A Framework for Analyzing Composition of Security Aspects

Jorge Fox[1], Jan Jürjens[2]

[1] Institut für Informatik, Technische Universität München
Boltzmannstr. 3, D-85748 Garching b. München, Germany
fox@in.tum.de
[2] The Open University, Walton Hall
Milton Keynes. MK7 6AA
j.jurjens@open.ac.uk

Abstract. The methodology of aspect-oriented software engineering has been proposed to factor out concerns that are orthogonal to the core functionality of a system. In particular, this is a useful approach to handling the difficulties of integrating non-functional requirements such as security into complex software systems. Doing so correctly and securely, however, still remains a non-trivial task. For example, one has to make sure that the "weaving" process actually enforces the aspects needed. This is highly non-obvious especially in the case of security, since different security aspects may actually contradict each other, in which case they cannot be woven in a sequential way without destroying each other. To address these problems, this paper introduces a framework for the aspect-oriented development of secure software using composition filters at the model level. Using an underlying foundation based on stream-processing functions, we explore under which conditions security properties are preserved when composed as filters. Thanks to this foundation we may also rely on model level verification tools and on code and model weaving to remedy security failures. Our approach is explained using as case-studies a web banking application developed by a major German bank and a webstore design.

Keywords. Aspects in software engineering, aspect interference, verification, semantics, formal methods

1 Introduction

Aspect-Oriented Software Development (AOSD) is a novel software development paradigm still under evolution. It aims at overcoming limitations of previous paradigms such as that many requirements do not decompose neatly into behavior centered in a single decomposition element. For instance, OOP has difficulty localizing global concerns because the unit of modularity or first class instance in OOP is the class, which leads to what is called the 'tyranny of the dominant decomposition'. This dominant decomposition encapsulates data concerns effectively into classes, though systemic concerns, such as security, cannot be neatly encapsulated within the dominant modules. The result is that these kind of concerns are scattered across several classes.

Aspect-orientation aims at identifying and specifying *crosscutting concerns* i. e. aspects in separate modules, providing one more dimension to modularize software. For our purposes, it is sufficient to understand an *aspect* as a desired feature stemming from the requirements specification which is orthogonal to the main functional requirements and that contains functionality which affects several other modularization entities. The idea is thus that computer systems are better programmed by separately specifying the various concerns (understood as properties or areas of interest) of a system, describing their relationships, and then relying on mechanisms in the underlying AOSD environment to *weave* i. e. compose them into a coherent program.

This brings a problem we intend to deal with in this work, namely, the possible interaction of different aspects as a result of the weaving process. Since there exist aspects that actually contradict each other (for example the security aspects of *anonymity* and *accountability*), it is in general not clear whether two different aspects can be woven into a program without negative interference. Furthermore, establishing this for a given set of aspects and a given program is a non-trivial problem which requires having a formal foundation to analyze the system and the aspect interaction. Specifically, the problem of aspect composition involves analyzing not only their execution ordering, but is rather a problem of semantic analysis of the aspects that are woven on a given base element (such as an object's method, attribute, or the interface of a component). To explore the problem of aspect interaction we draw inspiration from one of the main research lines in AOSD, the Composition Filters (CF) approach along the lines of [1]. The reason is that this approach allows for a particularly insertion of aspects over the communication channels between objects. We recall that every system i.e. subsystem (in the sense of logical system entities) interaction can be reduced to the receiving and sending of messages. Making the communication channels a first-class entity in the aspect-interaction analysis helps reasoning over a number of problems in AOSD. We chose to explore aspect interaction in this work at the hand of two security aspects from industry interesting enough for a formal mathematical analysis. Security issues are usually accepted as crosscutting. In addition, the interaction of security protocols is by no means trivial and aspect interaction happens actually at specific points that is why we show our theory on a few well-defined points of execution. Correspondingly, consider the definition of *clear-* and *black-box aspect-oriented programming* (AOP) from [2] in terms of quantifications over the internal structure of components (clear-box) or over the public interface of components (black-box). In both cases, we have communication channels, either external (at the interface level) or internal (in the decomposition of the subsystem). In this report we address the specific situation where one can weave aspects in the interfaces among components, also known as *black-box aspect-orientation*. We believe this situation to be of sufficient interest. As mentioned above, it is related to an important line of research in AOP that adds aspect behavior through input and output filters superimposed on sets of objects [1].

This work has three main contributions. Firstly, we provide a formal foundation for aspects which does not seem to have been provided so far for composition filters. Secondly, we demonstrate how to use this formal foundation to formally analyze aspect composition including the composition of aspects with other aspects, and possible unwanted interactions. Finally, we apply our approach more specifically to the case of security aspects, because of the particular challenges involved with developing security-critical software. A formal foundation for dynamic security aspects in the context of aspect-oriented development seems to be missing until now. Security is generally accepted as a crosscutting concern, that is a reason to explore our framework at the hand of these examples. Our approach is independent of object-orientation and therefore allows us to use ideas from AOP and aspect composition outside object-oriented programming (OOP), but can be extended with concepts to also consider object-orientation. We use the term *component* in a conceptual and logical sense, not as a programming or physical entity. Here we propose a framework for the analysis of aspect interaction in software architecture. Our proposal is not intended for the analysis of aspect orientation at the code level.

In order to reason about aspect composition a framework with precise syntax and semantics is required. Such a framework is introduced in the next chapter. In Sect. 3 we introduce the composition model we used in this report, based on Composition Filters. In Sect. 4 we introduce security aspects through the example of a *secure channel*[3] aspect that will be used in the case study. In Sect. 5, we demonstrate how this aspect can be composed with the *authentication* aspect in a secure way. In Sect. 6, we consider two different security aspects and determine under which conditions they can be securely composed together. In Sect. 7 we introduce a framework for verification of predefined security properties in UML models. Based on the diagnosis of such a framework we may correct the failures by weaving the required aspect. We relate our work at the modeling level to aspect weaving at the code level with an example in Java and Compose* in Sect. 8. We close with a comparison to related work and conclusions.

2 Formal Foundation

We recall the definitions of streams, stream-processing functions, and composition of stream-processing functions from [3,4]. These definitions help specifying the components. Through the channel history of components we may analyze their interaction. For instance, we may observe whether a given message $m$received as input at channel s_1 in component $\mathcal{P}$ (Fig. 1) at time $\mathbf{t}$ is present in some output channel o at time $\mathbf{t+1}$. Similarly, given that $\mathcal{P}$ can be specified under this theory as a stream-processing function with precise syntax and semantics, we may follow its output history to verify whether a given system

[3] To be more precise this is actually a *secrecy-enforcing channel*. We call it "secure channel" in this work to facilitate reading.

property[4] is actually fulfilled. That is the reason for the use of this formalism as theoretical background.

The formal theory FOCUS [4] describes a system based on input/output relations on sets of histories of externally observable events. A system is divided into components. The behavior of a component is described by the relationship between its external input and output histories, defined as streams-processing functions (where "streams" are the sequences of input and output values of the system, to be defined below). This way we obtain a *black-box view* of the component in question. This theory also allows us to distinguish between *elementary* and *composite* specifications. Composite specifications are built from elementary specifications using constructs for composition and network description.

We model components through their communication histories, that means their *I/O behavior*, modeled as *streams*. A stream is a finite or an infinite sequence of values, often called *messages*. Streams are used to model the communication histories of *directed channels*, that is, channels that transmit messages in one direction. Given that the communication histories of channels are modeled by the streams of messages sent along the channels, we may specify a component, in this case a function, by characterizing the relationship between its input and its output streams. This is also a reason to rely on this theory for modeling composition filters, since they operate on messages sent among objects in a directed fashion. The model is introduced in Sect. 3.

Stream $_C =^{def} (\mathbf{CExp})^C$ (where $C \in \mathbf{Channels}$) for the set of C-indexed tuples of sequences of expressions. The elements of this set are called *streams*, specifically *input resp. output streams* if C denotes the set of input, resp. output, channels of a process P. Each stream $s \in \mathbf{Stream}_C$ consists of components s_c (for each $c \in C$) that denote the sequence of expressions appearing at channel c. The n^{th} element in this sequence is the expression appearing at time $t = n$.

