Sound Development of Secure Service-based Systems*

Martin Deubler Johannes Grünbauer† Jan Jürjens Guido Wimmel

Technische Universität München
Institut für Informatik
Boltzmannstrasse 3
85748 Garching, Germany

http://www4.in.tum.de/~(deubler|gruenbau|juerjens|wimmel)

ABSTRACT

Service-based software systems are a useful concept recently developed to support the development of systems offering functions (the so-called *services*) which may be interrelated or may mutually depend on each other. Although appealing from a practical point of view, the development of service-based software for security-critical systems is, unfortunately, not well understood. Services may easily interact with each other in a way which may have unforeseen consequences on the various security properties provided. In this work, we propose a method for facilitating the development of security-critical service-based software systems using the computer-aided systems engineering tool AUTOFOCUS based on the formal method FOCUS. We explain our method at the example of a service-based system from the automotive domain.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering (CASE); D.2.11 [**Software Architectures**]: Domain-specific architectures

General Terms

Design

Keywords

Service-based Systems, Security, Model-based Software Engineering, AutoFocus, UML, Automotive.

*This work was supported by the Bavarian high-tech funding program (High-Tech Offensive), the BMW Car IT GmbH and the Elektroniksystem- und Logistik-GmbH (ESG) within the project MEwaDis [19].

†Contact author.

ICSOC'04, November 15–19, 2004, New York, New York, USA.

1. INTRODUCTION

The concept of *service-based software systems* has been recently developed to support the development of systems offering *services* which are functions that may be interrelated or may mutually depend on each other.

Security requirements have become an increasingly important issue in developing distributed systems, especially in the electronic business sector. Because of the fact that failures of security mechanisms may cause very high potential damage (e.g., loss of money through fraud), the correctness of such systems is crucial. Apart from confidentiality, meaning that sensitive information should not be leaked to unauthorized parties, and integrity, meaning that unauthorized parties should not be able to modify sensitive information, the most important security requirements are authentication, meaning that the participants in the system are correctly identified, and authorization, meaning the correct process of granting participants access to certain sensitive privileges. All of these security requirements have to withstand attacks from motivated adversaries, which makes them notoriously hard to enforce in software-based systems.

Although appealing from a practical point of view, the development of service-based software for security-critical systems is, unfortunately, not well understood. Services may easily interact with each other in a way which may have unforeseen consequences on the various security properties provided.

Therefore, the consideration of security requirements has to be integrated into general systems development. Following the idea of model-driven service composition [21], common modeling techniques used in industry, such as collaboration diagrams, state charts and message sequence charts may be tailored for that purpose.

In this work, we propose a method for facilitating the development of security-critical service-based software systems using the computer-aided systems engineering tool AUTOFOCUS [5] based on the formal method FOCUS [6]. The goals of our contribution in this paper are the correct development of secure service-based system.

We explain our method at the example of a service-based system from the automotive domain. This work is done in the context of the project MEwaDis funded by the Bavarian regional government. The primary goal of the project is the development of techniques for the analysis, modeling and validation of reliable, adaptive, context-aware services, and of process models for their development. The results are prototypically implemented in the domain of automotive

systems.

We specify security-critical service-based software systems using state transition diagrams (STDs, similar to UML state charts). The composition of the various services can then be automatically checked for security weaknesses using the model checker SMV connected to AutoFocus to verify that the overall system provides the desired security properties. The AutoFocus tool integrates several analysis tools, facilitating adoption in industry. Since the AutoFocus tool builds on the formal development method Focus [6], our approach also supports formal proofs in this framework.

This work is a continuation of that in [12] which considered the composition of secure services in a specific technical situation, namely that of layered protocols. Here, we move up in levels of abstraction, so that "service" is now not necessarily a specific feature of a protocol, but more generally a functionality of a system sufficiently important to be considered in its own right. The current work is based on a formal model for service-based systems proposed in [4] and can be applied in the context of a model-based development process for service-based systems proposed in [8].

While there has been successful work on integrating security into service-oriented architectures (including [17, 24]), mostly regarding web-services, our work then aims to provide a methodological approach to software-engineering targeted to service-oriented architectures, in a general context (in fact, our case-study given here is taken not from the web application domain, but from the automotive software domain).

This paper is structured as follows. In Sect. 2, we explain general concepts, what we mean by service-based systems in more detail, and how we specify and analyze security requirements in this context. To apply our formal notation of a service to a model, we make use of the tool AutoFocus, which we introduce in Sect. 3. In Sect. 4, we give an overview over the automotive application under consideration, specify a security-critical part and carry out a security analysis. After discussing related work, we end with a conclusion and indicate further planned work.

The main contribution of this work is a soundly-based methodology for the correct development of secure service-based systems. For didactic purposes, our main insights and recommendations with respect to this methodology are, after being introduced in a general setting in Sect. 2.3, explained in more detail in the context of a case-study in Sect. 4, because we consider this to be more illustrative than an abstract discussion. Note that the results are not restricted to service-based systems in the automotive domain but apply to service-based systems in general, as explained in Sect. 2.1. Also note that our approach is tool-independent. We chose a particular tool, AutoFocus, again only to make the presentation more concrete. The approach can be easily adapted to similar notations, such as UML.

2. SERVICES AND SECURITY

2.1 Services

In general services are considered as system sub-functions. They determine the patterns of behaviors to use a system for specific purposes. Thus, a practical formalization of services are patterns of interactions [4]. The formal specification method Focus allows services to be specified in terms of relations on sequences of messages [6].

Figure 1: The Service Modeling Unit

In the development of service-based systems, services are used as abstract modeling units. They represent not only sub-functions encapsulating certain behaviors. A service is also well suited to handle multiple function interactions and function dependencies, which are typical for so-called *multifunctional systems* [8]. Function dependencies are abstracted by interrelations of services.

Service modeling takes place at an early stage in the development process. At this stage neither the system's structure (component architecture) nor the internal structure of the sub-functions matters, yet. The main focus is the interaction of the system sub-functions and their (black-box) behavior. Once services are identified and set into relation, input messages, output messages (see Fig. 1) and requirements on the black-box behavior describe a service [6].

