

Modeling and enforcing secure object flows in process-driven SOAs: an integrated model-driven approach

Bernhard Hoisl · Stefan Sobernig · Mark Strembeck

Received: 7 October 2011 / Revised: 25 April 2012 / Accepted: 13 July 2012 / Published online: 5 October 2012
© Springer-Verlag 2012

Abstract In this paper, we present an integrated model-driven approach for the specification and the enforcement of secure object flows in process-driven service-oriented architectures (SOA). In this context, a secure object flow ensures the confidentiality and the integrity of important objects (such as business contracts or electronic patient records) that are passed between different participants in SOA-based business processes. We specify a formal and generic meta-model for secure object flows that can be used to extend arbitrary process modeling languages. To demonstrate our approach, we present a UML extension for secure object flows. Moreover, we describe how platform-independent models are mapped to platform-specific software artifacts via automated model transformations. In addition, we give a detailed description of how we integrated our approach with the Eclipse modeling tools.

Keywords Process modeling · Secure object flows · Security engineering · Service-oriented architecture · Model-driven development · UML · SoaML · Web services

1 Introduction

1.1 Motivation

Business processes define an organization's operational procedures and are performed to reach operational goals. In recent years, service-oriented architectures (SOA; see, e.g., [29,62,64]) are increasingly used in the area of business process management. In this context, a process-driven SOA (see, e.g., [96]) is specifically built to support the definition, the execution, and monitoring of intra-organizational and cross-organizational business processes. The widespread use of service-oriented technologies also led to demands for a thorough integration of security features in the development process of service-oriented systems.

In particular, IT systems must comply with certain laws and regulations, such as the Basel II Accord, the International Financial Reporting Standards (IFRS), or the Sarbanes-Oxley Act (SOX). For example, adequate support for the definition and the enforcement of process-related security policies is one important part of SOX compliance (see, e.g., [7,9,43]). Corresponding compliance requirements also arise from security recommendations and standards, such as the NIST security handbook [45], the NIST recommended security controls [49], or the ISO 27000 standard family [16–18] (formerly ISO 17799). Legally binding agreements, such as business contracts, or company-specific (internal) rules and regulations do also have a direct impact on corresponding information systems (see, e.g., [87]).

Communicated by Dr. Juan M. Vara, Mike Papazoglou and Il-Yeol Song.

This work has partly been funded by the Austrian Research Promotion Agency (FFG) of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) through the Competence Centers for Excellent Technologies (COMET K1) initiative and the FIT-IT program.

B. Hoisl (✉) · S. Sobernig (✉) · M. Strembeck (✉)
New Media Lab, Institute for Information Systems,
Vienna University of Economics and Business (WU Vienna),
Vienna, Austria
e-mail: bernhard.hoisl@wu.ac.at

S. Sobernig
e-mail: stefan.sobernig@wu.ac.at

B. Hoisl · M. Strembeck
Secure Business Austria Research (SBA Research),
Vienna, Austria
e-mail: mark.strembeck@wu.ac.at

Yet, modeling process-related security properties yields different types of problems. First, contemporary modeling languages such as Business Process Model and Notation (BPMN, [52]), Event-driven Process Chain (EPC, [70]), or Unified Modeling Language (UML) activity models [59] do not provide native language constructs to model security features. A second problem is that the language used for process modeling is often different from (or not integrated with) the system modeling language that is used to specify the corresponding software system. This, again, may result in problems because different modeling languages provide different language abstractions that cannot easily be mapped to each other. In particular, such semantic gaps may involve significant efforts when conceptual models from different languages need to be integrated and mapped to a software platform (see, e.g., [4, 34, 95]). However, a complete and correct mapping of process definitions and related security properties to the corresponding software system is essential in order to assure consistency between the modeling-level specifications on the one hand, and the software system that actually manages corresponding process instances and enforces the respective security properties, on the other hand.

In this paper, we are especially concerned with the confidentiality and the integrity of object flows in process-driven systems. *Confidentiality* ensures that important/classified objects (such as court records, business contracts, or electronic patient records) which are used in a business process can only be read by designated subjects (see, e.g., [8, 49]). *Integrity* ensures that important objects are in their original/intended state, and enables the straightforward detection of accidental or malicious changes (see, e.g., [45, 50, 69]). At the modeling-level, an *object flow* defines that an object is passed from one node in a business process model to another. In a process-driven SOA, the corresponding object flow is then implemented via different messages that are passed between different software services. In the remainder of this paper, we use the term *secure object flow* to refer to an object flow whose confidentiality and/or integrity is ensured via cryptographic mechanisms.