Stream processing function A function $f : \mathbf{Stream}_I \rightarrow \mathbf{Stream}_O$ from sets of streams to sets of streams is called a *stream-processing function*. We consider deterministic systems in this definition.

Streams From a mathematical point of view, are functions mapping natural numbers to messages. The index on the channel (see $\mathbf{Stream}_C$ in the above definition) is needed since we have different channels and for each channel we have a stream. In the example below the index is i in channel s of component $\mathcal{P}$.

For instance, stream s_i (starting with message m_1 followed by messages m_2, and again m_2) is uniquely characterized by the function

$$s_i \in \{1, 2, 3\} \rightarrow \{m_1, m_2\}, \text{where } s_i(1) = m_1, s_i(2) = m_2, \text{and } s_i(3) = m_2.$$

As already mentioned, we build composite specifications out of elementary specifications by composing stream-processing functions. We therefore recall the

[4] i. e. set of behaviors see [5]

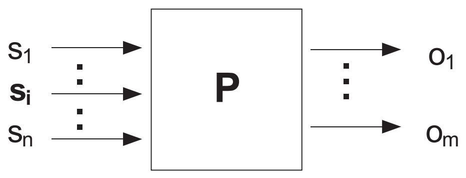


Fig. 1. Component $\mathcal{P}$ with input and output named channels

composition operator which we actually use as *weaving* function together with the tables we introduce in the next section.

Mutual-feedback composition operator $(\otimes)$
The composition of two stream-processing functions (illustrated in Fig. 2) $f_i : \mathbf{Stream}_{I_i} \rightarrow \mathbf{Stream}_{O_i}$ $(i = 1{,}2)$ with $O_1 \cap O_2 = \varnothing$ is:
$f_1 \otimes f_2 : \mathbf{Stream}_I \rightarrow \mathbf{Stream}_O$
with $I = (I_1 \cup I_2) \backslash (O_1 \cup O_2)$, $O = (O_1 \cup O_2) \backslash (I_1 \cup I_2)$,
where $f_1 \otimes f_2(\mathbf{s}) =^{def} \{\mathbf{t} \downharpoonright_O : \mathbf{t} \downharpoonright_I = \mathbf{s} \downharpoonright_I \wedge \mathbf{t} \downharpoonright_{O_i} \in f_1(\mathbf{s} \downharpoonright_{I_i})(i = 1, 2)\}$
(the time index $\mathbf{t}$ ranges over $\mathbf{Stream}_{I \cup O}$). For $\mathbf{t} \in \mathbf{Stream}_C$ and $C' \subseteq C$, the restriction $\mathbf{s} \downharpoonright_{C'} \in \mathbf{Stream}_{C'}$ is defined by $\downharpoonright_{C'}(c) = \mathbf{t}(c)$ for each $c \in C'$.

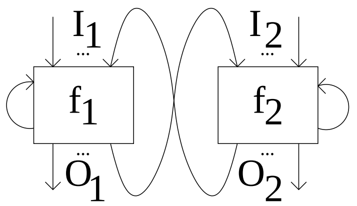


Fig. 2. Composition of two stream-processing functions with mutual feedback

The above definition formalizes the fact that the two stream-processing functions interact by exchanging messages over their shared input- and output-streams.

Example 1. If $f : \mathbf{Stream}_{I_1} \rightarrow \mathbf{Stream}_{O_1}$, $f(s) =^{def} \{1.s, 2.s\}$, is the stream-processing function with input channel I_1 and output channel O_1 that outputs the input stream s prefixed with either 1 or 2, and
$g : \mathbf{Stream}_{I_2} \rightarrow \mathbf{Stream}_{O_2}$, $g(s) =^{def} \{0.s\}$, the function with input (resp. output) channel I_2 (resp. O_2) that outputs the input stream prefixed with 0, then the composition $f \otimes g : \mathbf{Stream}_{I_1} \rightarrow \mathbf{Stream}_{O_2}$, $f \otimes g(s) = \{0.1.s, 0.2.s\}$,

outputs the input stream prefixed with either of the 2-element streams 0.1 or 0.2.

We shortly recall from [4] that one can associate a stream-processing function to a state machine. The definitions of state machine and state transition diagram are recalled from [6].

State transition relations are described by state transition rules. These are logically represented with the help of assertions that contain the state attributes v as identifiers in a primed form v' and in an unprimed form v as well. Unprimed identifiers relate to the values of the given attribute in the state before the state transition and the primed identifiers to the values of the attributes in the state after the transition.

Transition rules are of the syntactic form [$\mathcal{P}$] x:e / y:b, where $\mathcal{P}$ is a guard which is a state assertion referring only to the local attributes of the system. The identifier x denotes an input channel and y an output channel, e and b are messages of respective types of the channel. A transition is fired when an appropriate message is received and the specified guard is fulfilled.

State machines are described by a state assertion U that characterizes the set of initial states and a finite set of state transition rules R of the form presented above.

A state transition specification $\mathcal{S}$=[$\mathcal{I}/\mathcal{O}$, **attribute** B; **initial** $\mathcal{U}$;$\mathcal{R}$] consists of a given set of typed input channels $\mathcal{I}$, a given set of typed output channels $\mathcal{O}$ and a set of typed attributes B; furthermore, it contains a state assertion $\mathcal{U}$ which characterizes the set of initial states and a set $\mathcal{R}$ of state transition rules.

Furthermore, state machines can be described by a **state space**, **state transition rules**, and an **assertion on the initial states** of the machine. The state is specified by a set of typed attributes. Each valuation of these attributes describes a state. In the assertion related to the initial state we refer to the state attributes and the output channels. We model the state transitions between the system states and the input and output sequences of messages arriving through the channels. State machines are describe by state transition diagrams.

A state transition diagram is a graph with nodes labeled by control states and arcs labeled by state transitions. One control state or a set of them is marked as being initial.

In the state transition diagrams we present here, the channel type is indicated below the graph together with the corresponding expressions that each may send (respectively receive).

We denote a specification $\mathcal{S}$ as $[\![S]\!]$. So that, for instance, the specification represented as a state machine in Figure 5(a) will be referred as $[\![Send_E]\!]$. The input and output channels are specified in the lower part of the diagrams.

We refer indistinctly to components as stream-processing functions.

In the next section, we introduce the concern composition framework (i. e. model) which is based on the formalism exposed here. The formalism, despite being complicated, enables us to reason about aspect composition in logical terms. This means in terms of predicate logic. Although it is not further explored in this work, the framework may allow us to go a step further and rely on automated software verification tools like *Automated Theorem Provers (ATP)*, as well as on *High-Order Logic (HOL)* tools. We demostrate its use on an informally written theorem and its proof in Sect. 6, for more complex cases we suggest relying on ATPs.

In this sense our framework might be considered as a solid step toward (formal) verification of aspect interaction at a fine granular level, as illustrated in Sect. 6. In the coming section we introduce the composition model (Sect. 3), and explain aspect composition in(Sect. 4, and Sect. 5).

3 Composition Model

The composition model we propose is inspired by the composition filters model presented in [1]. It is a concern composition mechanism based on message interception over communication channels between components. Input and output filters are defined around a base object. The filters select messages based on given criteria, and either perform a process on the accepted messages or send the message (intact or altered) along to a predefined recipient. If the message was not accepted, it is then forwarded to the next filter. Input filters select incoming messages to the base object, output filters select outgoing messages of the base object. The model we refer to is implemented is implemented in the Compose* framework [7]. It is defined at the programming level, not at the software architecture level as is ours. Compose* is one of the first aspect-oriented languages and contrary to other aspect-oriented tools (such as Hyper/J) it is mature enough to be applied industrially. This is why we chose the former as inspiration for the composition model.

The model exploits the fact that objects in the OOP can only communicate by sending messages. Based on this principle, a set of filters is defined together with an advice. The filters select incoming and outgoing messages according to logical conditions stated by the developer and execute the respective advice. Please note that such filters are stated syntactically, namely over the signature of objects.