The behavior of a service is specified formally in an abstract way by relating its inputs and outputs. In addition, services can have a local state, represented by a number of local variables.

A service has a syntactic interface which is induced by the service's input and output message types. Requirements on the system's behavior can be mapped straightforwardly to service behaviors, since it is sufficient to partially specify the service interface. A service's behavior can be defined only for valid sequences of messages (so-called *service domain*, cf. [4]). That is, only the relevant behavior needs to be specified.

Interacting services are modeled via connecting channels. That way, they build up an execution scenario, which can be used as a basis for behavioral verification of security or safety properties.

2.2 Services in a Practical Context

Services are not only an abstract modeling unit, but also have a high relevance in practice. Also, service-based systems are not restricted to web-services. To explain how the general methodology proposed in this paper lends itself to application in a concrete practical context, we explain a concrete practical scenario for service-based systems in the automotive domain in this section. It is taken from the industrial application project that provided the motivation for our research.

In the MEwaDis project, the modeled prototypes are also implemented in a real environment using the Java language together with the OSGi (Open Service Gateway Initiative, see [20]) service framework. OSGi is a specification for a Java-based middleware for heterogeneous networks (like the different bus systems in vehicles), to manage services dynamically. The OSGi alliance itself is an independent and non-profit cooperation defining and promoting open specifications for the delivery and management of services to all types of networked devices in environments like home or vehicles.

To bridge the gap from our formal service model to the OSGi implementation, we briefly describe how they relate to

each other. To get services running in the OSGi framework, they have to be encapsulated in so-called *bundles*, which basically are Java archive (JAR) files. A bundle contains a *Manifest*-file, which describes the import and export interfaces. This defines the syntactical interface. The import interfaces specify the needed services, the export interfaces the provided services. Furthermore, a bundle contains a `BundleActivator` class, which registers and deregisters the services to the framework according to the Manifest description.

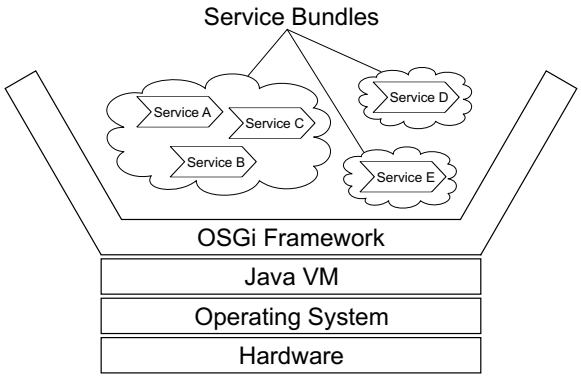


Figure 2: OSGi software platform

The notion of an interface is equivalent to our model, where the typed input and output channels also define the interface of the service. The behavioral specification of services in the formal model, which is done for example with automatons, is coded in Java to run within the OSGi framework. With the modeling tool AutoFocus (see Sect. 3) we are able to finally build OSGi bundles via code generation from the modeled service architecture and use them inside the framework. To get a strict mapping from our formal model to the OSGi services, we generate the code in a way so that we get only one service per bundle.

Fig. 2 illustrates the system structure for the OSGi framework. On the top of the operation system (OS) a Java virtual machine is running, which hosts the OSGi framework. For embedded systems, like in the automotive domain, the OS is a real time operating system in general. Finally, the OSGi framework hosts and runs the bundled services.

As a real-life-application, the German car manufacturer BMW presented an "OSGi-enabled" vehicle at the OSGi World Congress 2003 (see [22]). In fact, it is possible to run the services which we specify, model and which finally code is generated in a real car. In the *Automotive Systems Lab* at the Technische Universität München we run an embedded platform which controls the different bus systems like MOST (Media Oriented Systems Transport) or CAN (Controller Area Network). Connecting multimedia devices and devices like haptic controller, ignition lock simulator to the embedded platform, it is possible to run our services in a car-like environment.

Hence, services indeed play a role in a practical context. To put the development of secure services on a firm footing, we provide in this paper a formal approach to secure the modeling of services at an early stage of development.

2.3 Security

It is necessary to consider security as early as possible in the development process and not to treat it as an afterthought. [8] describes a methodology to develop service based systems, in which the consideration of security aspects is integrated into the development process.

The process consists of the following service-specific phases: a *service identification phase*, where services are determined as actors in activity diagrams; a *use case modeling* phase where the flows of events of the service functions are specified in form of use case models; and a *service modeling phase* where the behavior of the services is specified formally as described above. A *component design phase* follows where services are mapped to system components. The consideration of security requirements starts with the use case modeling phase where security objectives are annotated in a structured way to the use cases. These security objectives are then mapped to requirements on the formal models of execution scenarios.

In this paper, we concentrate on security requirements in the service modeling phase, in particular on enforcing access control resp. authorization. We leave out the consideration of the trustworthiness of the communication channels, since this is not the focus of this paper. However, it could be integrated into our model using the methods of [29].

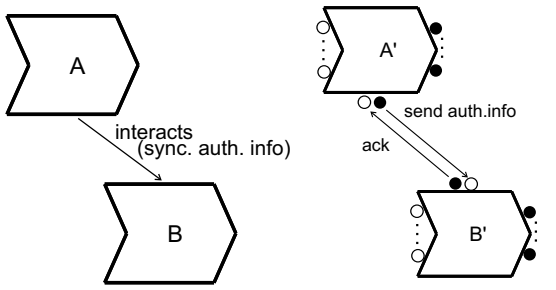
The goals of the contributions here are thus to provide a method for the correct development of secure service-based system. More specifically, here we concentrate on enforcing the two main important security mechanisms of authentication and authorization in the context of service-based systems. This is done using a notation that is close to the industrial modeling notation UML, but is soundly based on a formal foundation, which allows automated proofs that the required security properties are actually provided by a given design.

An important characteristic of service-based systems is that the information for access control decisions can be distributed over multiple services (reflected by the $s_1, \ldots, s_n$ in the authentication predicate below). Enforcing access control is very difficult in such a setting. In a formal model, possible violations can be found using verification techniques such as model checking.

