carry out the security requirements elicitation process are given only very broadly. A threat driven method is proposed but is not described in detail.

In [13] anti-goals derived from negated security goals are used to systematically construct threat trees by refinement of these anti-goals. Security requirements are then obtained as countermeasures. This method aims to produce more complete requirements than other methods based on misuse cases. The refinement steps in this method can be performed informally or formally.

In [4] different kinds of security requirements are identified and informal guidelines are listed that have proven useful when eliciting concrete security requirements. The author emphasises that there has to be a clear distinction between security requirements and security mechanisms.

In [9] it is proposed to use Jackson‘s problem diagrams to determine security requirements which are given as constraints on functional requirements. Though this approach presents a methodology to derive security requirements from security goals, it does not explain the actual refinements process, which leaves open, the degree of coverage of requirements, depending only on expert knowledge.

In [10–12] Hatebur et al. describe a security engineering process based on security problem frames and concretised security problem frames. The two kinds of frames constitute patterns for analysing security problems and associated solution approaches. They are arranged in a pattern system with formal preconditions and postconditions for the frames which makes dependencies between them explicit. A method to use this pattern system to analyse a given security problem and find solution approaches is described. The focus of [10] is on anonymity, while [11] focusses on confidential data transmission, and [12] addresses accountability by logging and the steps of the process.

In [14] actor dependency analysis is used to identify attackers and potential threats in order to identify security requirements. The so called i^* approach facilitates the analysis of security requirements within the social context of relevant actors. In [6] a formal framework is presented for modelling and analysis of security and trust requirements at an organisational level. Both of these approaches target organisational relations among agents rather than functional dependence. Those approaches might be utilised complementary to the one presented in this paper, as the output of organisational relations analysis may be an input to our functional security analysis.

Though all of the approaches may lead to a sufficient level of security for the designed architecture, there is no obvious means by which they can be compared regarding the security requirements that they fulfil. The choice of the appropriate abstraction level and system boundaries constitutes a rather big challenge to SoS architecture design, especially with respect to SoS applications like the one presented here.

The method described in Sect. 4 in this paper is based on the work presented in [5], whereas the tool-assisted methodology that builds on this approach presented in Sect. 5 is a new contribution of this work. We are targeting here the identification of a consistent and complete set of authenticity requirements. For an analysis of privacy-related requirements with respect to vehicular communication systems please refer to [26].

3 Vehicular Communication Systems Scenario

The derivation of security requirements in general, especially the derivation of authenticity requirements represents an essential building block for system design. With an increase in the severity of safety-relevant systems' failures the demand increases for a systematic approach of requirements derivation with a maximum coverage. Also during the derivation of security requirements, no preassumptions should be made about possible implementations. We will further motivate this with respect to the requirements derivation process with an example from the field of vehicle-to-vehicle communications.

3.1 Example Use Cases

In order to illustrate our approach we use a scenario taken from the project EVITA (E-Safety Vehicle Intrusion Protected Applications) [23]. The scenario is based on an evaluation of security relevant use cases for vehicular communication systems in which vehicles and roadside units communicate in an ad hoc manner to exchange information such as safety warnings and traffic information. Optionally, local danger warning information can also be provided to in-vehicular safety concepts for further processing.

Our example system consists of vehicles $V_1, \dots, V_n$. Each V_i has its driver D_i and is equipped with an Electronic Stability Protection (ESP) sensor ESP_i and a Global Positioning System (GPS) sensor GPS_i. Within each vehicle's on-board network, the scenario involves a communication unit (CU) CU_i for sending and receiving messages. Furthermore, a connection to a Human Machine Interface (HMI) HMI_i is required for displaying the warning message, e.g. via audio signals or on a display. Furthermore, the example system includes a roadside unit (RSU) that can send cooperative awareness messages *cam*. For simplicity reasons we assume that the same information is provided by all roadside units in the system, so we can abstract from the individual entity. Our vehicle-to-vehicle scenario is based on the following use cases:

Use case 1 A roadside unit broadcasts a cooperative awareness message.

Use case 2 A vehicle's ESP sensor recognises that the ground is very slippery when accelerating in combination with a low temperature. In order to warn successive vehicles about a possibly icy road, the vehicle uses its communication unit to send out information about this danger including the GPS position data indicating where the danger was detected.

Use case 3 A vehicle receives a cooperative awareness message, such as a warning about an icy road at a certain position, from a roadside unit or another vehicle. It compares the information to its own position and heading and signals the driver a warning if the dangerous area lies up front.

Use case 4 A vehicle receives a cooperative awareness message. It compares the information to its own position and heading and retransmits the warning, given that the position of this occurrence is not too far away.

For local danger warning applications, at least two entities are involved, namely the vehicle receiving a critical warning message and the entity sending such a message. The entity that sends out the message can be another vehicle, a roadside unit or traffic light, or an infrastructure based server. The scenario uses the actions described in table 1.

4 Functional Security Analysis

The approach described in the following can be decomposed into three basic steps. The first one is the derivation of the functional model from the use case descriptions in terms of an action oriented system. In a second step the system at stake is defined and possible instantiations of the first functional model are elaborated. In a third and final step, the actual requirements are derived in a systematic way, resulting in a consistent and complete set of security requirements.

Table 1. Actions for the example system

Action	Explanation
$send$(cam(pos))	A roadside unit broadcasts a cooperative awareness message cam concerning a danger at position pos.
$sense(ESP_i, \mathrm{sW})$	The ESP sensor of vehicle V_i senses slippery wheels (sW).
$pos(GPS_i, \mathrm{pos})$	The GPS sensor of vehicle V_i computes its position.
$send(CU_i, \mathrm{cam(pos)})$	The communication unit CU_i of vehicle V_i sends a cooperative awareness message cam concerning the assumed danger based on the slippery wheels measurement for position pos.
$rec(CU_i, \mathrm{cam(pos)})$	The communication unit CU_i of vehicle V_i receives a cooperative awareness message cam for position pos from another vehicle or a roadside unit.
$fwd(CU_i, \mathrm{cam(pos)})$	The communication unit CU_i of vehicle V_i forwards a cooperative awareness message cam for position pos.
$show(HMI_i, \mathrm{warn})$	The human machine interface HMI_i of Vehicle V_i shows its driver a warning warn with respect to the relative position.