We introduce an architecture level composition-filters model. In the case of a *black-box* view, components as defined in Sect. 2 communicate only through their input and output channels, i. e. interfaces. In the case of a *glass-box* (also called *clear-box*) view, the communication is analyzed at the level of internal communication channels. The main elements we propose for a formal composition filters model are a set of base components, a set of component-filters which are each defined as a stream processing function, and a mapping relating both.

To explain the relationship between our concepts and the prevalent AOP terminology we briefly recall (from [2] and [8]) the concepts of: *Join Point* (JP), *Pointcut (Designator)* (PCD), and *Advice*.

The model relies on the concept of components as interactive system entities. We consider the base concern as a set of interconnected components. This can be understood as a network of components that constitute a software system. The aspects are then modeled as a set of filters, also defined as components. The filters are composed with the base concern by weaving them as components added in the communication channels of selected members of the base set. The selection mechanism works in this case as a JP selector or PCD. In *aspect-orientation* a `join point` is a well-defined place in the structure or execution flow of a program where additional behavior can be attached e. g. method call/execution or data element. While a `Pointcut (Designator)` describes a set of join points. *Weaving* refers to the composition of aspects with other concerns in the system [8]. In this work, our JPs are communication channels among components (namely the granularity level of our approach is at the messages sent over communication channels), and our PCD is expressed as a table indicating the weaving of components (as illustrated in the last part of tables Table 1 through 4). We define weaving as a component transformation function based on the composition operator $\otimes$.

An `Advice` is a (functional) element which augments or constrains other concerns at JP's matched by a pointcut expression i. e. pointcut delimiter. In our model the advice is specified by state machines expressing the behavior of a given filter.

Composition Filter Given a set of n stream-processing functions a Composition Filter (CF) is defined as a set: $CF =^{def} \{CF_1, \ldots, CF_n\}$, where CF_j is a component (i. e. stream processing function) with index $j \in \mathbb{N}^+$

The CF model in [1] allows one to direct messages to internal or external objects. We believe that, although powerful, this mechanism can be difficult to understand, to control, and to verify whether it is actually being used as intended. Anyways, our model also allows to direct messages to components other than the ones originally considered.

We restrict our analysis to a composition model defined over the external channels of components (i. e. at the level of component interface), since this is sufficient for our purposes and allows the kind of analysis described in Sect. 6. Please note that our model can be perform message redirection. Since the filters are defined as components, we may provide for channels i. e. streams from the filter to components in other parts of the system.

We consider now a means for specifying the insertion of the composition filters expressed in Table 1. For that purpose we adapt the aspect abstraction proposed in [9] (for an object-oriented model) to our component-oriented approach. This table helps visualizing aspect weaving. The first part of the table provides the aspect name and the second one its identifier. The third row of the table designates the set of components ($\mathbb{M}$) affected by the aspect as well as the base component around which the filters will be composed. This *base component* is intended to act as a pivotal point around which the aspects are connected. In the case we explore here, the base element is the communication channel in Fig. 3. The fourth row of the table relates each component in the filter set to the formal specification of the intended behavior i. e. *advice* (expressed as state machines). The fifth row contains our Pointcut (or PCD), in other words, the composition order.

In Fig. 3 the process of weaving i. e. composing the aspects with the channel and other parts of the system is illustrated. The tables are explained in more detail in § 4 and 6 at the hand of the particular case study.

The *weaving* process is considered as a transformation function of the form

$$\Psi : \ \{Components\} \times CF \ \mapsto \ \{Components\}$$

We explore it in terms of model transformation in [10]. Alternatively, we relate the actual realization of the weaving function to the composition operator ($\otimes$) from Sect. 2 together with the specification of the weaving order in the corresponding tables. We briefly recall that there are two possible weaving techniques considering when the weaving process is performed. On the one hand, dynamic weaving if the process is performed at run time, on the other hand, static weaving if the process is performed during compilation. Our proposal belongs to the second group. It performs static weaving without tool support. Advancing our composition model with other weaving mechanisms as well as tool support belongs to future work.

Table 1. Aspect weaving to base concern: Generic case

Aspect Name
Aspect identifier
Set of base components $\mathbb{M} = \{C_0, ..., C_K, ..., C_m\}$ Base Component: $Base = C_K$
Composition Filter: $[\![\mathsf{CF}_1]\!] = [\![Specification_1]\!]$, $[\![\mathsf{CF}_2]\!] = [\![Specification_2]\!]$ $\ldots$ $[\![\mathsf{CF}_n]\!] = [\![Specification_n]\!]$
Weaving order of CF to base concern (Pointcut Designator): $[\![C_0]\!] \otimes \cdots \otimes [\![\mathsf{CF}_1]\!] \otimes [\![Base]\!] \otimes [\![\mathsf{CF}_2]\!] \otimes \cdots \otimes [\![C_m]\!]$

To summarize, we introduced our composition model based on the concepts of stream, stream-processing function, and state machine, also inspired by the *aspect-oriented* language *Compose** which performs aspect weaving by message interception and superimposition of filters over a base object (described at the beginning of this section). The formal definitions in Sect. 2 provide precise syntax and semantics to our model. On this basis we specify a generic secure channel in the next section together for which we will then perform a formal analysis of (security) aspect composition.

The relation of our proposal to object-orientation can be drawn if we consider the methods of a class as a subsystem on its own i. e. a component. We recall that the composition filters approach as in [11] stresses the fact that the communication between methods in OOP is abstracted into the messages sent between objects, we model this message sending as communication channels between components.

4 Specifying Security Aspects

We now illustrate the proposed composition model on the case of a web banking system taken from a joint project with a major German bank. We consider the base concern to be a Client and a Web server communicating over a given Internet medium (also referred to as communication channel in the tables). The aspects are a secure channel as defined in [12] and an authentication protocol from [13].

The web banking system consists of an Internet-based application that allows clients to complete and sign a digital order form. The main security concerns of this application we consider for the analysis are two. First, user data must be kept confidential. This implies the use of a secure communication channel or a protocol that ensures the privacy of the data. We build in section 4.2 our system over a secure channel that may be considered generic and usable for similar problems. This secure channel has been modeled against a generic attacker and fulfills the property of ensuring user's privacy. Second, it is required that orders may not be submitted in the name of other users. On account of this when the user logs-in an authentication protocol runs and a confidential connection is established. The authentication protocol is based on SSL and it constitutes the second concern that we will compose with the secure channel. This demonstrates the practical application of our approach.

An overview of the system is shown in Fig. 3. We focus on the examination of the two security concerns and their interaction. Nevertheless, the approach is of interest for related problems. For instance, composing a different authentication protocol over the secure channel, or modifying the secure channel while preserving the outer layers of the system and analyzing whether the security concerns still hold. As shown in Fig. 3 we build the authentication process over the secure channel. Both aspects are first treated independently of one another, and are afterward woven together with the base components using the composition operator $\otimes$ defined in Sect. 2. From the software engineering point of view keeping changes as modular as possible is of relevance and even more if we need to preserve complex concerns as the ones related to security. What we aim at is allowing a given commercial institution to make use of a verified set of components. Constructing a secure channel that guarantees `secrecy` and a protocol that guarantees `authenticity` each as an aspect allows one to later reuse both