1.2 Approach synopsis

We use model-driven development (MDD) techniques (see, e.g., [77, 79, 82]) to provide an integrated, tool-supported approach for the definition, for the deployment, and for the execution of secure object flows in process-driven SOAs. In the context of MDD, a computation-independent model (CIM) defines a certain domain (or sub-domain) at a generic level. The CIM is independent of a particular modeling language or technology. A CIM can be used to build a platform-independent model (PIM) of the corresponding domain. While it is independent of any platform, and thereby neutral from an implementation point of view, the PIM is typically

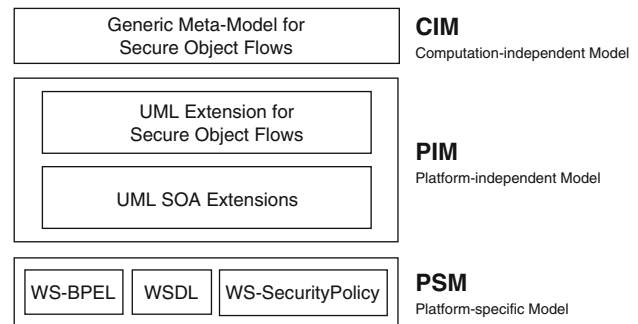


Fig. 1 The secure object flows approach covers the CIM, PIM, and PSM layers

specified in a particular modeling language (such as BPMN or UML) and describes the structure of a system, the elements/results that are produced by a system, or the control and object flow in a system. Finally, a platform-specific model (PSM) describes the realization/implementation of a software system via platform-specific technologies and tools.

Our work on *secure object flows* presented in this paper is an integrated approach which covers the CIM, PIM, and PSM layers (see Fig. 1). At the CIM layer, we provide a generic metamodel for secure object flows that can be used to extend arbitrary process modeling languages. At the PIM layer, we provide a UML extension that allows to model secure object flows via extended activity diagrams. Moreover, we integrate our extension with the SoaML [58] and UML4SOA [38] to enable the definition of secure object flows for process-driven SOAs. At the PSM layer, we generate WS-BPEL [61], WSDL [92], and WS-SecurityPolicy [63] specifications from the PIMs.

To enable the specification and the implementation of secure object flows in process-driven SOAs, we provide an integrated tool support for our approach based on the Eclipse IDE [10]. Figure 2 gives an overview of our tool support on different abstraction levels for the definition and for the implementation of secure object flows. In particular, we apply model transformations [42, 80] to automatically generate executable, platform-specific service descriptions that are deployed in a SOA process engine. At the topmost layer, our tool supports the definition of security-enhanced *Business Process* models via UML activity diagrams (see Fig. 2). At the *SOA Models* level, the service-oriented architecture is modeled via component structures, service activities, message types, as well as service and invocation protocols. *Web Service Artifacts* (such as WS-BPEL, WSDL, or WS-SecurityPolicy specifications) are derived from SOA models through automatic model transformations. Finally, these artifacts are deployed for execution in a process-driven *Runtime Environment*.

Our contribution is based on previous publications concerning the modeling of secure object flows [27] and its adop-

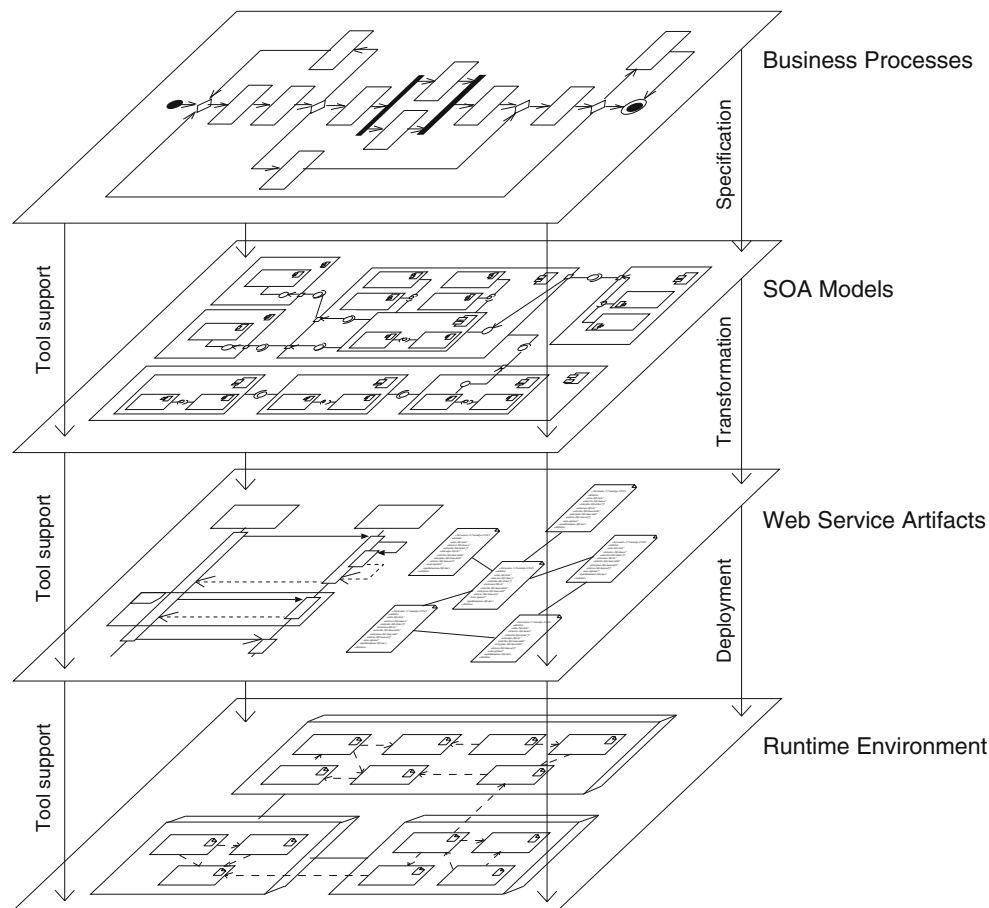


Fig. 2 Integrated tool support for the definition and implementation of secure object flows

tion for SOA modeling [26]. These previous contributions do, however, only discuss specific and limited modeling options at the PIM level. They do neither provide a generic CIM nor integrated PIM models, or tool support. In this paper, we extend our previous contributions via MDD techniques to build an integrated approach (see Figs. 1, 2) for the specification and for the enforcement of secure object flows in process-driven SOAs. We provide a thorough description of the capabilities of our approach to model secure object flows in SOAs, both at a generic and at the modeling language level. In addition, we present our tool support (including automated model transformations) for the specification and for the deployment of secure object flows.