4.1 Functional Model

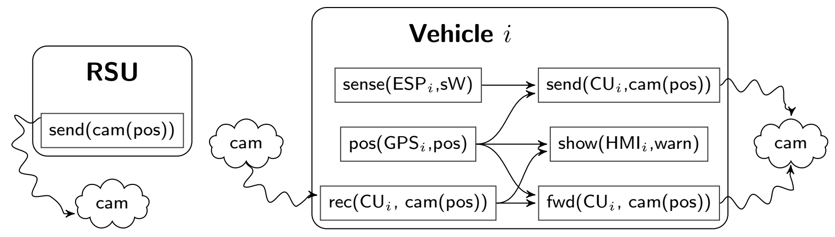
Information flow between systems and system components is highly complex, especially given that a system can evolve via the replacement of its components. Consequently, an important aspect of security evaluation is the analysis of the potential information flows. We use the analysis of the potential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

For the description of the functional model from the use cases an action-oriented approach is chosen. The approach is based on the work from [18]. For reasons of simplicity and readability the formal description of the model is omitted here and a graphical representation is used to illustrate the behaviour of the evaluation target.

A functional model can be derived from a use case description by identifying the atomic actions in the use case description. These actions are set into relation by defining the functional flow among them. This action oriented approach considers possible sequences of actions (control flow) and information flow (input/output) between interdependent actions.

In the case of highly distributed systems and especially a distributed system of distributed systems, it is very common that use cases do not cover a complete functional cycle throughout the whole system under investigation. Rather only certain components of the system are described regarding their behaviour. This must be kept in mind when deriving the functional model. In order to clarify this distinction, functional models that describe only parts of the overall system behaviour will be called *functional component model*.

Figures 1(a) and 1(b) show functional component models for a roadside unit and a vehicle respectively. These models are derived from the example use cases



(a) Roadside Unit (b) Vehicle

Fig. 1. Functional component models

given in Sect. 3.1. The functional flow arrows outside of the vehicle's boundaries refer to functional flows between different instances of the component, whilst internal flow arrows refer to flows within the same instance of the component. For the given example, the external flows represent data transmission of one system to another, whilst the internal flows represent communication within a single system.

4.2 System of Systems Instances

Based on the functional component model, one may now start to reason about the overall system of systems which consists of a number of instances of the functional components. The synthesis of the internal flow between the actions within the component instances and the external flow between systems (in this case vehicles and roadside units) builds the global system of systems behaviour. In order to model instances of the global system of systems, all structurally different combinations of component instances shall be considered. Isomorphic combinations can be neglected. Finally, all possible instances may be regrouped and the system's boundary actions (denoting the actions that are triggered by or influence the system environment) have to be identified. These will be the basis for the security requirements definition in the next step.

In Fig. 2 an example for a possible SoS instance combining use cases 1 and 3 comprising a roadside unit and a vehicle is presented. In this SoS instance vehicle V_w receives cooperative awareness message from a RSU.

4.3 Functional Security Requirements Identification

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

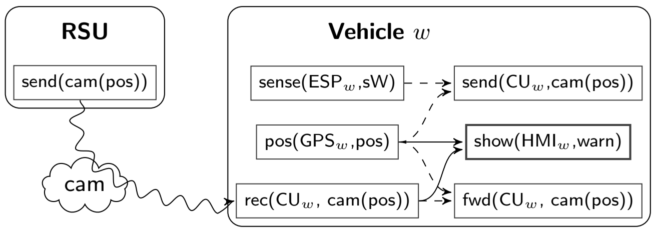


Fig. 2. Vehicle w receives warning from RSU

With the boundary actions being identified, one may now follow the functional graph backwards. Beginning with the boundary actions by which the system takes influence on the outside, we may propagate backwards along the functional flow. These backwards references basically describe the functional dependencies of actions among each other. From the functional dependency graph we may now identify the end points - the boundary actions that trigger the system behaviour that depends on them. Between these and the corresponding starting points, the requirement exists that without such an action happening as input to the system, the corresponding output action must not happen as well. From this we formulate the security goal of the system at stake:

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

Example 1 (Boundary Actions and Dependencies). In the SoS instance in Fig. 2 we are interested to identify the authenticity requirements for the boundary action $show(HMI_w, \text{warn})$. Following backwards along the functional flow we derive that the output action $show(HMI_w, \text{warn})$ is depending on the input actions $pos(GPS_w, \text{pos})$ of vehicle w and $send(\text{cam}(\text{pos}))$ of the RSU.

These dependencies shall now be enriched by additional parameters. In particular, it shall be identified which is the entity that must be assured of the respective authenticity requirements. With these additional parameters set, we may utilise the following definition of authenticity from the formal framework of Fraunhofer SIT [8] to specify the identified requirements.

Definition 1. *$auth(a, b, P)$: Whenever an action b happens, it must be authentic for an Agent P that in any course of events that seem possible to him, a certain action a has happened (for a formal definition see [8]).*

Example 2 (Derive Requirements from Dependencies). For the dependencies in Example 1 this leads to the following authenticity requirements with respect to the action $show(HMI_w, \text{warn})$:

– It must be authentic for the driver of vehicle w that the relative position of the danger he/she is warned about is based on correct position information of his/her vehicle. Formally: $auth(pos(GPS_w, \text{pos}), show(HMI_w, \text{warn}), D_w)$

– It must be authentic for the driver of vehicle w that the roadside unit issued the warning. Formally: $auth(send(\text{cam}(\text{pos})), show(HMI_w, \text{warn}), D_w)$

It shall be noted that the requirements elicitation process in this case utilises positive formulations of how the system should behave, rather than preventing a certain malicious behaviour. Also it has to be stressed that this approach guarantees for the system / component architect to be free regarding the choice of concepts during the security engineering process.

This manual analysis may reveal that certain functional dependencies are presented only for performance reasons. This can be valuable input for the architects as well, and sometimes reveals premature decisions about mechanisms that were already done during the use case definition phase.