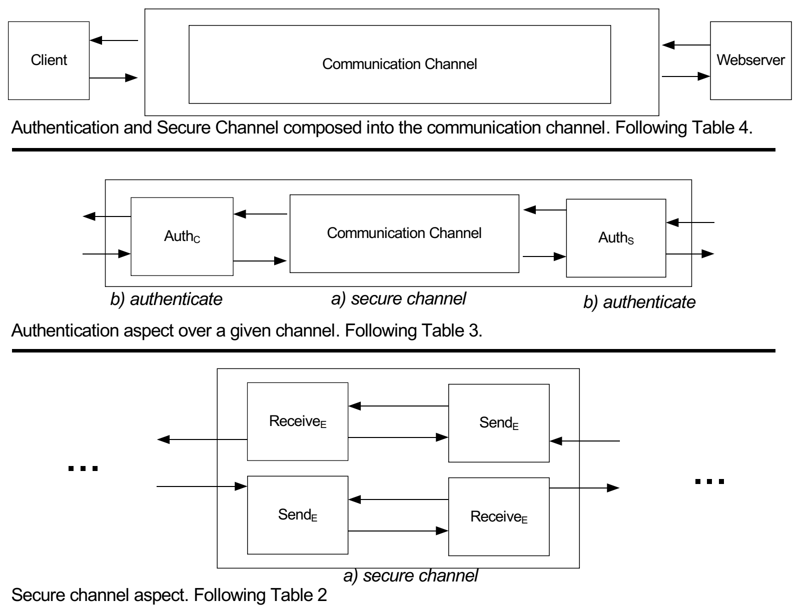

Client
Communication Channel
Webserver
Authentication and Secure Channel composed into the communication channel. Following Table 4.
AuthC
Communication Channel
AuthS
b) authenticate
a) secure channel
b) authenticate
Authentication aspect over a given channel. Following Table 3.
ReceiveE
SendE
SendE
ReceiveE
...
...
a) secure channel
Secure channel aspect. Following Table 2

Fig. 3. Architectural view of the process. Weaving as component composition

in other parts of the system. The former can be found in Sect. 4.2, the latter in Sect. 4.3.

4.1 Cryptographic Model

We introduce some definitions that are used for modeling cryptography and authentication in the coming specifications.

We assume a set **Keys** with a partial injective map $()^{-1}$: **Keys** $\rightarrow$ **Keys**. The elements in its domain, which may be public, can be used for encryption and for verifying signatures. Those in its range, usually assumed to be secret, are used for decryption and signing. We assume that every key is either an encryption or decryption key, or both: any key k satisfying $k^{-1} = k$ is called *symmetric*; otherwise *asymmetric*. We fix sets **Var** of *variables* and **Data** of *data values*. We assume that **Keys**, **Var**, and **Data** are mutually disjoint. **Data** may also include *nonces*, and other secrets. A *nonce* is a random value supposed to be used only once.

We recall that a term algebra generated by a set of elements and operations is the set of terms formed by applying the operations to the elements. A quotient of the term algebra under a given set of equations is derived from the term algebra by imposing these equations, and those that can be derived from them, on the terms. It follows that the algebra of *cryptographic expressions* **Exp** is the

• $\_::\_$	(concatenation)
• $\mathsf{head}(\_)$ and $\mathsf{tail}(\_)$	(head and tail of a concatenation)
• $\{\_\}_\_$	(encryption)
• $Dec_\_(\_)$	(decryption)
• $Sign_\_(\_)$	(signing)
• $Ext_\_(\_)$	(extracting from signature)
• $Hash_\_(\_)$	(hashing)

- $Dec_{K^{-1}}(\{E\}_K) = E \;\; \forall E \in \mathbf{Exp}$ *and* $K \in \mathbf{Keys}$
- $Ext_K(Sign_{K^{-1}}(E)) = E \;\; \forall E \in \mathbf{Exp}$ *and* $K \in \mathbf{Keys}$
- and the usual laws regarding concatenation, $\mathsf{head}()$, and $\mathsf{tail}()$:
  - $(E_1 :: E_2) :: E_3 = E_1 :: (E_2 :: E_3)(\forall E_1, E_2, E_3 \in \mathbf{Exp})$
  - $\mathsf{head}(E_1 :: E_2) = E_1$ $(\forall E_1, E_2 \in \mathbf{Exp})$ and
  - $\mathsf{tail}(E_1 :: E_2) = E_2$ $(\forall E_1, E_2 \in \mathbf{Exp}$ such that there exist no E, E' with $E_1 = E :: E'$. For all other cases, $\mathsf{head}()$ and $\mathsf{tail}()$ are undefined.

Fig. 4. Cryptographic expressions

quotient of the term algebra generated from the set **Var** ∪ **Keys** ∪ **Data** with the operations in Fig. 4 and by factoring out the equations given there.

Furthermore we define the type **Events**, for the communication channels, as type **Events** = request | return(e) | transmit(e) | receive | clientHello | Data(e,f) | Dataform(e) | e where e, f ∈ **Exp**.

4.2 Secure Channel

Based on the cryptographic model from the previous subsection we may now introduce the secure channel aspect as the composition filter specifications of a sender (Send_E) and a receiver (Receive_E). Both are weaved with the authentication aspect in Sect. 5. We then analyze the composition of the two aspects in a general setting in Sect. 6.

The secure channel aspect consists of two composition filters. On the one hand, the sender filter that will be coupled to the client and the communication channel. The sender is in charge of encrypting and signing the messages that will be further sent over the communication channel. On the other hand, we have the receiver filter that gets the messages from the channel, decrypts and unsigns the message, and sends it further.

The next two specifications represent the aspect we compose on the base concern. We delineate the weaving of the secure channel over the communication channel in Table 2.

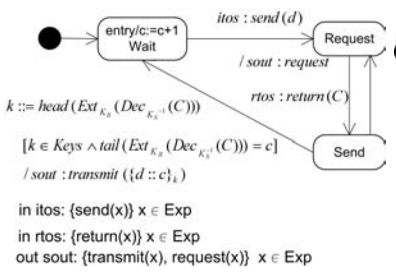
The specification in Fig. 5(a) represents the sender side of our first aspect according to the above security property. The state machine represents the specification. The sender retrieves the signed and encrypted symmetric session key k_j from the receiver, checks the signature, and encrypts the data under the symmetric key. Encryption is performed together with a sequence number c, to avoid replay.

Table 2. Aspect weaving to base concern: Secure channel

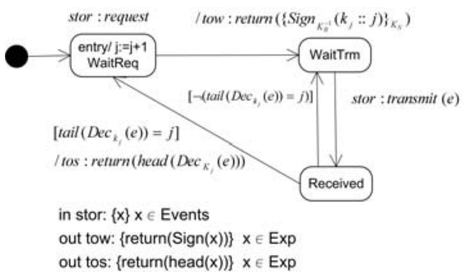
Aspect Name: Secure channel
Aspect identifier:CF_{SC}
Set of base components $\mathbb{M} = \{Communication\ channel\}$ Base Component: $Base = Communication\ channel$
Composition Filter: $[\![\mathsf{CF}_1]\!] = [\![Send_E]\!]$, $[\![\mathsf{CF}_2]\!] = [\![Receive_E]\!]$
weaving order of CF to base concern: $[\![\mathsf{CF}_1]\!] \otimes [\![Base]\!] \otimes [\![\mathsf{CF}_2]\!]$

The secrecy property considered here relies on the idea that a system specification preserves the secrecy of a piece of data d if the system never sends out any information from which d could be derived, even in interaction with an adversary.

The second part of the secure channel aspect, namely the receiver, is represented in Fig. 5(b). The receiver first gives out the key k_j with a signature and also with a sequence number j, and later decrypts the received data checking the sequence number.


entry/c:=c+1
Wait
itos : send(d)
Request
/ sout : request
rtos : return(C)
Send
k ::= head(Ext_{K_s}(Dec_{K_s^{-1}}(C)))
[k ∈ Keys ∧ tail(Ext_{K_s}(Dec_{K_s^{-1}}(C))) = c]
/ sout : transmit({d :: c}_k)
in itos: {send(x)} x ∈ Exp
in rtos: {return(x)} x ∈ Exp
out sout: {transmit(x), request(x)} x ∈ Exp

(a) Sender: $Send_E$


stor : request
/ tow : return({Sign_{K_s^{-1}}(k_j :: j)}_{K_s})
entry/ j:=j+1
WaitReq
WaitTrm
[¬(tail(Dec_{k_j}(e)) = j)]
stor : transmit(e)
[tail(Dec_{k_j}(e)) = j]
/ tos : return(head(Dec_{K_j}(e)))
Received
in stor: {x} x ∈ Events
out tow: {return(Sign(x))} x ∈ Exp
out tos: {return(head(x))} x ∈ Exp

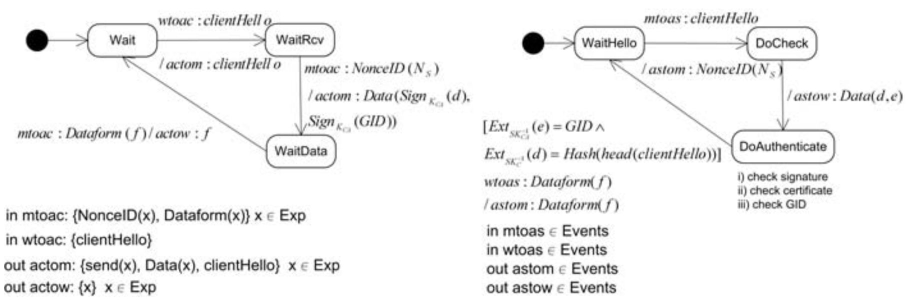
(b) Receiver: $Receive_E$

Fig. 5. State Transition Diagram Secure Channel

4.3 Authentication

In this section we specify the authentication aspect. Its composition with the secure channel from the previous on is examined in Sect. 6.