The remainder of this paper is structured as follows. Section 2 discusses the general characteristics of secure object flows and Sect. 3 defines a formal and generic CIM. Next, we describe our PIM which consists of a generic UML extension for secure object flows (see Sect. 4) as well as an integration of secure object flows with SOA-based modeling primitives (see Sect. 5). Subsequently, Sect. 6 presents tool support for our approach and describes automated model

transformations that produce PSM artifacts from corresponding PIMs. Finally, Sect. 7 discusses related work and Sect. 8 concludes the paper.

For the sake of readability, we moved the formal constraints that define the semantics of our UML extension to Appendices A and B.

2 Characteristics of secure object flows

Process models typically have (implicit or explicit) token semantics, and object tokens are passed along object flow edges. Thus, to ensure the consistency of the corresponding process models, it is especially important to thoroughly specify the semantics of secure object flows with respect to control nodes (such as fork, join, decision, and merge nodes).

In general, a *secure object flow* consists of one or more arcs in a business process model that transport important, security-sensitive workflow objects (e.g., electronic patient records or business contracts) between two *secure nodes* of the respective process model. In particular, we have to ensure

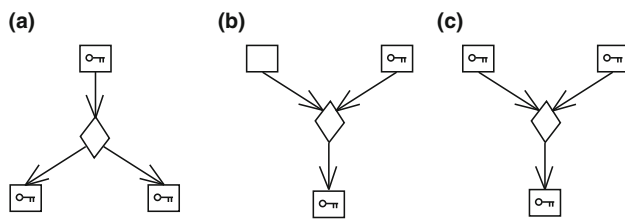


Fig. 3 Secure object flows with decision and merge nodes

that the security attributes determined by the source node of a secure object flow (such as the confidentiality algorithm used to encrypt the corresponding objects) are understood by the respective target node(s). In this context, control nodes (fork, join, decision, merge) are of special importance because they influence the semantics of secure object flows. Below, we give an overview of the impact that different configurations of control nodes have on the corresponding secure object nodes. Subsequently, Sect. 3 provides generic definitions that formally define the semantics of secure object nodes at the CIM level.

Figure 3 shows examples for the different configuration options of secure object flows that include decision or merge nodes.¹ In the subsequent figures, a rectangle including a key symbol represents a secure object node, while a blank rectangle represents an ordinary object node.

Figure 3a shows a configuration in which a decision node has an incoming secure object flow and presents the corresponding object tokens to multiple outgoing edges. As the source of the incoming object flow is a secure node both target nodes must also be secured. Otherwise, a secure object flow could have a secure node as its source and an ordinary object node as its target, which would result in an inconsistency because ordinary object nodes cannot ensure the confidentiality or the integrity of object tokens. Furthermore, target nodes of a secure object flow must support the same security properties as the respective source node. This constraint ensures that (1) security properties cannot be lost when traversing a decision node and that (2) the target node(s) are able to check and to ensure the corresponding security properties.

Figure 3b shows a configuration where a merge node brings together different flows, one of which is a secure object flow. For such a configuration, we define that if a merge node receives at least one secure object flow, the target node of this merge node must also be a secure node. This constraint guarantees that each secure object token passing a merge node can be checked and processed by the corresponding target node.

¹ For the sake of simplicity, Figs. 3 and 4 show only two incoming/outgoing flows for the respective control nodes. However, the corresponding discussion equally applies to an arbitrary number of incoming/outgoing edges, of course.

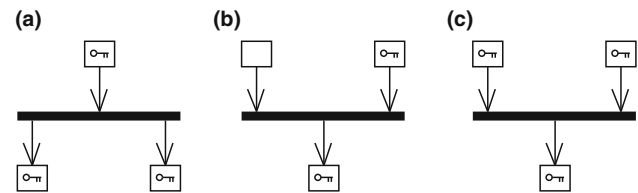


Fig. 4 Secure object flows with fork and join nodes

Figure 3c shows a configuration where a merge node brings together different secure object flows. In this case, the target must also be a secure node. Furthermore, we define that all source nodes must provide compatible security properties, i.e., the nodes must support the same confidentiality and/or integrity algorithms. In addition, the target node must support all security properties of the respective source nodes. Otherwise, incompatibilities could emerge if the security properties supported by the source nodes are different from the security properties supported by the target node.

Figure 4 shows examples for the different configuration options of secure object flows that include fork or join nodes. Figure 4a shows a configuration where a fork node splits a secure object flow into multiple concurrent flows. Because the tokens arriving at a fork node are duplicated, all target nodes must be secure nodes. Furthermore, the target nodes must support the same security properties as the corresponding source node. This constraint ensures (1) that security properties cannot be lost when traversing a fork node and (2) that the target node(s) are able to check and ensure the corresponding security properties.