Regarding the authorization requirement the following security specific development activities are integrated into the service modeling phase:

1. Specify interactions between services.
2. Threat analysis, that is, identify all services (i.e. service variables) which are involved by the authentication process.
3. Build the authorization predicate and perform model checking. For model checking, we convert the predicate into a temporal logic expression in CTL (cf. Sect. 4.4).
4. If there are any errors regarding the last step find necessary service dependencies and modify the communication links and the specifications of the service behavior accordingly to satisfy the authorization predicate.

Note, here we are not interested in how the authentication is actually performed. We rather express the fact of successfully authenticated users by describing the necessary system states. These system states characterize the consistency of identification information, which is crucial for authorization



(a) Introduced Dependency (b) Corresponding Execution Scenario Transformation

Figure 3: Pattern for Synchronizing Authentication Information

decisions. The following predicates, an authentication predicate and an authorization predicate, respectively, are used to specify parts of the system's security policy:

Let $t_{auth} = (s_1.v_1, \ldots, s_n.v_m)$ be a tuple of the local variables of the services $s_1, \ldots, s_n$, that are involved in the authentication process. $\phi(s_1.v_1, \ldots, s_n.v_m)$ is a valuation of t_{auth}, that is, the tuple of values of the variables $v_1, \ldots, v_m$ at a unique time. M_{user} is a certain subset of all valuations of t_{auth}:
$M_{user} = \{\phi : \ \phi \text{ correct valuation for authenticated } user\}$.

A user is authenticated by the services, if the *authentication predicate*

$$\mathsf{authenticated}(user) \Rightarrow \phi(s_1.v_1, \ldots, s_n.v_m) \in M_{user} \qquad (1)$$

holds, which is the prerequisite for gaining access to certain (i.e. critical) operations.

A service grants access to a service function $s.f$ if the *authorization predicate*

$$\mathsf{use}(s.f) \Rightarrow \mathsf{authenticated}(user) \qquad (2)$$

holds. This means that the service function $s.f$ can only be used if *user* is authenticated correctly.

Identifying the security related services for the authentication process is non trivial. However, there exist some requirements which may help to succeed. Remember that in service based development data is distributed over a number of services. Therefore one should have a look at services, which deal with information, that should be kept consistent (e.g. like user information). Another hint for identifying security related services are those which may perform critical operations like reading from or writing to a database.

As a design pattern for the enforcement of distributed access control decisions in service-based systems, we introduce additional dependencies between the security relevant services and possibly extend the corresponding state space. The additional dependencies ensure that the access control decision is indeed based on the complete required authentication information.

The pattern applied in our case is illustrated in more detail in Fig. 3. Fig. 3(a) shows the additional dependency introduced, signifying that service A is responsible for distributing authentication information to service B. The dependency leads to a transformed execution scenario with two additional communication channels between the services and a modification of their behavior (see Fig. 3(b)). The behavior of A is changed such that each time it receives new authentication information, it propagates this information to B and waits for an acknowledgment (service A′). Conversely, B must be prepared to receive such requests, update its information accordingly and send an acknowledgment (service B′).

In a model where the described pattern has been applied, verification of the authorization predicate (2) can be simplified by splitting it into two properties: one predicate stating that service s acts correctly based on the information for access control that is locally available, and one predicate stating that the information for access control is synchronized correctly. Here, the second predicate is a result of the correct application of the pattern. In addition, it must be ensured that access control information is not used when the system is in a critical state where the authentication information could be inconsistent, e.g. during a synchronization.

Let $t_{auth;s} = (s.v_1, \ldots, s.v_j)$ be the local variables in t_{auth} belonging to the service s (with $s \in \{s_1, \ldots, s_n\}$) and $M_{user;s}$ be the set of projections of the tuples in M_{user} to tuples of the values of $s.v_1, \ldots, s.v_j$. A user is locally authenticated in s if the local authentication predicate

$$\mathsf{authenticated}_s(user) \Rightarrow \phi(s.v_1, \ldots, s.v_j) \in M_{user;s} \qquad (3)$$

holds.

Now, we split the authorization predicate (2) into a local authorization predicate

$$\mathsf{use}(s.f) \Rightarrow \mathsf{authenticated}_s(user) \wedge \neg\mathsf{criticalstate} \qquad (4)$$

and a synchronization predicate

$$\neg\mathsf{criticalstate} \wedge \mathsf{authenticated}_s(user) \Rightarrow \mathsf{authenticated}(user) \qquad (5)$$

From (4) and (5) logically follows (2). Note we have to exclude critical states by adding ¬criticalstate to the formulas. The predicate to be used for critical states depends on the actual application of the pattern. See Section 4 for more detail with respect to our case study.

3. AUTOFOCUS

For modeling and verification of service based systems, we use the tool AutoFocus [13, 25]. AutoFocus is a CASE tool for graphically specifying distributed systems. It is based on the formal method Focus [6], and its models have a simple, formally defined semantics. AutoFocus offers standard, easy-to-use description techniques for an end-user who does not necessarily need to be a formal methods expert, as well as state-of-the-art techniques for validation and verification.

Systems are specified in AutoFocus using static and dynamic views, which are conceptually similar to those offered in UML-RT, a UML profile for component-based communicating systems. AutoFocus has been used and adapted to model security-critical systems in a number of case studies (see e.g. [12, 16, 29]).

To specify systems, AutoFocus offers the following views:

- **System Structure Diagrams (SSDs)** are similar to data flow resp. collaboration diagrams and describe the structure and the interfaces of a system. In the SSD view, a system consists of a number of communicating

components, which have input and output ports (denoted as empty and filled circles) to allow for receiving and sending messages of a particular data type. The ports can be connected via channels, making it possible for the components to exchange data. SSDs can be hierarchical, i.e. a component belonging to an SSD can have a substructure that is defined by an SSD itself. Besides, the components in an SSD can be associated with local variables.