We explain a typical run of the authentication aspect, specified as state machines in Fig. 6(a) and Fig. 6(b). The client sends the authentication client a *clientHello* message which is forwarded to the server. Afterwards a randomly generated number (nonce) is sent by the web server. The client signs this nonce with his private key K_C and sends it back together with a global identification number (GID) which is signed using the Key provided by a certification authority K_{CA}. The server checks the signature, the certificate and the GID, and sends the client a Data Form. In the specification $Auth_S$ we do not establish a communication channel with the web server itself – for simplicity we assume the data form is generated by the web server and provided to the authentication server. Note that NonceID(**Exp**) is a member of the set **Data** as described in 4.1 and is represented in the specification $Auth_S$ as a hash function over the first element of the clientHello message.



(a) Client: $Auth_C$ (b) Server): $Auth_S$

Fig. 6. State Transition Diagram Authentication

We may define other aspects similarly. Once the base component(s) and aspects are specified we proceed to define how they are actually related as a system in the coming chapter. This is actually performed by relating the channel names which gives the weaving ordering and the resulting system. Therefore, channel renaming is a necessary step for weaving. Just as in AOP the underlying framework ultimately relates aspects and base code at the level of names in class methods or data, as is the case in the prevailing aspect languages.

5 Composing Security Aspects

The authentication aspect is composed over the secure channel aspect which is first established and generates the session keys. The orders will be signed based on these keys and therefore allow us to guarantee that bank orders can not be sent in the name of other users, which is the second aspect we need to consider.

Similarly to the secure channel aspect, the authentication aspect is implemented as composition filters composed with the client and the server. Table 3 establishes how this aspect is composed over the communication channel. As designated in Fig. 3 the weaving of the CFs to the base concern is:

$$[\![Client]\!] \otimes [\![Auth_C]\!] \otimes [\![CommunicationChannel]\!] \otimes [\![Auth_S]\!] \otimes [\![Webserver]\!]$$

$$\text{Where } [\![CommunicationChannel]\!] = [\![Send_E]\!] \otimes [\![Receive_E]\!]$$

In order to send and receive through the secure channel, $Send_E$ is connected on the communication channel from the `Client` to the `Server` with the client side $Auth_C$ of the authentication protocol, and when the message is sent from the `Server` to the `Client`, $Send_E$ connects to $Auth_S$. This is illustrated in Fig. 3. Ultimately, the channel names are related as indicated below[5]:

I. Communication Channel:
  i. $[\![Send_E]\!]$.sout $=$ $[\![Receive_E]\!]$.stor
  ii. $[\![Receive_E]\!]$.tos $=$ $[\![Send_E]\!]$.rtos
II. Authentication and Communication Channel
  i. $[\![Auth_C]\!]$.actom $=$ $[\![Base]\!]$.itos
  ii. $[\![Auth_C]\!]$.mtoac $=$ $[\![Base]\!]$.tow
  iii. $[\![Auth_S]\!]$.mtoas $=$ $[\![Base]\!]$.tow
  iv. $[\![Auth_S]\!]$.astom $=$ $[\![Base]\!]$.itos

Table 3. Aspect weaving to base concern: Authentication

Aspect Name: Authentication
Aspect identifier:$\mathsf{CF}_{Authentication}$
Set of base components $\mathbb{M} = \{Communication\ channel\}$ Base Component: $Base = Communication\ channel$
Composition Filter: $[\![\mathsf{CF}_1]\!] = [\![Auth_C]\!]$, $[\![\mathsf{CF}_2]\!] = [\![Auth_S]\!]$, $\mathsf{CF}_{Authentication} = \{[\![\mathsf{CF}_1]\!], [\![\mathsf{CF}_2]\!]\}$
Weaving order of CF to base concern: $[\![\mathsf{CF}_1]\!] \otimes [\![Base]\!] \otimes [\![\mathsf{CF}_2]\!]$

Altogether, in Sect. 4 we introduced the two aspect specifications that represent crosscutting concerns i. e. *aspects* in the sense of the definition presented in [9]. First, the secure channel's specification in Sect. 4.2. Second, the authentication aspect in Sect. 4.3. The next step is to weave them in a way that allows us to analyze whether the composition of both specifications respects and entails the desired security properties, namely *secrecy* and *authentication*. Weaving is performed according to the scheme in Sect. 3. It is a case of static weaving, yet

[5] Notation: $[\![SpecificationName]\!]$.Channel name

Table 4. Aspect weaving to base concern: Authentication and Secure Channel

Aspect Name: Authentication and Secure Channel
Aspect identifier:CF_{AuthSC}
Set of base components $\mathbb{M} = \{Client, Webserver, \mathsf{CF}_{SC}\}$ Base Component: $Base = \mathsf{CF}_{SC}$
Composition Filter: $[\![\mathsf{CF}_1]\!] = [\![Auth_C]\!]$, $[\![\mathsf{CF}_2]\!] = [\![Auth_S]\!]$
Weaving order of CF to base concern: $[\![\mathsf{Client}]\!] \otimes [\![\mathsf{CF}_1]\!] \otimes [\![Base]\!] \otimes [\![\mathsf{CF}_2]\!] \otimes [\![\mathsf{Webserver}]\!]$

the framework can be extended to consider dynamic weaving, since once the components are verified the phase at which they are composed to the system does not affect its behaviour (at least the way components and therefore aspects are defined in the formal theory recalled in Sect. 2).

6 Analyzing Composition of Security Aspects

Resulting from the previous chapters, in which we define our aspects and set their composition i. e. weaving, now, we may analyze the *secure channel* and *authentication* aspects. The idea is to determine under which conditions they can be securely composed together. The authentication aspect is now implemented over the secure channel as shown in Table 4. Please note that CF_{SC} in Table 4 was defined in Table 2.

The resulting system is specified:
$[\![Client]\!] \otimes [\![Auth_C]\!] \otimes [\![SecureChannel]\!] \otimes [\![Auth_S]\!] \otimes [\![Webserver]\!]$, where
$[\![SecureChannel]\!] = [\![Send_E]\!] \otimes [\![Receive_E]\!]$

We define how this aspect is woven over a given communication channel e. g. internet in Table 3.

Specifically, the method we applied here can be summarized in three steps. First, we establish that the secure channel defined in Sect. 4.2 is generic in the sense that it can securely be composed with a system that satisfies certain saneness condition[6]. Then, we establish the properties that the resulting system should preserve (see below). Finally, we prove it.