This approach cannot prevent the specification of circular dependencies among systems' actions but usually this is avoided for well-defined use cases. This actually originates from the fact that every action represents a progress in time. Accordingly an infinite loop among actions in the system would indicate that the system described will not terminate. The requirements derivation process will however highlight every functional dependency that is described within the use cases. Accordingly, when the use case description incorporates more than the sheer safety related functional description, additional requirements may arise. Therefore, the requirements have to be evaluated towards their meaning for the system's safety. Whilst one can be assured not to have missed any safety relevant requirement, this is a critical task because misjudging a requirement's relevance would induce security holes. Once an exhaustive list of security requirements is identified, a requirements categorisation and prioritisation process can evaluate them according to a maximum acceptable risk strategy.

4.4 Formalisation

Formally, the functional flow among actions can be interpreted as an ordering relation ζ_i on the set of actions Σ_i in a certain system instance i. To derive the requirements the reflexive transitive closure ζ_i^* is constructed. In the following we assume that the functional flow graph is sequential and free of loops, as every action can only depend on past actions. Accordingly, the relation is anti-symmetric. ζ_i^* is a partial order on Σ_i, with the maximal elements max_i corresponding to the outgoing boundary actions and the minimal elements min_i corresponding to the incoming boundary actions. After restricting ζ_i^* to these elements $\chi_i = \{(x, y) \in \Sigma_i \times \Sigma_i \mid (x, y) \in \zeta_i^* \wedge x \in min_i \wedge y \in max_i\}$ this new relation represents the authenticity requirements for the corresponding system instance: *For all* $x, y \in \Sigma_i$ *with* $(x, y) \in \chi_i$: $auth(x, y, stakeholder(y))$ is a requirement. Accordingly the union of all these requirements for the different instances poses the set of requirements for the whole system. This set can be reduced by eliminating duplicate requirements or by use of first-order predicates for a parameterised notation of similar requirements.

Example 3 (Formal Derivation of Authenticity Requirements). For the given system model instances, we may now identify the authenticity requirements for the action $show(HMI_w, \text{warn})$ using the actions and abbreviations defined in table 1. Graphically, this could be done by reversing the arrows and removing the dotted arrows and boxes.

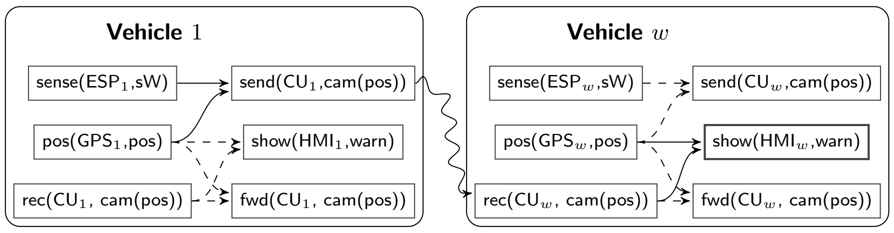


Fig. 3. Vehicle w receives a warning from vehicle 1

Figure 3 shows an example for a possible SoS instance combining use cases 2 and 3 comprising two vehicles. In this SoS instance vehicle V_w receives cooperative awareness message from vehicle V_1. Formally, for the SoS instance depicted in Fig. 3, we can analyse:

$$\begin{aligned}
\zeta_1 =&\{(sense(ESP_1, \text{sW}), send(CU_1, \text{cam(pos)})),\\
&(pos(GPS_1, \text{pos}), send(CU_1, \text{cam(pos)})),\\
&(send(CU_1, \text{cam(pos)}), rec(CU_w, \text{cam(pos)})),\\
&(pos(GPS_w, \text{pos}), show(HMI_w, \text{warn})),\\
&(rec(CU_w, \text{cam(pos)}), show(HMI_w, \text{warn}))\}\\
\zeta_1^* =&\zeta_1 \cup \{(x, x) \mid x \in \Sigma\} \cup \{\\
&(sense(ESP_1, \text{sW}), rec(CU_w, \text{cam(pos)})),\\
&(sense(ESP_1, \text{sW}), show(HMI_w, \text{warn})),\\
&(pos(GPS_1, \text{pos}), rec(CU_w, \text{cam(pos)})),\\
&(pos(GPS_1, \text{pos}), show(HMI_w, \text{warn})),\\
&(send(CU_1, \text{cam(pos)}), show(HMI_w, \text{warn}))\}\\
\chi_1 =&\{(sense(ESP_1, \text{sW}), show(HMI_w, \text{warn})),\\
&(pos(GPS_1, \text{pos}), show(HMI_w, \text{warn})),\\
&(pos(GPS_w, \text{pos}), show(HMI_w, \text{warn}))\}
\end{aligned}$$

For further analysis we consider a possible SoS instance combining use cases 2, 3 and 4 comprising three vehicles as shown in Fig. 4. In this SoS instance vehicle V_2 forwards warnings from vehicle V_1 to vehicle V_w.

An analysis of the SoS instance with 3 vehicles as depicted in Fig. 4 will result in:

$$\chi_2 = \chi_1 \cup \{(pos(GPS_2, \text{pos}), show(HMI_w, \text{warn}))\}$$

In the given SoS model the forwarding of a message is restricted by a *position based forwarding policy* with respect to the distance from the danger that is being

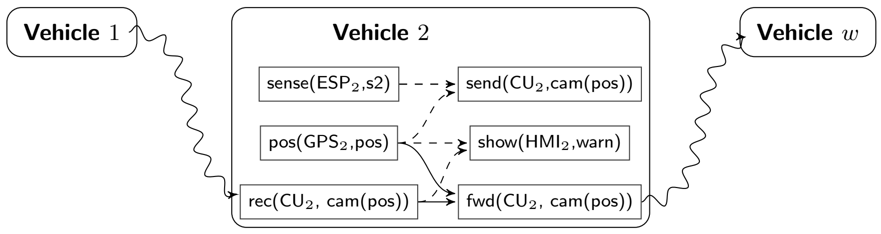


Fig. 4. Vehicle 2 forwards warnings (vehicles 1, 2 and w are instances from Fig. 1)

warned about and the time of issue of the danger sensing. We could therefore assume a maximal number of system instances involved general enough to cover all these cases, e.g. by utilising a description in a parameterised way. An analysis for an SoS instance with i vehicles will result in:

$$\chi_i = \chi_{i-1} \cup \{(pos(GPS_i, \text{pos}), show(HMI_w, \text{warn}))\}$$

The first three elements in each χ_i will obviously always be the same in all instances of the example. The rest of the elements can be expressed in terms of first-order predicates. This leads to the following authenticity requirements for all possible system instances for the action $show(HMI_w, \text{warn})$:

$$auth(pos(GPS_w, \text{pos}), show(HMI_w, \text{warn}), D_w) \tag{1}$$
$$auth(pos(GPS_1, \text{pos}), show(HMI_w, \text{warn}), D_w) \tag{2}$$
$$auth(sense(ESP_1, \text{sW}), show(HMI_w, \text{warn}), D_w) \tag{3}$$
$$\forall x \in V_{forward} : auth(pos(GPS_x, \text{pos}), show(HMI_w, \text{warn}), D_w) \tag{4}$$

$V_{forward}$ denotes the set of vehicles per system instance, that forward the warning message.