- **Data Type Definitions (DTDs)** specify the data types used in the model, with the functional language Quest [23]. In addition to basic types as integer, user-defined hierarchic data types are offered that are very similar to those used in functional programming languages such as Haskell [26].
- **State Transition Diagrams (STDs)** represent extended finite automata and are used to describe the behavior of a component in an SSD. The automata consist of a set of states (one of which is the initial state, marked with a black dot) and a set of transitions between the states, where each transition t is annotated with
  - $\mathsf{pre}(t)$, a boolean precondition (guard) on the inputs and local variables
  - input patterns $\mathsf{inp}(t) = \mathsf{inp}_1?\mathsf{pat}_1; \mathsf{inp}_2?\mathsf{pat}_2; \ldots$, specifying that values are to be read at the ports inp_i that should match the patterns pat_i (terms in the functional language that specify values of data types and can include variables). During the execution of t, variables in the patterns are bound to the matching values. For example, the pattern $\mathsf{inp}_1?\mathsf{openCar}(x)$ matches if the value $\mathsf{openCar}(\mathsf{admin})$ is received on port inp_1 and binds x in the preconditions, output expressions and postconditions to admin.
  - output expressions $\mathsf{outp}(t)$ of the form $\mathsf{out}_1!\mathsf{term}_1; \mathsf{out}_2!\mathsf{term}_2; \ldots$
  - postconditions $\mathsf{post}(t)$ of the form

    $$\mathsf{lvar}_1 = \mathsf{term}_1; \mathsf{lvar}_2 = \mathsf{term}_2; \ldots$$

  In the concrete syntax of the STDs, the annotation is written as $\mathsf{pre}(t) : \mathsf{inp}(t) : \mathsf{outp}(t) : \mathsf{post}(t)$. Leaving out components is interpreted as **true** for preconditions, and as an empty sequence in the other cases. A transition is executable if the input patterns match the values at the input ports and the precondition is true. At each clock tick, one executable transition in each component fires, outputs the values specified by the output patterns and sets the local variables according to the postcondition. The values at the output ports can be read by the connected components in the next clock cycle.
- **Extended Event Traces (EETs)** finally make it possible to describe exemplary system runs, similar to MSCs [14].

The Quest extensions [25] to AutoFocus offer various connections of AutoFocus to programming languages and formal verification tools, such as Java code generation, model checking using SMV, and bounded model checking and test case generation.

We apply the tool AutoFocus to model and analyse service-based systems by representing a service as a special kind of component in a system structure diagram, distinguished in the concrete notation by the use of the arrow symbol (cf. Fig. 1). Further kinds of special components are other interacting systems (box symbol) and roles for human actors (stickman). A service can carry annotations, such as relations to other services (e.g. dependencies) or quality of service attributes. The behaviour of a service is specified by a (network of) state transition diagrams. For an analysis, from a number of interacting services an execution scenario is built up in form of a system structure diagram consisting of an appropriately connected network of actors (roles, systems and services), which must fulfill the constraints implied by the annotations of the services.

Note that although we had to select a specific tool for the case study, the general concepts we present in this paper do not depend on the use of AutoFocus. Its main prerequisites are an executable, component-based description technique and verification support.

4. AUTOMOTIVE CASE STUDY

4.1 The Example

In this section we present an example of the authentication of a driver with the car key for personal settings and permissions as well as starting the car. This two-step authentication is useful for several settings like the skinning of the Man Machine Interface (MMI), accelerating the startup procedure of the car electronics or, much more important, security relevant settings. The first authentication step of the driver simply happens on opening the car with the car-key (either wireless or in the door lock) with an electronic code, which represents the profile of the user. The second authentication step happens when inserting the key into the ignition lock to release the immobilizer system and starting the car.

As an example for our scenario we take a car rental agency. A customer rents a car, but, however, for reasons of economy and insurance protection, she purchases a license only to drive in the inland and not to a foreign country. The car is equipped with a Global Positioning System (GPS) which locates the position of the car continuously and if the driver would pass a frontier to another country, the car would slow down and finally stop (of course, not before warning the driver insistently and instruct him to turn back). So, the customer's identity will be coded in the key, and in the car's configuration database the information about the different drivers is stored.

In the remaining, we refer to the clerk as *admin* and to the customer as *client*, respectively.

4.2 Modeling in AutoFocus

Now we start building a model of our system with the tool AutoFocus. In [8] we proposed a methodology for developing service based systems. According to this we start with the service identification phase, where we've discovered the actors which are relevant for our model: Driver, AutomotiveSystem, CustomizationService, ConfigDBService, MenuService, GPSService, DisplayService and StartService. In the use case modeling phase, we built up a sequence dia-

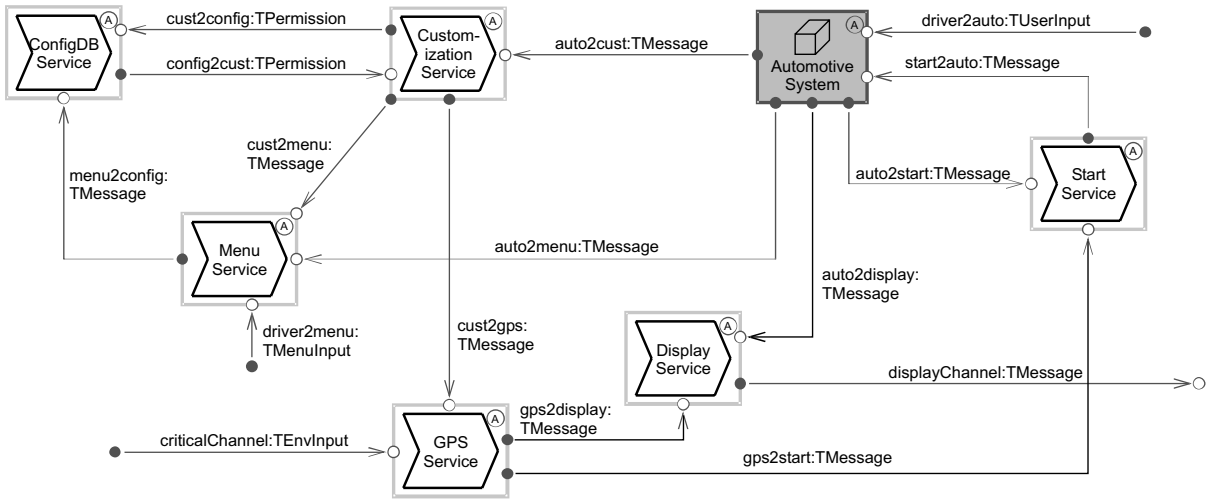


Figure 4: SSD of Authentication System

gram and so we can now start to enter the service modeling phase to build the model in the tool AutoFocus (Fig. 4).