Theorem 1. *The secure channel aspect preserves the secrecy of the variable* d *from adversaries whose knowledge before initialization of the system does not include any values in the set* $\{\mathsf{K}_\mathsf{S}^{-1}, \mathsf{K}_\mathsf{R}^{-1}\} \cup \{\mathsf{k}_n, \{x :: n\}_{\mathsf{k}_n}\}$ *and includes only such values of the form* $Sign_{\mathsf{K}_\mathsf{R}^{-1}}(\mathsf{k}' :: m)$ *for which we have* $\mathsf{k}' = \mathsf{k}_m$ *for all* $m \in \mathbb{N}$ *and* $\mathsf{k}' \in \boldsymbol{Exp}$*.*

[6] for example, that the system itself does not send the secret values to the adversary outside the secure channel

Proof

The proof is of informal nature.
Note that the adversary knowledge set $\mathcal{K}_A$ is contained in the algebra generated by $\mathcal{K}_A^0 \cup \{\{\mathcal{S}ign_{\mathsf{K}_\mathsf{R}^{-1}}(\mathsf{k_i} :: \mathsf{j})\}_{\mathsf{K_S}}\}$ and the expressions $\{\mathsf{d} :: n\}_\mathsf{K}$ for inputs d, where $\mathcal{K}_A^0$ is the initial knowledge of the adversary: Firstly, the adversary can obtain no certificate $\{\{\mathcal{S}ign_{\mathsf{K}_\mathsf{R}^{-1}}(k :: \mathsf{j})\}_{\mathsf{K_S}}\}$ for $k \neq \mathsf{k_j}$, because the Receiver object only outputs the certificates $\{\mathcal{S}ign_{\mathsf{K}_\mathsf{R}^{-1}}(\mathsf{k_j} :: \mathsf{j})\}_{\mathsf{K_S}}$ (for $\mathsf{j} \in \mathbb{N}$) to the Internet. Secondly, the sender outputs only messages of the form $\{\mathsf{d} :: n\}_k$ to the Internet, for inputs d and any $k \in \mathbf{Keys}$ for which a certificate $\{\mathcal{S}ign_{\mathsf{K}_\mathsf{R}^{-1}}(k :: n)\}_{\mathsf{K_S}}$ has been received. Here k must be K_n since no other certificate can be produced (since the key $\mathsf{K}_\mathsf{R}^{-1}$ is never transmitted). Note also that $\mathcal{K}_{\mathcal{A}}^\mathsf{p} = \mathcal{K}_{\mathcal{A}}^0$ since there are no components accessed by the adversary.
Also, the values that an adversary may insert into the Internet link may only delay the behavior of the two objects regarding $\mathsf{outQu}_{\mathcal{C}'}$ since the adversary has no other certificate signed with $\mathsf{K}_\mathsf{R}^{-1}$ and does not have access to the key $\mathsf{K}_\mathsf{R}^{-1}$, and because of the transaction numbers used. Thus any other value inserted is ignored by the two objects.

This means in particular that the secure channel aspect can be securely composed with any aspect which obeyes the saneness conditions required in the above result. This gives us a general result on aspect composition, instantiated at the case of the secure channel aspect.

The protocol that implements the authentication aspect introduced in Sect. 4.3 has indeed been verified, not only to provide the authentication aspect as hoped, but moreover to be secure in the sense of the assumptions in the Theorem above, using a model-checker in [13]. Thus we can actually apply the theorem here. In particular, we can compose both aspects and obtain a composed *secure authentication* aspect. Applying this aspect, again given the saneness assumptions of the theorem, now results into a system which provides both secrecy and authentication.

As a result, we outline our methodology in the following steps:

i. Specify the base component(s) and aspects as state machines
ii. Define the composition order and the composition tables
iii. Relate the state machines renaming their channels accordingly
iv. Based on the specification of each aspect and the property to explore, define a theorem
v. Based on the channel histories find a proof for the theorem above.

7 Model Verification Framework

To explain the use of verification tools together with our proposal, consider a UML model annotated with desired security properties. Such properties in this case are formulated in UMLsec which is a *light-weight* extension to the UML, and constitutes a flexible framework to define dynamic and static properties in UML diagrams. UMLsec specifies important security properties such as the stereotype `<<high>>` that denotes dependencies that are supposed to provide the respective security requirement for the data that is sent along them as arguments or return values of operations or signals. The stereotype `<<encrypted>>` in a deployment diagram denotes the kind of communication link and the associated threats in view of a default or insider attacker. The stereotypes are explaind in depthin [12]. In Fig. 10, we have a dependency between `web` and `bank` tagged with both stereotypes. The UML model of the Web Store was checked against internal and external attackers particularly in view of the stereotype `<<high>>`. It followed that the link between web and bank were prone to attacks, we focus on this particular result. Therefore, an encryption mechanism to make the channel secure along with an authentication mechanism was needed.

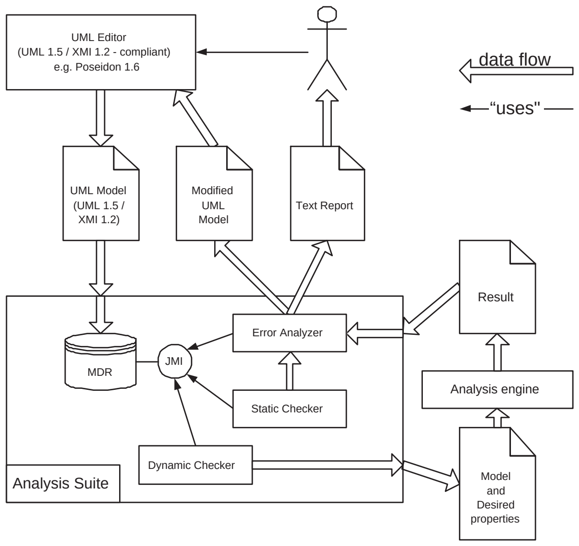


Fig. 7. UML Verification Framework

The framework in Fig. 7 is used as follows. The developer creates a model and stores it in the UML 1.5 /XMI 1.2 file format. The file is imported by the UML verification framework into the internal MetaData Repository (MDR) repository. MDR is an XMI-specific data-binding library which directly provides a representation of an XMI file on the abstraction level of a UML model through Java interfaces (JMI). This allows the developer to operate directly with UML concepts, such as classes, statecharts, and stereotypes. It is part of the Netbeans project [14]. Each plug-in accesses the model through the JMI interfaces generated by the MDR library, they may receive additional textual input, and they may return both a UML model and textual output. The two exemplary analysis plug-ins proceed as follows: The static checker parses the model, verifies its static features, and delivers the results to the error analyzer. The dynamic checker translates the relevant fragments of the UML model into the automated theorem prover input language. The automated theorem prover is spawned by the UML framework as an external process; its results are delivered back to the error analyzer. The error analyzer uses the information received from the static checker and dynamic checker to produce a text report for the developer describing the problems found, and a modified UML model, where the errors found are visualized. Besides the automated theorem prover binding presented in [15] there are other analysis plugins including a model-checker binding [16] and plugins for simulation and test-sequence generation.

The framework is designed to be extensible: advanced users can define stereotypes, tags, and first-order logic constraints which are then automatically translated to the automated theorem prover for verification on a given UML model. Similarly, new adversary models can be defined.

The framework and its application on secure software development are further described in [17,15]. The user webinterface and the source code of the verification framework is accessible at [18].

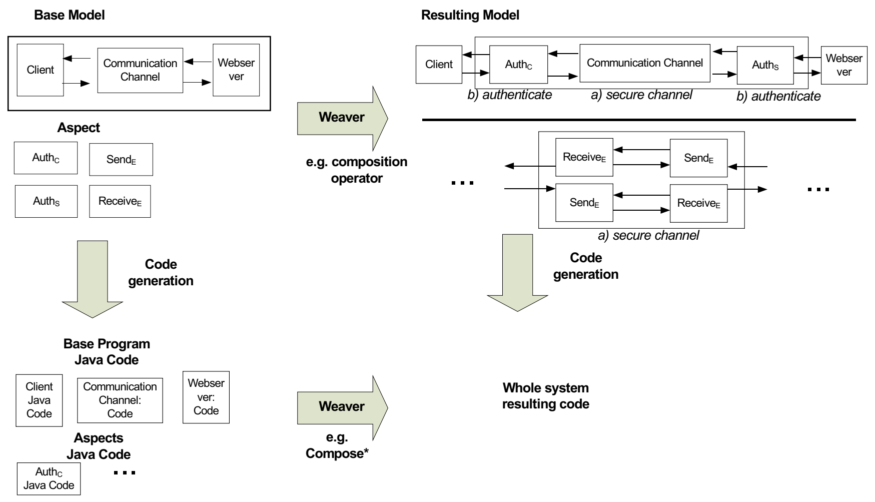


Fig. 8. Overview of the Methodology: Model and code composition i. e. weaving

8 Reusing Aspects

In Sect. 4 we introduced our two security aspects on the case of a banking system. The idea was to discuss on the one hand the importance of these security protocols, and on the other to abstract from the problem a generic secure channel and an authentication aspect. In this part, we use these two aspects on the case of a web store design. This shows how our method allows for the definition and verification of security aspects that may later be (re)used as building blocks on existing software models, such as the web store whose deployment diagram is shown in Fig. 10.