As mentioned above, the resulting requirements have to be evaluated regarding their meaning for the functional safety of the system. For the first three requirements the argumentation is very straight forward regarding why they have to be fulfilled:

1. It must be authentic for the driver that the relative position of the danger he/she is warned about is based on correct position information of his/her vehicle.
2. It must be authentic for the driver that the position of the danger he/she is warned about is based on correct position information of the vehicle issuing the warning.
3. It must be authentic for the driver that the danger he/she is warned about is based on correct sensor data.

The last requirement (4) however must be further evaluated. Studying the use case, we see that this functional dependency originates from the position based forwarding policy. This policy is introduced for performance reasons, such that bandwidth is saved by not flooding the whole network. Braking this requirement would therefore result either in a smaller or in a larger broadcasting area. As bad as those cases may be, they cannot cause the warning of a driver that should not be warned. Therefore we do not consider requirement (4) to be a safety related authenticity requirement. It can be considered a requirement regarding availability by preventing the denial of a service or unintended consumption of bandwidth.

In practice, the method described here has been applied in the project EVITA [23] to derive authenticity requirements for the development of a new automotive on-board architecture utilising vehicle-to-vehicle and vehicle-to-infrastructure communication. A total of 29 authenticity requirements have been elicited by means of a system model comprising 38 component boundary actions with 16 system boundary actions comprising 9 maximal and 7 minimal elements.

5 Tool-assisted Requirements Identification

The method for deriving authenticity requirements as described in the previous section relies on manual identification and processing only. In this section we will give an example on how to use the capabilities of existing tools, such as the SH verification tool [20] in order to facilitate the process especially for larger models.

As the previous section explained, the basis for the systematic identification of authenticity requirements for a given system is the relations between maxima and minima of the partial order of functional dependence. In this approach we first identified the direct relations of adjacent actions, then built the reflexive transitive closure and finally extracted those relations from this set that exist between maxima and minima of this partial order.

The tool-assisted approach will proceed in reverse order. First we will identify the maxima and minima of the partial order – without deriving the actual partial order – and then we will identify combinations of maxima and minima that are related by functional dependence. This approach will be illustrated with a simple example first, to provide the general idea and then with a more complex example, in order to demonstrate the application of abstraction techniques to cover the analysis of non-trivial systems.

5.1 Formal Modelling Technique

In order to analyse the system behaviour with tool support, an appropriate formal representation has to be chosen. In our approach, we choose an operational finite state model of the behaviour of the given vehicular communication scenario that is based on *Asynchronous Product Automata (APA)*, a flexible operational specification concept for cooperating systems [20]. An APA consists of a family of so called *elementary automata* communicating by common components of their state (shared memory). We now introduce the formal modelling techniques used, and illustrate the usage by our collaboration example.

Definition 2 (Asynchronous Product Automaton (APA)).
An Asynchronous Product Automaton *consists of*

- *a family of* state sets $Z_s, s \in \mathbb{S}$,
- *a family of* elementary automata $(\Phi_t, \Delta_t), t \in \mathbb{T}$ *and*
- *a* neighbourhood relation $N : \mathbb{T} \rightarrow \mathfrak{P}(\mathbb{S})$.

$\mathbb{S}$ and $\mathbb{T}$ are index sets with the names of state components and of elementary automata and $\mathfrak{P}(\mathbb{S})$ is the power set of $\mathbb{S}$.
For each elementary automaton (Φ_t, Δ_t) with Alphabet Φ_t, *its* state transition relation *is*

$$\Delta_t \subseteq \bigtimes_{s\in N(t)}(Z_s) \times \Phi_t \times \bigtimes_{s\in N(t)}(Z_s).$$

For each element of Φ_t the state transition relation Δ_t defines state transitions that change only the state components in $N(t)$. An APA's (global) states *are elements of $\bigtimes_{s\in\mathbb{S}}(Z_s)$. To avoid pathological cases it is generally assumed that $N(t) \neq \emptyset$ for all $t \in \mathbb{T}$.*
Each APA has one initial state $q_0 = (q_{0s})_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$.
In total, an APA $\mathbb{A}$ is defined by

$$\mathbb{A} = ((Z_s)_{s\in\mathbb{S}}, (\Phi_t, \Delta_t)_{t\in\mathbb{T}}, N, q_0).$$

An elementary automaton (Φ_t, Δ_t) is activated *in a state $p = (p_s)_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$ as to an* interpretation *$i \in \Phi_t$, if there are $(q_s)_{s\in N(t)} \in \bigtimes_{s\in N(t)}(Z_s)$ with $((p_s)_{s\in N(t)}, i, (q_s)_{s\in N(t)}) \in \Delta_t$.*
An activated elementary automaton (Φ_t, Δ_t) can execute a state transition and produce a successor state

$$q = (q_r)_{r\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s),\ \textit{if}$$

$$q_r = p_r \textit{ for } r \in \mathbb{S} \setminus N(t) \textit{ and } ((p_s)_{s\in N(t)}, i, (q_s)_{s\in N(t)}) \in \Delta_t.$$

The corresponding state transition is $(p, (t, i), q)$.

For the following analysis by model checking and abstraction we use a reduced version of the functional component model of a vehicle that corresponds to the functional model illustrated in Fig. 1(b) but does not contain the $forward$ action.