The AutomotiveSystem represents the physical car. The driver can give input to the AutomotiveSystem in the form of instructions like `openCar`, `startCar`, `shutdownCar` or `closeCar`. Here, the identity of the driver will be passed to the AutomotiveSystem as an argument, e.g. `openCar(admin)`. The identity of the driver will be sent to the CustomizationService, which asks the database for the permissions of the driver. Now the permission is distributed to the GPSService and the MenuService where the user mode is set. The user mode can be set either to `adminMode` or `clientMode`, or `undef` by default.

After the `openCar(user)` message, the StartService will receive the message `startCar(user)` from the AutomotiveSystem and checks whether the *user* is allowed to drive the car (in our case, only client or admin) and sends an acknowledgment to AutomotiveSystem. If StartService has received a `GPSWarn` signal from the GPSService, it will stop the motor (if it is running) or will prevent it from starting (in the other case). Of course, stopping the motor would not happen immediately for safety reasons, but for simplicity, this is modeled in that way. In that situation, even the admin would be disallowed to start the motor again. This should only be possible by a special intervention to the car electronics at this point and will not be modeled here.

The GPSService may receive a warn signal from the environment, which indicates that the car is driving in a foreign country. If the boolean variable `FCpermission` is set to false, GPSService sends a message `GPSWarn` to the DisplayService and the StartService. DisplayService will show a message on the console display to inform the driver. The StartService will cause the car to stop (see below).

The Database ConfigDBService just contains two boolean variables: `adminFC` and `clientFC`, which indicate whether the user is allowed to drive to a foreign country. By default, `adminFC` is set to `true` and `clientFC` is set to `false`. On the message `getPermission(user)`, ConfigDBService answers with `returnPermission(user, value)`. value is the corresponding `FCpermission` of the user.

The MenuService can also be in three different modes: the `adminMode`, the `clientMode`, and `undef` by default. Only if the MenuService is in the `adminMode`, the user can change the permissions in the configuration database. The input is given from the environment and has the form `setPermission(user, value)`, where `value` can either be `true` or `false`. The MenuService is able only to configure the ConfigDBService, if it has received the `IgnitionOn` message from the AutomotiveSystem, that is, the user has received the start acknowledgment after inserting the key and starting the motor.

4.3 Threat Analysis

In the threat analysis we now check what can go wrong in the modeled system. Since every service provides a single function, we have to check whether the services are secure as a whole.

For a comprehensive threat analysis we would have to investigate all possible threats to the system. For this purpose, one can e.g. build up a threat tree for all possible security attacks against the system (cf. [1]). For this paper, we restrict the analysis to the authentication process of the driver against the AutomotiveSystem and have a look at the relevant services for the authentication process. The setting of the permissions in ConfigDBService is identified as a security critical operation. The following services are involved in the authentication process: MenuService, which is customized by CustomizationService with the person which opens the car and StartService, which is customized by AutomotiveSystem with the person who starts the car. Ideally, these two persons are identical.

Hence, the question is, whether these services can get in different modes anyhow and so a critical operation can be performed? In this case, the critical operation is to change the permissions in the ConfigDBService. If this could be done by the driver, she could apply rights to herself to drive to and inside a foreign country without bearing the consequences.

4.4 Model Checking

In this subsection we describe how to verify a service model with regard to security properties. For this purpose

AutoFocus generates an input file for the symbolic model checker SMV which carries out the actual model checking process, using symbolic model checking based on Binary Decision Diagrams (BDDs) [18].

We specify the required properties of the system using the temporal logic CTL (Computation Tree Logic, see [10]). In addition, to facilitate the formulation of properties, AutoFocus supports the definition of specification patterns (such as those presented by Dwyer et al. in [9]) appropriate for the specification domain that are automatically translated to standard CTL formulas. We make use of the property implies, which is defined as follows:

$$\mathsf{implies}(s, p) = \mathsf{AG}(s \Rightarrow p) \tag{6}$$

We said that a correct authentication of a user implies that all related services of the authentication process are in the same mode, that is, the variable `userMode` is set to a unique value, before a critical operation can be performed.

The MenuService only can change the settings in the ConfigDBService, if it is in the `adminMode`. In the following test we show that there exists a possible attack to the system, where the StartService and the MenuService are in different modes and the settings can be changed by a driver who is not the admin.

According to Sect. 2.3 we define the authentication predicate for the user `admin`. We identified the local variables of StartService and MenuService (both named `userMode`) as relevant for the authentication process. We therefore get:
$t_{auth} = (\mathit{MenuService.userMode}, \mathit{StartService.userMode})$.
The set of the correct valuations for the user `admin` is:
$M_{admin} = \{(\mathit{adminMode}, \mathit{adminMode})\}$.
The authentication predicate is

$$\begin{aligned}&\mathsf{authenticated}(\mathit{admin}) \Rightarrow \\ &\phi(\mathit{MenuService.userMode}, \mathit{StartService.userMode}) \\ &\in M_{admin}\end{aligned} \tag{7}$$

and the authorization predicate:

$$\begin{aligned}&\mathsf{use}(\mathit{MenuService.setPermission}(\mathit{user}, \mathit{value})) \\ &\Rightarrow \mathsf{authenticated}(\mathit{admin})\end{aligned} \tag{8}$$

Both predicates can be combined to a single implication, which results in the following AutoFocus/SMV formula:

$$\begin{aligned}&\mathsf{implies}(\mathsf{is\_Msg}(\texttt{MenuService.menu2config}), \\ &\quad(\texttt{MenuService.userMode} == \texttt{adminMode} \;\&\& \\ &\quad\texttt{StartService.userMode} == \texttt{adminMode}))\end{aligned} \tag{9}$$

That means, every time a message is sent on the channel `menu2config`, both the StartService and the MenuService have to be in the `adminMode` at the same time. Because the `setPermission(...)` message is the only one which is sent via the `menu2config` channel, it is sufficient to check whether there is a message sent via the channel at all (`is_Msg(...)`).