The UML model of the Web Store was checked using the tools introduced in Sect. 7. It was checked against internal and external attackers particularly in view of the stereotype `<<high>>`. The verification showed that the link between `web` and `bank` were prone to insider attacks, we focus on this particular result. Therefore, an encryption mechanism to make the channel secure is needed.

A class diagram of the Web Store is shown in Fig. 9. In order to illustrate the use of our specifications as aspects we relate the State Machines $Auth_C$, and $Auth_S$ to the *join points* in the message flow of Fig. 10, namely, the communication points where a transaction between bank and web is realized. Moreover, we relate $Send_E$, $Receive_E$, to the message calling of $Auth_C$, and $Auth_S$ and view it as a *Shared Join Point*, meaning a Join Point at which two aspects interact. We illustrate the use of these specifications as aspects by relating them to an implementation of the web store design using Compose* as code level aspect weaver. Please note that we propose aspect weaving at two levels as a that complement each other. We consider managing aspects at the modeling level (upper part of Fig. 8), weaving them with the tables and composition operator explained in previous chapters, and generate code from the resulting model. We may also, specify the individual aspects, the base components, and generate code from these. Then, weave it with a code level aspect weaver (lower part ofFig. 8). The expected result should be equivalent, though proving this belongs to future work.

We explore the second proposal mentioned above. That is to say, specifying the components, transforming these to code, and weaving them at the code level. Yet, the composition of security components is verified at the model level and weaved after its translation to code. We assume the translation to code preserves the specifications in the state machines.

A sample code of the `Dress 4 Less` web store is implemented on J# (.Net), the pointcut designator is defined as the `PaymentSecurity` concern in Compose* and is shown in listing 1.1. This concern specifies two filters, *authenticate_filter* referring to the authentication aspect, and *encrypt_filter*. In the *superimposition* section of the concern (line 16 in listing 1.1) we define the methods of the classes `Bank` and `Web` from Fig. 9 into which the security aspects are superimposed i. e. weaved. The encryption filters ($[\![Send_e]\!]$, $[\![Receive_E]\!]$) are composed with the authentication filter ($[\![Auth_C]\!]$,$[\![Auth_S]\!]$), this is the *shared join point* whose composition we outlined in Sect. 5 and analyzed in Sect. 6.

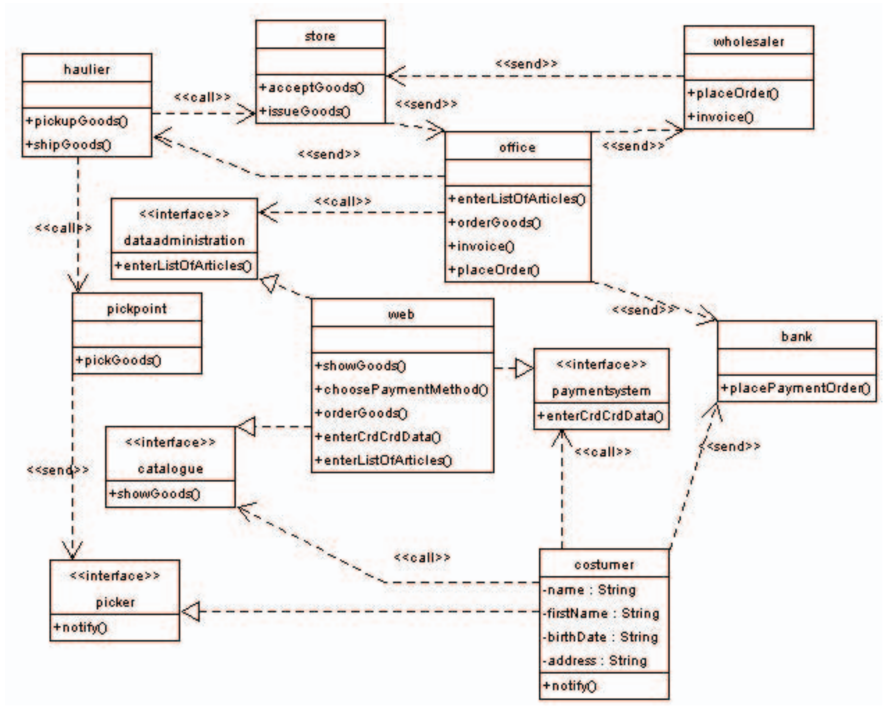


Fig. 9. Class Diagram Web Store

The state machine specifications can be translated into Java code and defined in classes `authenticate` and `secureChannel` in package `dress4Less.SecurityProtocols`. This way security aspects can be first defined and formally analyzed, and later on weaved onto a given program (or model) with the use of an aspect weaver such as Compose* in the case of the code level. Fig. 8 illustrates the relation between model and code weaving as a complementary process. The base concern on the upper left part is either a component model or a UML one, this is transformed by composing the security aspects on it. In the case of components by the composition operator defined in Sect. 2 and some pointcut designator as in Table 1 in the case of code by defining a concern as in listing 1.1. Code generation can happen either before or after composing the aspects in the model (upper part of the figure). In the case we explored above, the UML model is translated into code and the state machines also, the weaving process is performed at the code level. Please note that the analysis of aspect interaction between the secure channel and authentication was performed at the model level and its suggested translation to code is performed after its composition (right hand side of Fig. 8).

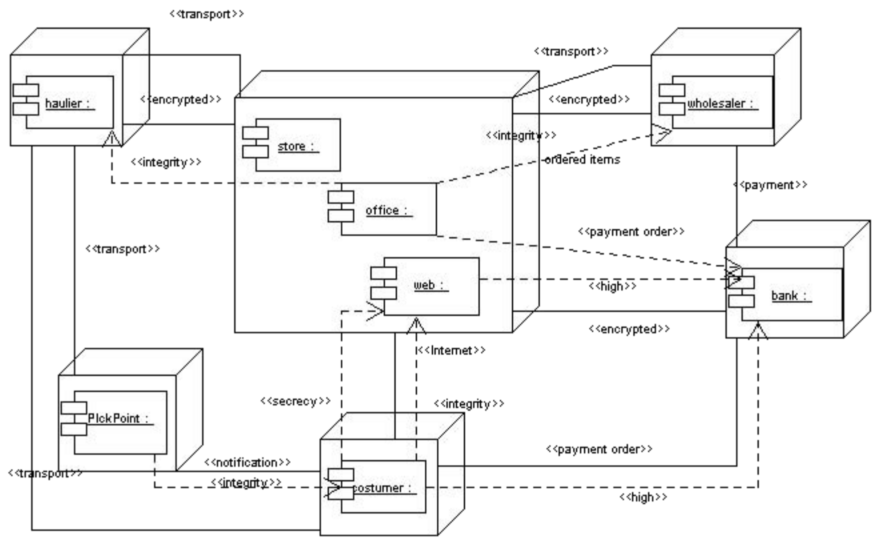


Fig. 10. Deployment Diagram with security specification stereotypes (Web Store)

Selection of the join point might be made, for instance, by picking the stereotypes `<<high>>` and `<<encrypted>>`. In this case, when we have both of them on a communication line between packages in the deployment diagram shown in Fig. 10, we select the payment related methods that are called. This is expressed in lines 11, 12, and 14 of listing 1.1. Namely, from the class diagram `placePaymentOrder`, `choosePaymentMethod`, and `sendAuthenticated` that is the implementation of the authentication aspect with the encryption protocol i. e. the implementation of the secure channel. We may select other methods in classes such as `Costumer` if needed, based on the verification analysis or as a result of changing requirements. Weaving the secure channel is performed at the code level using Compose* as pointcut delimiter (see listing 1.1). This way security aspects might be verified and re-used.