Figure 4b shows a configuration where a join node synchronizes multiple object flows, one of which is a secure object flow. We define that if a join node receives at least one secure object flow, then the target node of this join node must also be a secure node. This constraint guarantees that each secure object token passing a join node can be checked and processed by the corresponding target node.

Figure 4c shows a configuration where a join node synchronizes multiple secure object flows. In such a situation, the target must also be a secure node. Furthermore, all source nodes and the target node must support compatible security properties. Otherwise, inconsistencies could emerge if the security properties supported by the source nodes are different from the security properties supported by the target node.

The examples from Figs. 3 and 4 only include a single control node, respectively. However, in principle, the path from one secure object node to another secure object node may include an arbitrary number of control nodes. Figure 5 shows examples of such configurations. In case a path between two secure object nodes includes two or more intermediate control nodes, we also have to ensure that the source and the

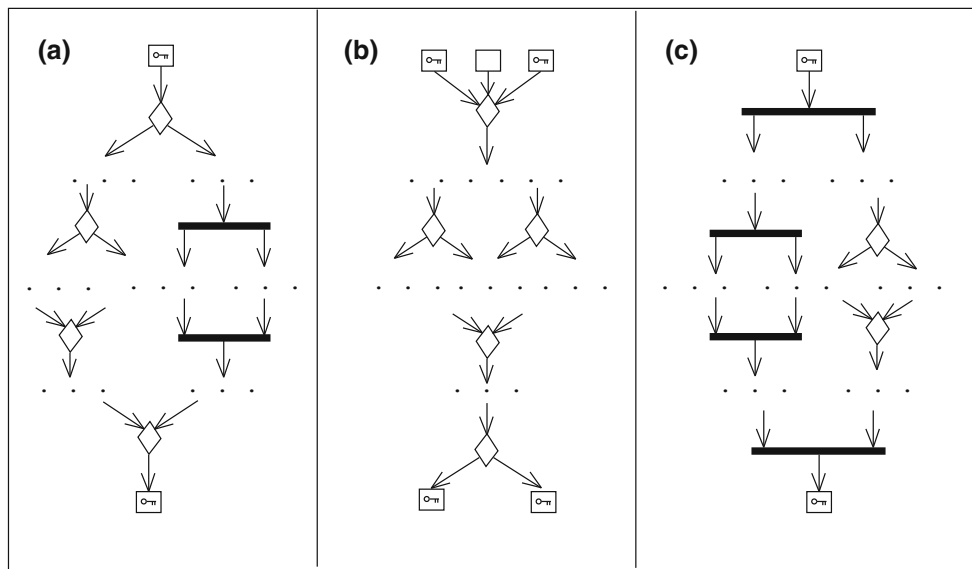


Fig. 5 Secure object flows with an arbitrary number of intermediate control nodes

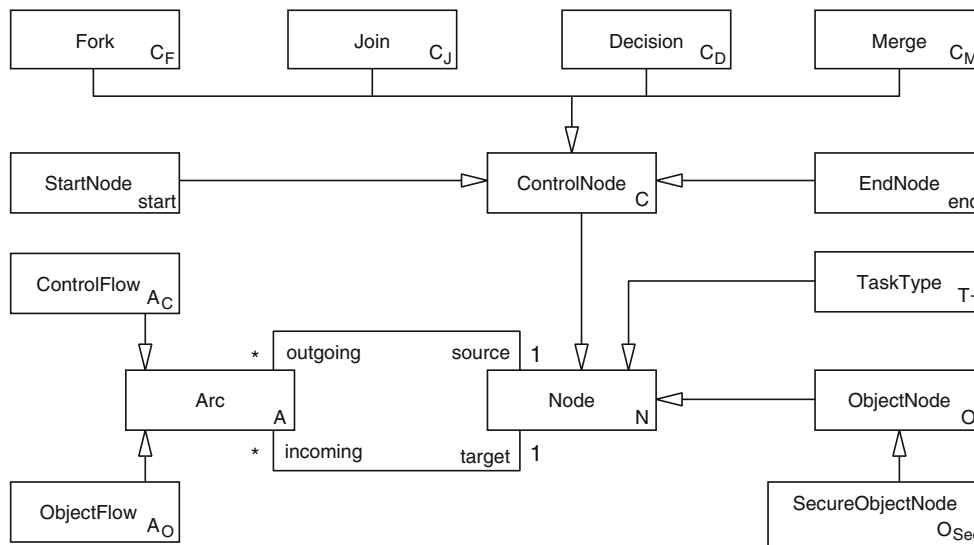


Fig. 6 Conceptual overview: main elements of Business Activity process flows (see also [85])

target nodes of the respective path provide compatible security features.

After discussing the constraints for secure object flows on the above examples, Sect. 3 now provides a formal and generic metamodel (CIM) for secure object flows.

3 A formal and generic metamodel for secure object flows

Figure 6 shows the basic elements of Business Activity process flows [85] and the main relations between these elements as a MOF-compliant structural diagram [53]. While this graphical model only gives an overview, we now pro-

vide a formal specification of the process flow model. The formal definitions below complement the definitions from [85].