At the modeled abstraction level, model checking is not very time consuming and the computation time only takes a few seconds on modern machines. Just after having started the SMV run, a moment later we get the result that indeed the `setPermission`-message can be sent if the services are in different modes.

4.5 The Flaw

After exporting the AutoFocus-model to the SMV format and performing model checking, we get the result that a driver with a non-admin status could compromise the car system under certain circumstances.

If a check fails, SMV generates a counter example. AutoFocus uses the counter example to produce an EET, which helps us to understand how the flaw could occur. In Fig. 5 we see in detail what leads to the flaw.

In the following we explain the necessary events for the client to get writing access to the database:

1. The admin opens the car for the client with his key (which provides him admin permissions).
2. The client starts the car. Now the StartService is in the `clientMode` and the MenuService is in the `adminMode`.
3. The client is now able to change the permissions in the configuration database.
4. If the client closes the car, opens and starts it again, she will get the permissions to drive the car to a foreign country permanently.

This is the way the client is able to apply the permission to drive in a foreign country to herself.

The problem arises because two different persons take part in the whole authentication process: The admin opens the car for the client. So the system gets configured to the admin's profile. This, of course, is a good idea in principle, because the startup-procedure of the car computer can be started even before the driver gets into the car. But if the client starts the car after it was opened by the admin, the system, in particular the MenuService, is still configured to the admin's profile. Hence, the client owns the admin's permissions and therefore she is able to change the permissions in the configuration database.

4.6 Fixing the Flaw

The problem in modeling service based systems is to figure out all necessary dependencies of the security relevant services. For modeling a correct authentication process we have to make sure that all relevant services are in the same state before a security critical operation can be performed. Hence, the authentication process has to be considered as a whole and therefore we need to insert a dependency of CustomizationService and StartService. In this case we introduced a connection between the StartService and the CustomizationService instead of the StartService the MenuService. This is done in that way, because the CustomizationService propagates the user information to both the MenuService and the GPSService and helps to reduce the channels between the services.

According to Sect. 2.3, we have to extend the STDs as well. In StartService we added a new state "await_ack" (see Fig. 6(b)). In this state, StartService waits for an acknowledgement from the CustomizationService that the new user information has been set (resp. has been propagated successfully to the other services by the CustomizationService in this case).

If this is done correctly, the problem of independent mode settings in the StartService, the MenuService and the GPSService can be avoided and the system is secure.

We can now verify our changes in the model by making use of the local authentication predicate (cf. (4)) and the synchronization predicate (cf. (5)). We identify the critical states as init and await_ack. In none of these states

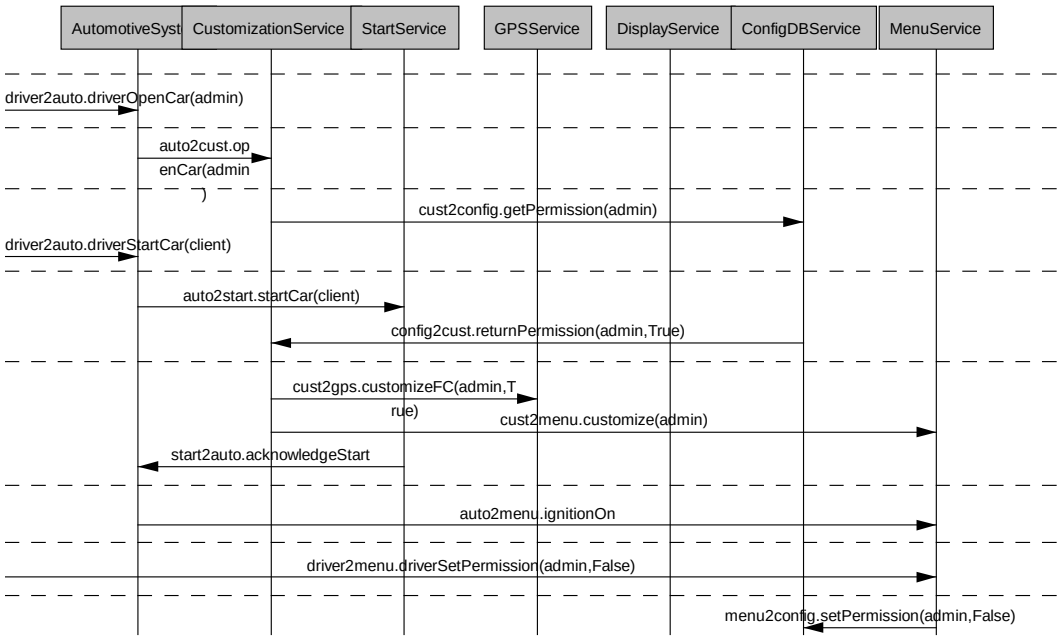


Figure 5: EET of the flaw

the local authentication process has been finished, so the state car_started remains for the use in the SMV formula. Converted to an SMV notation, for the local authentication predicate we get

$$\begin{aligned}&\mathsf{implies}(\mathsf{is\_Msg}(\texttt{MenuService.menu2config}),\\&\quad(\texttt{MenuService.userMode} == \texttt{adminMode}\ \&\&\\&\quad\texttt{StartService.@} == \texttt{car\_started}))\end{aligned} \tag{10}$$

whereby StartService.@ denotes the current state of StartService. The model checker returns true for this formula.