Listing 1.1. Code pointcut delimiter: Compose*

```
concern PaymentSecurity in dress4Less
{ filtermodule securePayment
 { internals
   authentication : dress4Less.SecurityProtocols. ↩
       authenticate;
   encryption : dress4Less.SecurityProtocols. ↩
       secureChannel;
  conditions
   authenticated :
     dress4Less.SecurityProtocols.authenticate. ↩
         clientAuthenticated();
  inputfilters
   authenticate_filter : Dispatch = {True =>
          [*.placePaymentOrder]authentication. ↩
              paytobankAuth,
          [*.choosePaymentMethod]authentication. ↩
              choosePaymentAuth};
   encrypt_filter : Dispatch = {True =>
          [*.sendAuthenticated]encryption.encChannel}
  }
 superimposition
 { selectors
   payment = { *=dress4Less.Bank, *=dress4Less.Web,
     *=dress4Less.SecurityProtocols.authenticate};
   filtermodules
  payment <- securePayment;
 }
}
```

9 Related Work

This work is a continuation from previous work, where we considered analyzing crosscutting concerns on a formal theory [9], transforming models with the use of AOSD [10] as well as work where we considered how to weave in security aspects on the transition from models to code [19].

Most aspect-oriented approaches address the aspect interaction problem at the programming and language level. However, the particular issue of semantic analysis of CF compositions is still an open question, and to the best of our knowledge has not been formally achieved combining CF and a formal theory. Related work on aspect interaction can be found in [20], however the authors restrict themselves to a global overview of the problem and present no results comparable to ours. Nagy et al [21] explore aspect interaction and propose a set of requirements for composing aspects at so-called *shared join points*, yet their analysis is rather syntactic.

Although some work toward a formal foundation has been carried out in the case of Aspect/J [22], no formal foundation has been published so far for

Composition Filters to the best of our knowledge. The tool SyncGen assisting aspect-oriented development with semantic analysis has been presented in [23]. The tool automatically synthesizes synchronization implementations from high-level specifications using first-order logic.

10 Conclusions

This work introduced a composition model that allows us to provide concerns with a (formal) syntax and semantics. In this way we might be able to pose the problem of aspect composition with other aspects (i. e. aspect interaction) at a formal level. This work suggests that building a composition model on the concept of *I/O behavior* and the composition operators for *stream-processing functions*, we may at least explore under which conditions aspects may be safely composed. This framework allowed us to consider aspect analysis at the semantic level, which up to the present time has been mostly performed syntactically. Moreover, this work also proposed a generic secrecy-enforcing channel (called secure channel through this work as explained in the introduction) that has been formally verified. Altogether, we introduced a formal model for composition filters and aspect interaction analysis, a generic secure channel aspect composed with an authentication protocol, and an analysis of their interaction in view of preservation of security properties during aspect composition.

Although in the current work we considered security aspects in particular, we propose that our method can be generalized. Future work will consider under which assumptions security is composable in general terms. We did not explore the issue in the present paper. We therefore totally agree on the need to further assess the generality of this proposal.

Acknowledgments. Thanks to Prof. Mehmet Aksit for valuable insights and remarks on the relation of our work to aspect-orientation, special thanks to Prof. Manfred Broy for discussions on the formal foundation and in particular on component composition and modeling. Thanks also to Robin Laney for his remarks on the formal foundation, and our colleagues: Leonid Kof and Sebastian Winter for their questions helping better convey the contents of our work.

References

1. Bergmans, L., Aksit, M., Tekinerdogan, B.: Aspect composition using composition filters. In Aksit, M., ed.: Software Architectures and Component Technology. Kluwer Academic Publishers (2001) 357–384
2. Filman, R.E., Friedman, D.P.: Aspect-Oriented Programming is Quantification and Obliviousness. In Filman, R., Elrad, T., Clarke, S., Aksit, M., eds.: Aspect-Oriented Software Development. Addison-Wesley (2004) 21–35
3. Broy, M.: A logical basis for component-based systems engineering. In Broy, M., Steinbrüggen, R., eds.: Calculational System Design. Volume 158 of NATO ASI Series, Series F: Computer and System Sciences. IOS Press (1999)
4. Broy, M., Stoelen, K.: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer-Verlag (2001)

5. Abadi, M., Lamport, L.: Composing specifications. ACM Trans. Program. Lang. Syst. **15** (1993) 73–132
6. Broy, M. In: From States to Histories: Relating States and History Views onto Systems. Volume 180 of Springer NATO ASI Series. IOS Press (2001) 149 – 186
7. Composestar project: (http://composestar.sf.net)
8. van den Berg, K., Conejero, J.M., Chitchyan, R.: Aosd ontology 1.0 - public ontology of aspect-orientation. AOSD-Europe-UT-01 D9, AOSD-Europe (2005)
9. Fox, J.: A Formal Foundation for Aspect-Oriented Software Development. Research on Computing Science, CIC-IPN, ISSN: 1665-9899 **14** (2005) 241–251
10. Fox, J., Jürjens, J.: Introducing Security Aspects with Model Transformation. In: Model Based Development (MBD) Workshop in Proc. 12th Annual IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2005), Greenbelt, Washington, 4-5 April, IEEE Computer Society (2005) 543 – 549
11. Aksit, M., Bergmans, L., Vural, S.: An object-oriented language-database integration model: The composition-filters approach. In Madsen, O.L., ed.: Proc. 7th European Conf. Object-Oriented Programming, Springer-Verlag (1992) 372–395
12. Jürjens, J.: Secure Systems Development with UML. Springer (2004)
13. Grünbauer, J., Hollmann, H., Jürjens, J., Wimmel, G.: Modelling and Verification of Layered Security Protocols: A Bank Application. In: SAFECOMP 2003, 23-26 September 2003 Edinburgh, GB, Springer-Verlag (2003)
14. : Netbeans project. Open source. Available from http://mdr.netbeans.org (2003)
15. Jürjens, J., Fox, J.: Tools for model-based security engineering. In: ICSE '06: Proceeding of the 28th international conference on Software engineering, New York, NY, USA, ACM Press (2006) 819–822
16. Jürjens, J., Shabalin, P.: Automated verification of UMLsec models for security requirements. In Jézéquel, J.M., Hußmann, H., Cook, S., eds.: UML 2004 – The Unified Modeling Language. Volume 2460 of LNCS., Springer (2004) 412–425
17. Jürjens, J.: Sound methods and effective tools for model-based security engineering with UML. In: 27th International Conference on Software Engineering (ICSE 2005), IEEE Computer Society (2005)
18. : UMLsec tool (2002-04) Open-source. Accessible at http://www.umlsec.org.
19. Jürjens, J., Houmb, S.: Dynamic secure aspect modeling with UML: From models to code. In: ACM / IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS / UML 2005). LNCS, Springer (2005)
20. Bergmans, L.: Towards detection of semantic conflicts between crosscutting concerns. In Hannemann, J., Chitchyan, R., Rashid, A., eds.: Workshop on Analysis of Aspect-Oriented Software. ECOOP 2003 (2003)
21. Nagy, I., Bergmans, L., Aksit, M.: Composing aspects at shared join points. In Robert Hirschfeld, Ryszard Kowalczyk, A.P., Weske, M., eds.: Proceedings of International Conference NetObjectDays, NODe2005. Volume P-69 of Lecture Notes in Informatics., Erfurt, Germany, Gesellschaft für Informatik (GI) (2005)
22. David Walker, S.Z., Ligatti, J.: A Theory of Aspects. In: ACM SIGPLAN International Conference on Functional Programming, Uppsala, Sweden, August. (2003)
23. Deng, X., Dwyer, M., Hatcliff, J., Mizuno, M.: Syncgen: An aspect-oriented framework for synchronization. In Jensen, K., Podelski, A., eds.: TACAS. Volume 2988 of LNCS., Springer (2004) 158–162