Definition 1 (Business Activity Process Flow Model) A Process Flow Model $PFM = (N, A, S, Y)$ where $N = T_T \cup C_F \cup C_J \cup C_D \cup C_M \cup O \cup O_{Sec} \cup \{start, end\}$ and $A = A_C \cup A_O$ refer to pairwise disjoint sets of the metamodel, $A \subseteq N \times N$ refers to a set of arcs that connect nodes, $S = S_C \cup S_I$ refers to pairwise disjoint sets of security attributes, and $Y = in \cup out \cup source \cup target \cup ofpath \cup successors \cup predecessors \cup ca \cup ia$ refers to mappings that establish relationships such that

- an element of N is called *node* and an element of A is called *arc*;
- an element of A_C is called *control flow* and an element of A_O is called *object flow*;
- an element of S_C is called *confidentiality algorithm* and an element of S_I is called *integrity algorithm*;
- an element of T_T is called *task type*;
- an element of O is called *object node* and an element of O_{Sec} is called *secure object node* with $O_{Sec} \subseteq O$;
- an element of $C = C_F \cup C_J \cup C_D \cup C_M$ is called *control node*. An element of C_F is called *fork*, an element of C_J *join*, an element of C_D *decision*, and an element of C_M *merge*;
- *start* is called *start node* and *end* is called *end node*;
- all nodes $n \in N$ are on a path from *start* to *end*.

Below, we iteratively define the partial mappings of the Business Activity Process Flow Model and provide corresponding formalizations (\mathcal{P} refers to the power set):

1. The mapping $in : N \mapsto \mathcal{P}(A)$ is called **incoming arc**. For $in(n) = A_{in}$ with $n \in N$ and $A_{in} \subseteq A$ we call each $a \in A_{in}$ an incoming arc of node n .
2. The mapping $out : N \mapsto \mathcal{P}(A)$ is called **outgoing arc**. For $out(n) = A_{out}$ with $n \in N$ and $A_{out} \subseteq A$ we call each $a \in A_{out}$ an outgoing arc of node n .
3. The mapping $source : A \mapsto N$ is called **source node**. For $source(a) = n$ with $a \in A$ and $n \in N$ we call n the source node of arc a .
4. The mapping $target : A \mapsto N$ is called **target node**. For $target(a) = n$ with $a \in A$ and $n \in N$ we call n the target node of arc a .
5. The mapping $ofpath : (O \times O) \mapsto \mathcal{P}(A_O)$ is called **object flow path**. For $ofpath(o_s, o_t) = A_{path}$ with $o_s, o_t \in O$ and $A_{path} \subseteq A_O$ we call o_s source node, o_t target node, and each $a \in A_{path}$ is an arc on the path from o_s to o_t . Thus, an object flow path between two object nodes o_s and o_t must only include arcs or control nodes, it must not include intermediary tasks or (other) object nodes. Therefore, the following consistency requirements must hold for each object flow path:

- An object flow path connects the source node o_s and the target node o_t via an arbitrary number of arcs and intermediary control nodes, therefore: $\forall a \in A_{path} : source(a) = o_s \vee source(a) \in C$ and $\forall a \in A_{path} : target(a) = o_t \vee target(a) \in C$.
- The first arc a_{first} in an object flow path is an outgoing arc of the source node o_s , therefore: $\exists a_{first} \in A_{path} : source(a_{first}) = o_s$.
- Each object flow path includes exactly one outgoing arc of the source node o_s , therefore: $\forall a_1, a_2 \in A_{path} : a_1 \in out(o_s) \wedge a_2 \in out(o_s) \Rightarrow a_1 = a_2$.

- In an object flow path, the source node o_s has no incoming arcs, therefore: $\forall a \in A_{path} : a \notin in(o_s)$.
 - The last arc a_{last} in an object flow path is an incoming arc of the target node o_t , therefore: $\exists a_{last} \in A_{path} : target(a_{last}) = o_t$.
 - Each object flow path includes exactly one incoming arc of the target node o_t , therefore: $\forall a_1, a_2 \in A_{path} : a_1 \in in(o_t) \wedge a_2 \in in(o_t) \Rightarrow a_1 = a_2$.
 - In an object flow path, the target node o_t has no outgoing arcs, therefore: $\forall a \in A_{path} : a \notin out(o_t)$.
6. The mapping $successors : O \mapsto \mathcal{P}(O)$ is called **succeeding object nodes**. For $successors(o_s) = O_{succ}$ with $o_s \in O$ and $O_{succ} \subseteq O$ we call o_s source node and each $o_t \in O_{succ}$ a direct successor of o_s . In particular, O_{succ} is the set of object nodes for which a path exists between o_s and each $o_t \in O_{succ}$. Formally: $\forall o_s \in O, o_t \in successors(o_s) : ofpath(o_s, o_t) \neq \emptyset$.
 7. The mapping $predecessors : O \mapsto \mathcal{P}(O)$ is called **preceding object nodes**. For $predecessors(o_t) = O_{pre}$ with $o_t \in O$ and $O_{pre} \subseteq O$ we call o_t target node and each $o_s \in O_{pre}$ a direct predecessor of o_t . In particular, O_{pre} is the set of object nodes for which a path exists between each $o_s \in O_{pre}$ and o_t . Formally: $\forall o_t \in O, o_s \in predecessors(o_t) : ofpath(o_s, o_t) \neq \emptyset$.
 8. The mapping $ca : O_{Sec} \mapsto S_C$ is called **confidentiality algorithm**. For $ca(o_s) = s_c$ with $o_s \in O_{Sec}$ and $s_c \in S_C$ we call s_c the confidentiality algorithm used by o_s .
 9. The mapping $ia : O_{Sec} \mapsto S_I$ is called **integrity algorithm**. For $ia(o_s) = s_i$ with $o_s \in O_{Sec}$ and $s_i \in S_I$ we call s_i the integrity algorithm used by o_s .