According to this, the synchronization predicate now gets to

$$\begin{aligned}&\mathsf{implies}(\texttt{StartService.@} == \texttt{car\_started}\ \&\&\\&\quad\texttt{MenuService.userMode} == \texttt{adminMode},\\&\quad\texttt{MenuService.userMode} == \texttt{StartService.userMode})\end{aligned} \tag{11}$$

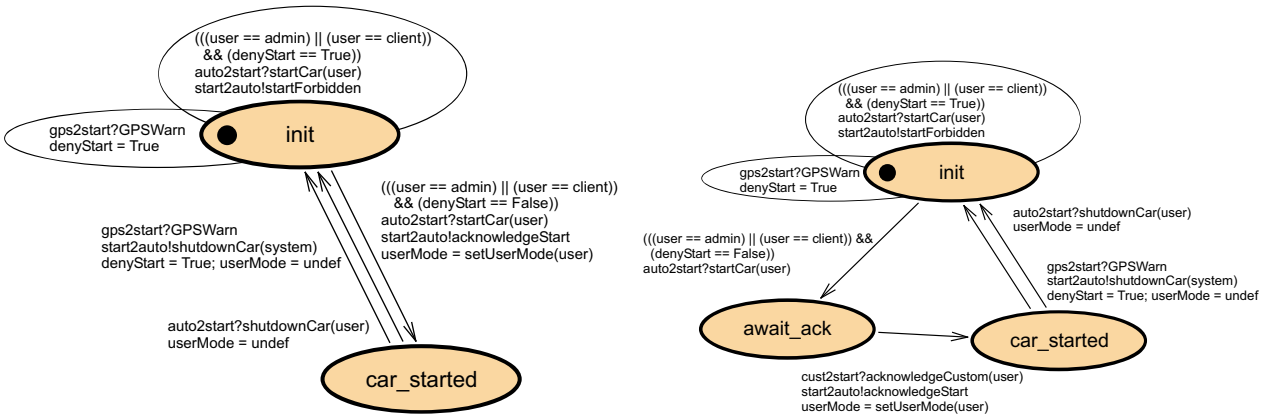
Again, the model checker reports that no flaw has been found in the model anymore. Therefore our changes have been applied successfully regarding the authentication process to our model. Of course, the original equation (9) also passed the test successfully.

5. RELATED WORK

There has been successful work on integrating security into service-oriented architectures, mostly regarding web-services. For example, [17] provides an Event-driven Framework for Service Oriented Computing (EFSOC) that is divided into four tiers relating to the events, business processes, resources tier, and access control. The aim is to provide an infrastructure that not only integrates business processes, services and associated support resources, but also access control mechanisms. [24] presents the trust negotiation framework Trust-Serv which supports policy lifecycle management for Web services, in a model-based approach. Furthermore, [2] describe an architectural solution to ensure between services. [7] develops security annotations for web services based on well-known security concepts. A reasoning engine is used whether agents and web service offer consistent levels of security. With respect to Grid service security, [27] presents the Globus Toolkit whose security implementation uses Web services security mechanisms for example to exchange credentials exchange. It makes use of a least-privilege model and aims to avoid the need for any privileged network service. Compared to the above-mentioned research cited, our work does not aim to provide a concrete architecture, but aims to provide a methodological approach to software-engineering targeted to service-oriented architectures, in a general context of service-based systems beyond the concrete situation of web-services. Also, our approach is based on a formal foundation and thus offers the possibility of automated proof of security properties.

Model-based engineering of service-oriented architectures has also been investigated. [21], proposes an approach to build business processes by composing web services in a model driven fashion. An approach to the development of service compositions is used in that context which includes abstract definition, scheduling, construction and execution. [3] gives a usage of UML models for modeling the architectur of a platform and the application level. The UML models can be checked for consistency between the two levels using a formal basis. Both pieces of research are not concerned with security requirements.

As an example for the treatment of security in the context of general systems engineering, not tailored to service-oriented systems, [15] presents work towards using the UML notation in security engineering. Other approaches have been proposed. For example, [11] presents a method extending on the Tropos approach with a notion of services offered (such as handling data, performing a task or fulfilling a goal) and their ownership and applies this to security-critical systems. AutoFocus has been used for security e.g. in [28, 29].



(a) old StartService (b) fixed StartService

Figure 6: Comparison of both STDs of the StartService

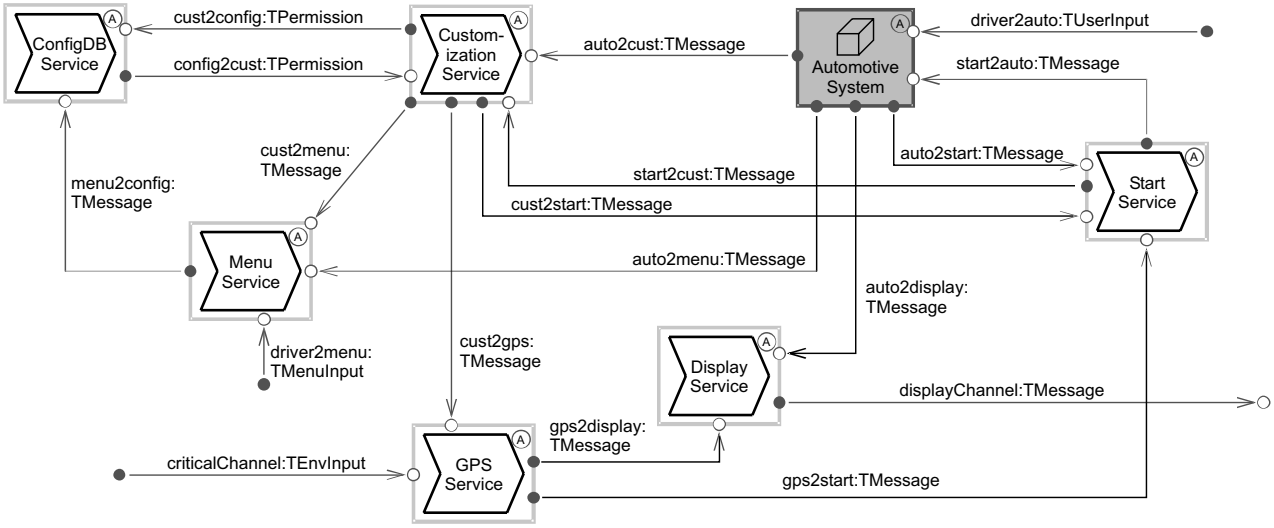


Figure 7: SSD of fixed SecureServices

6. CONCLUSION

This paper presented work regarding the formal analysis of security-critical service-based software systems using the computer-aided systems engineering tool AutoFocus. An example of an automotive application arisen from a project with a major car manufacturer was modeled and analyzed for security weaknesses using model checking.