Example 4 (Finite State Model of the Collaboration Components). The vehicle component model described in Sect. 4.1 is specified for the proposed analysis method using the following *APA state components* for each of the vehicles:

$$\begin{aligned}
\mathbb{S}_i = \{&esp_i, gps_i, hmi_i, bus_i, net\}, \text{ with}\\
Z_{esp_i} &= \mathfrak{P}(\{\text{sW}\}),\\
Z_{gps_i} &= \mathfrak{P}(\{pos1, pos2, pos3, pos4\}),\\
Z_{hmi_i} &= \mathfrak{P}(\{\text{warn}\}),\\
Z_{bus_i} &= \mathfrak{P}(Z_{esp} \cup Z_{gps} \cup Z_{hmi})) \text{ and}\\
Z_{net} &= \mathfrak{P}(\{\text{cam}\} \times \{V_1, V_2, V_3, V_4\} \times Z_{gps}).
\end{aligned}$$

The *inputs* to the vehicle model are represented by the state components esp_i and gps_i. esp_i represents input measurements taken by the ESP sensor. A pending data set here will trigger the *sense* action for slippery wheels. gps_i represents the derivation of GPS position information. Pending data here will trigger the *pos* action for retrieving the current position of the vehicle.

The *outputs* of the vehicle model are represented by the state component hmi_i that represents the HMI interface's display, showing (warning) information to the driver. The *show* action will push information to this medium.

Internally the vehicle component has an additional state component bus_i representing its internal communication bus. It is filled with information from the *rec*, *sense* and *pos* action and read by the *send* and *forward* action.

Finally, *net* is a shared state component between all the vehicles that represents the wireless communication medium. A pending message here will trigger the *rec* action of the component. The actions *send* and *forward* will push a message into this medium.

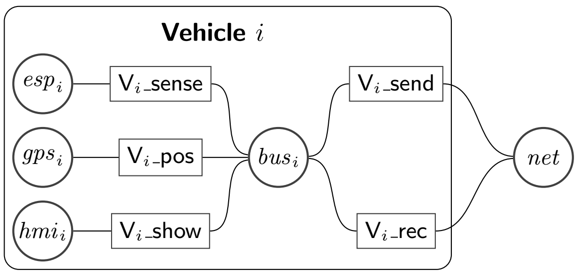


Fig. 5. APA model of a vehicle

The *elementary automata* $\mathbb{T}_i = \{V_i\_pos, V_i\_sense, V_i\_rec, V_i\_send, V_i\_show\}$ represent the possible actions that the systems can take. These specifications are represented in the data structures and initial configuration of the state components in the APA model. Elementary automata and state components of the APA model of a vehicle are depicted in Fig. 5. The lines in Fig. 5 between state components and elementary automata represent the neighbourhood relation.

The state transition relation for the APA model of a vehicle is given by:

$$\begin{aligned}
\Delta_{V_i\_sense} =& \{((esp_i, bus_i), (esp), (esp_i \backslash \{esp\}, bus_i \cup \{esp\})) \\
& \in (Z_{esp_i} \times Z_{bus_i}) \times ESP \times (Z_{esp_i} \times Z_{bus_i}) \mid esp \in esp_i\} \\
\Delta_{V_i\_pos} =& \{((gps_i, bus_i), (gps), (gps_i \backslash \{gps\}, bus_i \cup \{gps\}) \\
& \in (Z_{gps_i} \times Z_{bus_i}) \times GPS \times (Z_{gps_i} \times Z_{bus_i}) \mid gps \in gps_i\}
\end{aligned}$$

$$\begin{aligned}\Delta_{V_i\_send} =&\{((bus_i, net), (esp, gps, msg), (bus_i \backslash \{esp, gps\}, net \cup \{msg\}))\\ &\in (Z_{bus_i} \times Z_{net}) \times (ESP \times GPS \times NET) \times (Z_{bus_i} \times Z_{net}) \mid\\ &esp \in bus_i \wedge gps \in bus_i \wedge msg = (\text{cam}, gps)\}\\ \Delta_{V_i\_rec} =&\{((net, bus_i), (msg, gps, \text{warn}), (net \backslash \{msg\}, bus_i \backslash \{gps\} \cup \{\text{warn}\}))\\ &\in (Z_{net} \times Z_{bus}) \times (NET \times GPS \times HMI) \times (Z_{net} \times Z_{bus_i}) \mid\\ &msg \in net \wedge gps \in bus_i \wedge distance(msg, gps) < \text{range}\}\\ \Delta_{V_i\_show} =&\{((bus_i, hmi_i), (\text{warn}), (bus_i \backslash \{\text{warn}\}, hmi \cup \{\text{warn}\}))\\ &\in (Z_{bus_i} \times Z_{hmi_i}) \times HMI \times (Z_{bus_i} \times Z_{hmi_i}) \mid \text{warn} \in bus_i\}\end{aligned}$$

The model is parameterised by i except for the shared state component *net*.

5.2 Formal Representation of System of Systems Instances

The SoS instance that we investigate first includes two vehicle components that are assumed to be within the wireless transmission range similar to the example given in Fig. 3. In this SoS instance vehicle V_2 receives cooperative awareness message from vehicle V_1. Therefore the *net* components are mapped together, such that outputs of each one of the vehicles are input for the other vehicle. The rest of the inputs (Sensors and GPSs) as well as outputs (displays) are not internal parts of the system but filled and read by the systems environment. It should be noted that timing behaviour is not included in the model, because we solely want to retrieve functional dependencies. As we want to instantiate V_1 to perform use Case 2 and V_2 to perform use Case 3, we set

- V_1's sensor input to a measurement of slippery wheels sW,
- V_1's GPS input to some position pos1 that is within warning range of V_2 and
- V_2's GPS input to some position pos2 that is within warning range of V_1.

Example 5 (Finite State Model of an SoS Instance with 2 Vehicles).
The state components for this instance are

$$\mathbb{S} = \{esp_1, pos_1, bus_1, hmi_1, esp_2, pos_2, bus_2, hmi_2, net\}$$

and the set of elementary automata is

$$\begin{aligned}\mathbb{T} = \{&V_1\_sense, V_1\_pos, V_1\_send, V_1\_rec, V_1\_show,\\ &V_2\_sense, V_2\_pos, V_2\_send, V_2\_rec, V_2\_show\}.\end{aligned}$$

The neighbourhood relation $N(t)$ can be read directly from the graphical illustration in Fig. 6. The initial state for our simulation is defined as:

$$\begin{aligned}q_0 =&(q_{0\_esp_1}, q_{0\_gps_1}, q_{0\_bus_1}, q_{0\_hmi_1}, q_{0\_esp_2}, q_{0\_gps_2}, q_{0\_bus_2}, q_{0\_hmi_2}, q_{0\_net})\\ =&(\{\text{sW}\}, \{\text{pos1}\}, \emptyset, \emptyset, \emptyset, \{\text{pos2}\}, \emptyset, \emptyset, \emptyset).\end{aligned}$$

For example $(q_0, (V_1\_sense, \text{sW}), (\emptyset, \{\text{pos1}\}, \{\text{sW}\}, \emptyset, \emptyset, \{\text{pos2}\}, \emptyset, \emptyset, \emptyset))$ is a state transition of this SoS instance.