A *direct object flow* consists of a single arc that directly connects two object nodes without intermediary control nodes, i.e., $A_{DO} = \{a \in A_O | source(a) \in O \wedge target(a) \in O\}$. A *transitive object flow* consists of two or more arcs which connect two object nodes via an object flow path, i.e., $A_{TO} = \{a \in A_O | a \in ofpath(o_s, o_t)\}$ with $o_s, o_t \in O$. If the source of a (direct or transitive) object flow is a secure object node, i.e., $a \in A_O \wedge source(a) \in O_{Sec}$, we call this object flow a *secure object flow*.

Definition 2 Let $PFM = (N, A, S, Y)$ be a Business Activity Process Flow Model. PFM is said to be correct if the following requirements hold:

1. Each secure object node ensures either or both confidentiality and integrity: $\forall o_s \in O_{Sec} : ca(o_s) \cup ia(o_s) \neq \emptyset$.
2. The successor of a secure object node must also be a secure object node: $\forall o_s \in O_{Sec}, o_t \in successors(o_s) : o_t \in O_{Sec}$.
3. The successor of a secure object node must support the same confidentiality algorithm as the respective source

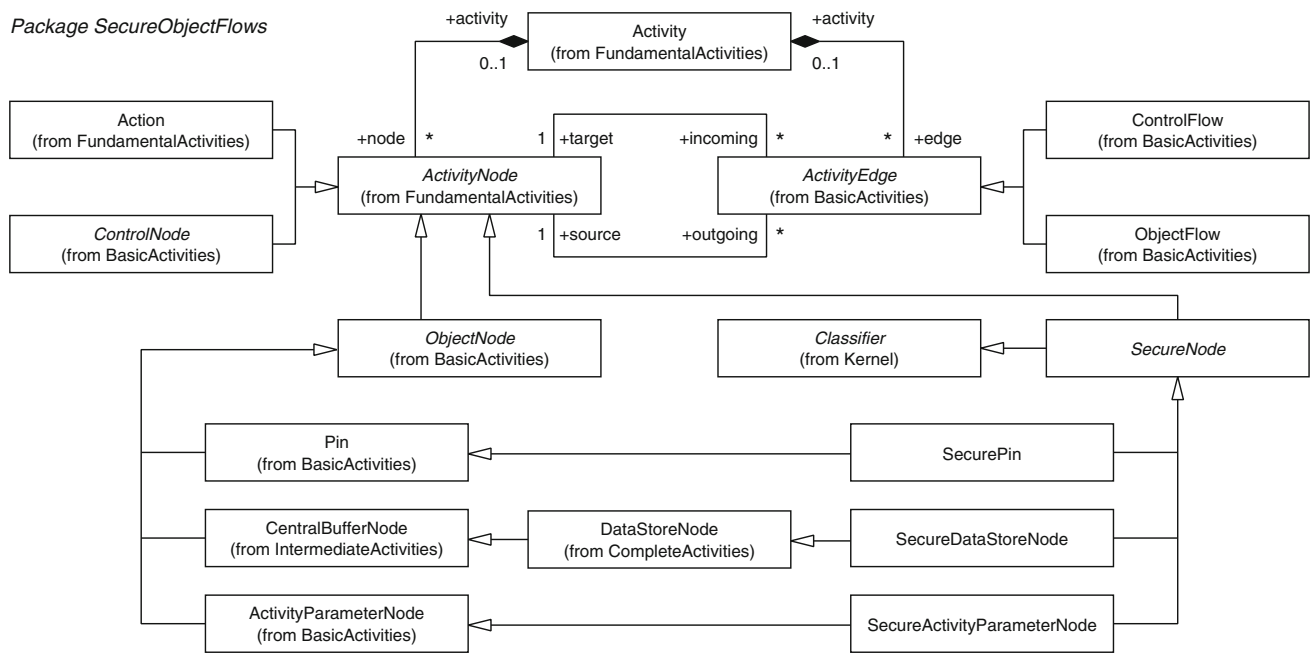


Fig. 7 UML metamodel extension for secure object flows

node: $\forall o_s \in O_{Sec}, o_t \in successors(o_s) : ca(o_s) = ca(o_t)$.