Since the development of service-based software for security-critical systems is difficult, because services may easily interact with each other in unforeseen ways, we believe that a method for facilitating the development of security-critical service-based software systems is highly needed. Although for space limitations we could only demonstrate our method at a simple example in the context of this paper, our experiences from the project MEwaDis on service-based systems engineering in the automotive domain seem promising. Having a methodology for the correct development of secure service-based systems avoids mistakes introduced by developers that are no security experts. The formal basis of the method and the associated industrial-strength CASE tool support even allows efficient automated formal verification of the required security properties, which is crucial since many security properties cannot be established with conventional testing methods, since they are relative to a highly non-deterministic adversary.

In further work, to substantiate more fully that our approach is indeed fit in a general service-oriented setting, we aim to apply it to more case-studies from different domains than the automotive domain considered in this paper, including secure web-service architectures such as [17]. Furthermore, we will expand our security investigations to other security requirements which are important in service-based systems, e.g. confidentiality, integrity and non-repudiation.

To make our results applicable in industrial practice, we are working on a plug-in to automatically build OSGi services directly from the AutoFocus model. On the theoretical side, we will also further investigate conditions on the secure composition of secure services.

7. REFERENCES

[1] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

[2] F. Arcieri, F. Fioravanti, E. Nardelli, and M. Talamo. A Layered IT Infrastructure for Secure Interoperability in Personal Data Registry Digital Government Services. In *14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)*, pages 95–102, March 2004.

[3] L. Baresi, R. Heckel, S. Thüne, and D. Varró. Modeling and validation of service-oriented architectures: application vs. style. In *ESEC / SIGSOFT FSE*, pages 68–77. ACM, 2003.

[4] M. Broy. Modeling Services and Layered Architectures. In H. König, M. Heiner, and A. Wolisz, editors, *Formal Techniques for Networked and Distributed Systems*, volume 2767 of *Lecture Notes in Computer Science*, pages 48–61. springer, 2003.

[5] M. Broy, F. Huber, and B. Schätz. AutoFocus – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. *Informatik Forschung und Entwicklung*, 14(3):121–134, 1999.

[6] M. Broy and K. Stølen. *Specification and Development of Interactive Systems:* FOCUS *on Streams, Interfaces and Refinement*. Springer, 2001.

[7] G. Denker, L. Kagal, T. W. Finin, M. Paolucci, and K. P. Sycara. Security for DAML web services: Annotation and matchmaking. In *International Semantic Web Conference*, pages 335–350, 2003.

[8] M. Deubler, J. Grünbauer, G. Popp, G. Wimmel, and C. Salzmann. Tool Supported Development of Service Based Systems. In *11th Asia-Pacific Software Engineering Conference (APSEC 2004)*, Busan, Korea, Nov.. 30–Dec. 3 2004. IEEE Computer Society.

[9] M. Dwyer, G. Avrunin, and J. Corbett. Patterns in Property Specifications for Finite-State Verification. In *Proc. International Conference on Software Engineering (ICSE) 1999*, 1999.

[10] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science Publishers, 1990.

[11] P. Giorgini, F. Massacci, and J. Mylopoulos. Requirement engineering meets security: A case study on modelling secure electronic transactions by VISA and Mastercard. In I.-Y. Song, S. Liddle, T. Ling, and P. Scheuermann, editors, *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *LNCS*, pages 263–276. Springer, 2003.

[12] J. Grünbauer, H. Hollmann, J. Jürjens, and G. Wimmel. Modelling and verification of layered security protocols: A bank application. In *Computer Safety, Reliability, and Security (SAFECOMP 2003)*, volume 2788 of *LNCS*, pages 116–129. Springer, 2003.

[13] F. Huber, S. Molterer, A. Rausch, B. Schätz, M. Sihling, and O. Slotosch. Tool supported Specification and Simulation of Distributed Systems. In *International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 155–164, 1998.

[14] ITU. ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). ITU-TS, Geneva, 1996.

[15] J. Jürjens. *Secure Systems Development with UML*. Springer, 2004.

[16] J. Jürjens and G. Wimmel. Security modelling for electronic commerce: The Common Electronic Purse Specifications. In *I3E 2001*, pages 489–506. International Federation for Information Processing (IFIP), Kluwer Academic Publishers, 2001.

[17] K. Leune, van den Heuvel, Willem-Jan, and M. Papazoglou. Exploring a Multi-Faceted Framework for SoC: How to Develop Secure Web-Service Interactions? In *14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)*, pages 56–61. IEEE Computer Society, March 2004.

[18] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.

[19] MEWADIS Projekt Homepage. MEWADIS website at `http://www4.in.tum.de/~mewadis`. In German.

[20] Open Services Gateway Inititative. OSGi™ Service Platform Specification. Release 3, March 2003, `http://www.osgi.org`.

[21] B. Orriëns, J. Yang, and M. P. Papazoglou. Model driven service composition. In M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, editors, *First International Conference on Service-Oriented Computing (ICSOC 2003)*, volume 2910 of *LNCS*, pages 75–90, Trento, Italy, December 15–18 2003. Springer.

[22] OSGi Alliance. OSGI World Congress 2003. `http://2003.osgiworldcongress.com`.

[23] J. Philipps and O. Slotosch. The Quest for Correct Systems: Model Checking of Diagrams and Datatypes. In *Asia Pacific Software Engineering Conference 1999*, 1999.

[24] H. Skogsrud, B. Benatallah, and F. Casati. Model-driven trust negotiation for web services. *IEEE Internet Computing*, 7(6):45–52, 2003.

[25] O. Slotosch. Quest: Overview over the Project. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *Applied Formal Methods - FM-Trends 98*, pages 346–350. Springer LNCS 1641, 1998.

[26] S. Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley Longman, 1999.

[27] V. Welch, F. Siebenlist, I. T. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for grid services. In *HPDC 2003*, pages 48–57, 2003.

[28] G. Wimmel and J. Jürjens. Specification-Based Test Generation for Security-Critical Systems Using Mutations. In *4th International Conference on Formal Engineering Methods (ICFEM2002)*, 2002.

[29] G. Wimmel and A. Wißpeintner. Extended description techniques for security engineering. In M. Dupuy and P. Paradinas, editors, *Trusted Information: The New Decade Challenge*, page 469ff. International Federation for Information Processing (IFIP), Kluwer Academic Publishers, 2001. Proceedings of SEC 2001 – 16th International Conference on Information Security.