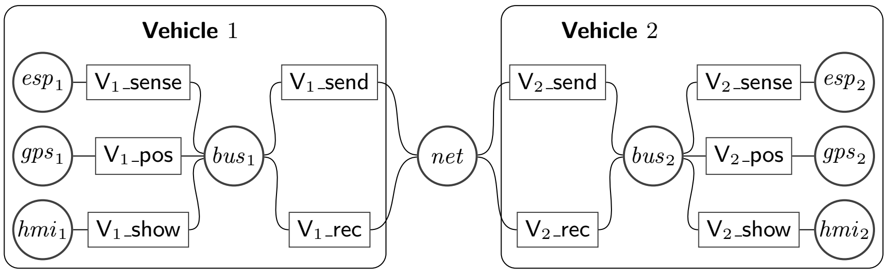


Fig. 6. APA model of a SoS instance with 2 vehicles

5.3 Computation of System of Systems Behaviour

Formally, the behaviour of our operational APA model of the vehicular communication system is described by a reachability graph. In the literature this is sometimes also referred to as labelled transition system (LTS).

Definition 3 (Reachability graph).
The behaviour of an APA is represented by all possible coherent sequences of state transitions starting with initial state q_0. *The sequence* $(q_0, (t_1, i_1), q_1)$ $(q_1, (t_2, i_2), q_2)$ $\ldots (q_{n-1}, (t_n, i_n), q_n)$ *with* $i_k \in \Phi_{t_k}$ *represents one possible sequence of actions of an APA.*

State transitions $(p, (t, i), q)$ *may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA:* $(p, (t, i), q)$ *is the edge leading from* p *to* q *and labelled by* (t, i). *The subgraph reachable from the node* q_0 *is called the* reachability graph *of an APA.*

We used the *Simple Homomorphism (SH) verification tool* [20] to analyse the functional component model for different concrete instantiations of the model. The tool has been developed at the *Fraunhofer-Institute for Secure Information Technology*. The applied specification method based on Asynchronous Product Automata is supported by this tool. The tool manages the components of the model, allows to select alternative parts of the specification and automatically *glues* together the selected components to generate a combined model of the APA specification. It provides components for the complete cycle from formal specification to exhaustive validation as well as visualisation and inspection of computed reachability graphs and minimal automata. The tool provides an editor to define homomorphisms on action languages, it computes corresponding minimal automata [3] for the homomorphic images and checks simplicity of the homomorphisms. If it is required to inspect some or all paths of the graph to check for the violation of a security property, as it is usually the case for liveness properties, then the tool's temporal logic component can be used. Temporal logic formulae can also be checked on the abstract behaviour (under a simple homomorphism). The method for checking approximate satisfaction of properties fits exactly to the built-in simple homomorphism check [20].

Computation of SoS Instance's Behaviour. Starting with the analysis, we define a representation of the component behaviour in preamble language of the SH verification tool according to the use cases. Then for a first simple example, we instantiated it twice – with a warning vehicle V_1 and a vehicle that receives the warning V_2, similar to Fig. 6. After an initial state is selected, the reachability graph is automatically computed by the SH verification tool. Fig. 7 shows the reachability graph resulting from the analysis of the model instance in Fig. 6. Please note that the tool prints the state q_0 as M-1.

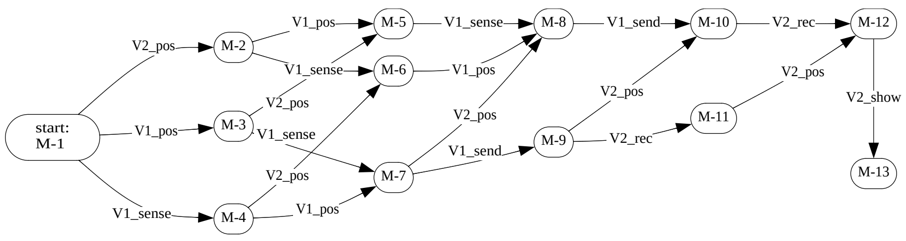


Fig. 7. Reachability graph of SoS instance with two vehicles in the SH verification tool

5.4 Evaluating the Functional Dependence Relation

Starting from the model of the system components and their instantiations the reachability analysis provides a graph with serialised traces of actions in the system. In order to identify the *minima* of such a system, we look at the initial state *M-1* of the reachability graph. Every action that leaves the initial state on any of the traces is obviously a minimum, because it does not functionally depend on any other action to have occurred before. In order to identify the *maxima* we investigate those actions leading to the dead state from any trace. These actions do not trigger any further action after they have been performed.

Example 6 (The SH verification tool's result for Example 5).

```
The minima of this analysis:        The corresponding maxima:
M–1
 V1_sense  M–4                      M–12  V2_show
 V1_pos  M–3                        M–13+
 V2_pos  M–2                        +++  dead  +++
```

Since we now have identified the maxima and minima of the partial order of functionally dependent actions, we must evaluate which of these maxima have a functional dependence relation. For this simple example, it can easily be seen from the reachability graph, that the maximum only occurs after all the minima have occurred in each of the traces, i.e. the maximum depends on all the identified minima. Accordingly, the simple example has the following set of requirements: $auth(V_1\_sense, V_2\_show, D_2)$, $auth(V_1\_pos, V_2\_show, D_2)$, $auth(V_2\_pos,\ V_2\_show, D_2)$.

5.5 Abstraction Based Verification Concept

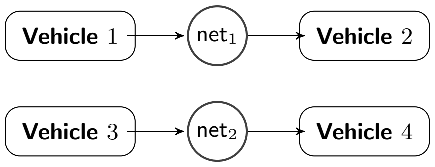


Fig. 8. Model for SoS instance with four vehicles

In order to further demonstrate our approach for a more complex scenario, a second example of a SoS instance that includes four vehicles – two pairs of two vehicles, each pair within communication range but out of range from the other pair, performing the same scenario each (V_1 warns V_2 and V_3 warns V_4) – can be seen in Fig. 8 with the corresponding reachability graph in Fig. 9.