4. The successor of a secure object node must support the same integrity algorithm as the respective source node: $\forall o_s \in O_{Sec}, o_t \in successors(o_s) : ia(o_s) = ia(o_t)$.
5. The security attributes of two secure object nodes $o_{s1}, o_{s2} \in O_{Sec}$ may influence each other, even if they are not connected via a direct or via a transitive object flow. In particular, this is the case if o_{s1} and o_{s2} are predecessors of a common target object node $o_t \in O_{Sec}$. In other words, if a secure object node o_t has two or more predecessors that are also secure object nodes, then each predecessor must support the same confidentiality algorithm as the respective target node: $\forall o_t \in O_{Sec}, o_s \in predecessors(o_t) : o_s \in O_{Sec} \Rightarrow ca(o_t) = ca(o_s)$.
6. If a secure object node o_t has two or more predecessors that are also secure object nodes, then each predecessor must support the same integrity algorithm as the respective target node: $\forall o_t \in O_{Sec}, o_s \in predecessors(o_t) : o_s \in O_{Sec} \Rightarrow ia(o_t) = ia(o_s)$.

4 UML extension for secure object flows

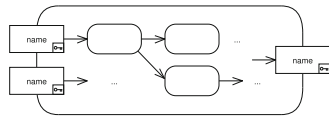
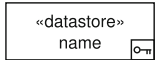
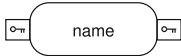
To provide modeling support for confidentiality and integrity properties of object flows at the PIM level, we define a new package called *SecureObjectFlows* as an extension to the UML metamodel (see Fig. 7). In particular, we introduce *SecureNode*, *SecurePin*, *SecureDataStoreNode*,

and *SecureActivityParameterNode* as new modeling elements. A secure object flow is defined as an object flow between two of the above mentioned secure object nodes. The *SecureNode* element is defined as an abstract node, and the *SecurePin*, *SecureDataStoreNode*, and *SecureActivityParameterNode* represent specialized secure nodes. In particular, these three node types inherit the properties from their corresponding parent object nodes as well as the security related properties from *SecureNode* (see Fig. 7).

Below, we specify the attributes of the *SecureNode* elements defined via the metamodel extension. In addition, we use the Object Constraint Language (OCL, [57]) to formally specify the semantics of the *SecureObjectFlows* package. For the sake of readability, we decided to move the associated OCL constraints to Appendix 8. However, these OCL constraints are a significant part of our UML extension, because they formally define the semantics of the new modeling elements. Therefore, each UML model that uses the *SecureObjectFlows* package must conform to these OCL constraints.²

² For some of our OCL constraints, Appendix A provides two optional OCL statements expressing identical constraints, where each of these optional constraints complies with a different version of the OCL standard. OCL Constraints 4a and 6a comply with OCL version 2.2 [56], while OCL Constraints 4b and 6b use new language constructs from the OCL 2.3.1 standard [57]. The changes affect only the *allSuccessors()* and *allPredecessors()* definitions which are interchangeable.

Table 1 Notation of elements for modeling secure objects

Node type	Notation	Explanation
SecurePin (attached to an action)		A SecurePin attached to an action is shown as a UML Pin element that includes a key symbol
SecureDataStoreNode		A SecureDataStoreNode is shown as a UML DataStoreNode element with a key symbol in the lower right corner surrounded by a small rectangle
SecureActivityParameterNode		A SecureActivityParameterNode is shown as a UML ActivityParameterNode element with a key symbol in the lower right corner surrounded by a small rectangle

- *confidentialityAlgorithm : Classifier [0..1]*
 - References a classifier that provides methods to ensure confidentiality properties of the object tokens that are sent or received by a SecureNode, e.g., a class implementing Data Encryption Standard (DES, [46]) or Advanced Encryption Standard (AES, [47]) functionalities.
- *confidentialityEnsured : Boolean [0..1]*
 - This attribute is derived from the attribute *confidentialityAlgorithm*. It evaluates to “true” if a SecureNode supports confidentiality-related security properties (see OCL Constraint 1 in Appendix A).
- *integrityAlgorithm : Classifier [0..1]*
 - References a classifier that provides methods to ensure integrity properties of the object tokens that are sent or received by a SecureNode, e.g., a class implementing SHA-1 or SHA-384 (Secure Hash Algorithm, [48]) functionalities.
- *integrityEnsured : Boolean [0..1]*
 - This attribute is derived from the attribute *integrityAlgorithm*. It evaluates to “true” if a SecureNode supports integrity-related security properties (see OCL Constraint 2).

With respect to the attributes defined above, we specify that a secure object node supports either or both confidentiality and integrity properties (see OCL Constraint 3). Table 1 shows the graphical elements for SecureNodes. Table 2 gives an overview of how each of the generic (CIM)

definitions from Sect. 3 is mapped to our UML extension (PIM) for secure object flows.

4.1 Example processes with secure object flows

Below, we show two examples that model secure object flows. In Sect. 4.1.1, we present a radiological image reading process that is conducted in a hospital. In Sect. 4.1.2, we show a simple credit application process in a bank.

4.1.1 Radiological examination process

Figure 8 shows a radiological examination process, modeled via a UML activity diagram that uses elements of the Secure-ObjectFlows package. The process starts with a *Radiological examination* action that produces images which are read in a next step. The corresponding SecurePins enforce the security properties defined in Table 3 for all *Image* object tokens traveling between the *Radiological examination* and *Image reading* actions. The attributes are derived from the SecureNode classifier defined via the SecureObjectFlows package. Note that the different attributes are properties of the corresponding SecureNodes and exist independent of their visualization in a model.³ If the images are of sufficient quality, the activity continues with two concurrent flows: the images are annotated and the patient record is fetched. Both actions produce output tokens of type *Image* and *Patient*

³ For example, an alternative visualization of SecureObjectFlows attributes would use comments/constraints attached to secure object nodes directly in an activity diagram.