The minima of this analysis:	The corresponding maxima:
M–1	
V1_sense M–7	M–168 V2_show
V3_sense M–6	M–167 V4_show
V1_pos M–5	M–169+
V2_pos M–4	+++ dead +++
V3_pos M–3	
V4_pos M–2	

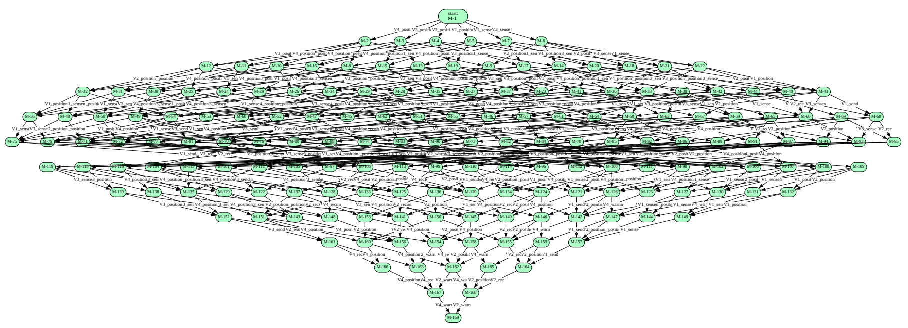


Fig. 9. Reachability graph of SoS instance with four vehicles in the SH verification tool

Obviously, the reachability graph in Fig. 9 that is generated from the complex scenario cannot be evaluated directly. However the technique of abstraction can help us to identify if a given maximum functionally depends on a given minimum.

Behaviour abstraction of an APA can be formalised by language homomorphisms, more precisely by alphabetic language homomorphisms $h : \Sigma^* \rightarrow \Sigma'^*$. By these homomorphisms certain transitions are ignored and others are renamed, which may have the effect, that different transitions are identified with one another. A mapping $h : \Sigma^* \rightarrow \Sigma'^*$ is called a language homomorphism if $h(\varepsilon) = \varepsilon$ and $h(yz) = h(y)h(z)$ for each $y, z \in \Sigma^*$. It is called alphabetic, if $h(\Sigma) \subset \Sigma' \cup \{\varepsilon\}$.

In order to analyse dependencies for each pair of maximum and minimum in the graph in Fig. 9, we can now define alphabetic language homomorphisms that will map every action except the given pairs of maximum and minimum to ε. The computed abstract representations then provide a visualisation focussing on the two actions, helping us to see directly, if the given maximum can occur independent of the given minimum or if it depends on the minimum's prior occurrence.

Example 7. The minimal automaton computed from the reachability graph under the homomorphism that preserves V_1_sense and V_2_show is depicted in Fig. 10. This graph indicates a functional dependence relation between the given maximum and minimum.

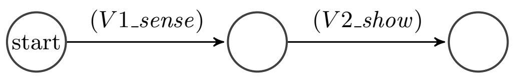


Fig. 10. Minimal automaton with maximum and minimum

The homomorphism preserving V_1_sense and V_4_show will result in the graph depicted in Fig. 11 that indicates the given maximum and minimum not to have a functional dependence relation.

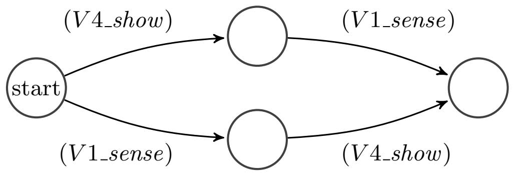


Fig. 11. Minimal automaton with independent maximum and minimum

Following this approach, testing each of the maxima with each of the minima for functional dependence, the complex scenario has the following set of requirements (with the stakeholder of V_4 to be driver D_4 of course):

$auth(V_1\_sense, V_2\_show, D_2)$, $auth(V_1\_pos, V_2\_show, D_2)$,
$auth(V_2\_pos, V_2\_show, D_2)$, $auth(V_3\_sense, V_4\_show, D_4)$,
$auth(V_3\_pos, V_4\_show, D_4)$, $auth(V_4\_pos, V_4\_show, D_4)$.

6 Conclusion

The presented approach for deriving safety-critical authenticity requirements in SoS solves several issues compared to existing approaches. It incorporates a clear scheme that will ensure a consistent and complete set of security requirements. Also it is based directly on the functional analysis, ensuring the safety of the system at stake. The systematic approach that incorporates formal semantics leads directly to the formal validation of security, as it is required by certain evaluation assurance levels of Common Criteria (ISO/IEC 15408). Furthermore the difficulties of designing SoS are specifically targeted.

Starting from this set of very high-level requirements, the security engineering process may proceed. This will include decisions regarding the mechanisms to be included. Accordingly the requirements have to be refined to more concrete requirements in this process. The design and refinement process may reveal further requirements regarding the internals of the system that have to be addressed as well.

Future work may include the derivation of confidentiality requirements in a similar way as was presented here. Though this will require for different security goals, as confidentiality is not related to safety in a similar way, but rather to privacy. Non-Repudiation may also be a target that should be approached in cooperation with lawyers in order to find the relevant security goals. Furthermore, the refinement throughout the design process should be evaluated regarding possibility of formalising it in schemes with respect to the security requirements refinement process.

For the tool-assisted method in Sect. 5, traditional model checking techniques allow a verification of SoS behaviour only for systems with very few components. We are developing an abstraction based approach to extend our current tool supported verification techniques to such families of systems that are usually parameterised by a number of replicated identical components. In [19] we demonstrated our technique by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario. In [21] we defined uniform parameterisations of phase based cooperations in terms of formal language theory. For such systems of cooperations a kind of self-similarity is formalised. Based on deterministic computations in shuffle automata a sufficient condition for self-similarity is given. Under certain regularity restrictions this condition can be verified by a semi-algorithm. For verification purposes, so called uniformly parameterised safety properties are defined. Such properties can be used to express privacy policies as well as security and dependability requirements. It is shown, how the parameterised problem of verifying such a property is reduced by self-similarity to a finite state problem.

Acknowledgments. Andreas Fuchs developed the work presented here in the context of the project EVITA (ID 224275) being co-funded by the European Commission within the Seventh Framework Programme. Roland Rieke developed the work presented here in the context of the projects Alliance Digital Product Flow (ADiWa) (ID 01IA08006F) and VOGUE (ID 01IS09032A) which are both funded by the German Federal Ministry of Education and Research.

References

1. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Sec. Comput. 1(1), 11–33 (2004)
2. Bodeau, D.J.: System-of-Systems Security Engineering. In: In Proc. of the 10th Annual Computer Security Applications Conference, Orlando, Florida. pp. 228–235. IEEE Computer Society (1994)
3. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, New York (1974)
4. Firesmith, D.: Engineering security requirements. Journal of Object Technology 2(1), 53–68 (2003)
5. Fuchs, A., Rieke, R.: Identification of authenticity requirements in systems of systems by functional security analysis. In: Workshop on Architecting Dependable Systems (WADS 2009), in Proceedings of the 2009 IEEE/IFIP Conference on Dependable Systems and Networks, Supplementary Volume (2009), http://sit.sit.fraunhofer.de/smv/publications/
6. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Requirements engineering meets trust management: Model, methodology, and reasoning. In: In Proc. of iTrust 04, LNCS 2995. pp. 176–190. Springer-Verlag (2004)
7. Group, T.C.: TCG TPM Specification 1.2 revision 103. www.trustedcomputing.org (2006)
8. Gürgens, S., Ochsenschläger, P., Rudolph, C.: Authenticity and provability - a formal framework. In: Infrastructure Security Conference InfraSec 2002. Lecture Notes in Computer Science, vol. 2437, pp. 227–245. Springer Verlag (2002)
9. Haley, C.B., Laney, R.C., Moffett, J.D., Nuseibeh, B.: Security requirements engineering: A framework for representation and analysis. IEEE Trans. Software Eng. 34(1), 133–153 (2008)
10. Hatebur, D., Heisel, M., Schmidt, H.: A security engineering process based on patterns. In: Proceedings of the International Workshop on Secure Systems Methodologies using Patterns (SPatterns), DEXA 2007. pp. 734–738. IEEE Computer Society (2007), http://www.ieee.org/
11. Hatebur, D., Heisel, M., Schmidt, H.: A pattern system for security requirements engineering. In: Proceedings of the International Conference on Availability, Reliability and Security (AReS). pp. 356–365. IEEE (2007), http://www.ieee.org/
12. Hatebur, D., Heisel, M., Schmidt, H.: Analysis and component-based realization of security requirements. In: Proceedings of the International Conference on Availability, Reliability and Security (AReS). pp. 195–203. IEEE Computer Society (2008), http://www.ieee.org/
13. van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: ICSE '04: Proceedings of the 26th International Conference on Software Engineering. pp. 148–157. IEEE Computer Society, Washington, DC, USA (2004)

14. Liu, L., Yu, E., Mylopoulos, J.: Analyzing security requirements as relationships among strategic actors. In: 2nd Symposium on Requirements Engineering for Information Security (SREIS'02) (2002)
15. Mead, N.R.: How To Compare the Security Quality Requirements Engineering (SQUARE) Method with Other Methods . Tech. Rep. CMU/SEI-2007-TN-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2007)
16. Mead, N.R., Hough, E.D.: Security requirements engineering for software systems: Case studies in support of software engineering education. In: CSEET '06: Proceedings of the 19th Conference on Software Engineering Education & Training. pp. 149–158. IEEE Computer Society, Washington, DC, USA (2006)
17. Mellado, D., Fernández-Medina, E., Piattini, M.: A common criteria based security requirements engineering process for the development of secure information systems. Comput. Stand. Interfaces 29(2), 244–253 (2007)
18. Ochsenschläger, P., Repp, J., Rieke, R.: Abstraction and composition – a verification method for co-operating systems. Journal of Experimental and Theoretical Artificial Intelligence 12, 447–459 (June 2000), `http://sit.sit.fraunhofer.de/smv/publications/`, copyright: ©2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.
19. Ochsenschläger, P., Rieke, R.: Abstraction based verification of a parameterised policy controlled system. In: International Conference "Mathematical Methods, Models and Architectures for Computer Networks Security" (MMM-ACNS-7). CCIS, vol. 1. Springer (September 2007), `http://sit.sit.fraunhofer.de/smv/publications/`, Springer
20. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing, The International Journal of Formal Method 11, 1–24 (1999)
21. Ochsenschläger, P., Rieke, R.: Uniform parameterisation of phase based cooperations. Tech. Rep. SIT-TR-2010/1, Fraunhofer SIT (2010), `http://sit.sit.fraunhofer.de/smv/publications/`
22. Papadimitratos, P., Buttyan, L., Hubaux, J.P., Kargl, F., Kung, A., Raya, M.: Architecture for Secure and Private Vehicular Communications. In: IEEE International Conference on ITS Telecommunications (ITST). pp. 1–6. Sophia Antipolis, France (June 2007)
23. Ruddle, A., Ward, D., Weyl, B., Idrees, S., Roudier, Y., Friedewald, M., Leimbach, T., Fuchs, A., Grgens, S., Henniger, O., Rieke, R., Ritscher, M., Broberg, H., Apvrille, L., Pacalet, R., Pedroza, G.: Security requirements for automotive onboard networks based on dark-side scenarios. EVITA Deliverable D2.3, EVITA project (2009), `http://evita-project.org/deliverables.html`
24. Sadeghi, A.R., Stüble, C.: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: NSPW '04: Proceedings of the 2004 workshop on New security paradigms. pp. 67–77. ACM, New York, NY, USA (2004)
25. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th USENIX Security Symposium. USENIX Association (2004)
26. Schaub, F., Ma, Z., Kargl, F.: Privacy requirements in vehicular communication systems. In: IEEE International Conference on Privacy, Security, Risk, and Trust (PASSAT 2009), Symposium on Secure Computing (SecureCom09). Vancouver, Canada (08/2009 2009), `http://doi.ieeecomputersociety.org/10.1109/CSE.2009.135`
27. Shirey, R.: Internet Security Glossary, Version 2. RFC 4949 (Informational) (Aug 2007), `http://www.ietf.org/rfc/rfc4949.txt`