Table 2 Consistency of the generic metamodel and the SecureObjectFlows UML extension

Generic definitions	Covered through
Definition 1.1: $in : N \mapsto \mathcal{P}(A)$	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7; [59])
Definition 1.2: $out : N \mapsto \mathcal{P}(A)$	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7; [59])
Definition 1.3: $source : A \mapsto N$	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7; [59])
Definition 1.4: $target : A \mapsto N$	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7; [59])
Definition 1.5: $ofpath : (O \times O) \mapsto \mathcal{P}(A_O)$	Implicitly defined via our metamodel extension, the specification of UML activity models (see Fig. 7; [59]), as well as the usage of the OCL quantifiers <code>collect</code> (see [56,57]) or <code>closure</code> (see [57])
Definition 1.6: $successors : O \mapsto \mathcal{P}(O)$	Implicitly defined via our metamodel extension, the specification of UML activity models (see Fig. 7; [59]), and helper OCL operations (see, e.g., <code>allSuccessors</code> in Constraints 4a and 4b in Appendix A)
Definition 1.7: $predecessors : O \mapsto \mathcal{P}(O)$	Implicitly defined via our metamodel extension, the specification of UML activity models (see Fig. 7; [59]), and helper OCL operations (see, e.g., <code>allPredecessors</code> in OCL Constraints 6a and 6b in Appendix A)
Definition 1.8: $ca : O_{Sec} \mapsto S_C$	Metamodel extension <code>SecureNode</code> and corresponding sub-types (see Fig. 7)
Definition 1.9: $ia : O_{Sec} \mapsto S_I$	Metamodel extension <code>SecureNode</code> and corresponding sub-types (see Fig. 7)
Definition 2.1: $\forall o_s \in O_{Sec} : ca(o_s) \cup ia(o_s) \neq \emptyset$	OCL Constraints 1, 2, and 3 in Appendix A
Definition 2.2: $\forall o_s \in O_{Sec} : \forall o_t \in successors(o_s) : o_t \in O_{Sec}$	OCL Constraints 4a and 4b in Appendix A
Definition 2.3: $\forall o_s \in O_{Sec} : \forall o_t \in successors(o_s) : ca(o_s) = ca(o_t)$	OCL Constraint 5 in Appendix A
Definition 2.4: $\forall o_s \in O_{Sec} : \forall o_t \in successors(o_s) : ia(o_s) = ia(o_t)$	OCL Constraint 5 in Appendix A
Definition 2.5: $\forall o_t \in O_{Sec} : \forall o_s \in predecessors(o_t) : o_s \in O_{Sec} \Rightarrow ca(o_t) = ca(o_s)$	OCL Constraints 6a and 6b in Appendix A
Definition 2.6: $\forall o_t \in O_{Sec} : \forall o_s \in predecessors(o_t) : o_s \in O_{Sec} \Rightarrow ia(o_t) = ia(o_s)$	OCL Constraints 6a and 6b in Appendix A

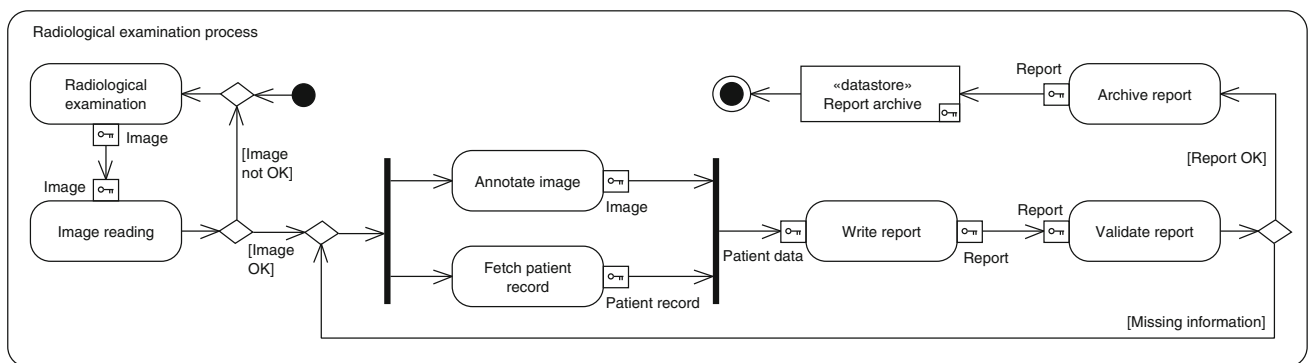


Fig. 8 Radiological examination process with secure object flows

record, respectively. Note that *Image* and *Patient record* are specialized classifiers of type *Patient data* (see Fig. 9) and, therefore, support the same *SecureObjectFlows* attributes as defined in Table 3.

After the report has been written, it is validated by a senior physician. If the report is incomplete, the corresponding actions have to be repeated. Otherwise, the report is archived via a *SecureDataStoreNode* (see Fig. 8).

Table 3 SecureObjectFlows attributes for the radiological examination process

Object type	SecureObjectFlows attributes
<i>Patient data</i>	confidentialityAlgorithm = Aes256; integrityAlgorithm = Sha512
<i>Report</i>	confidentialityAlgorithm = Aes256; integrityAlgorithm = Sha512

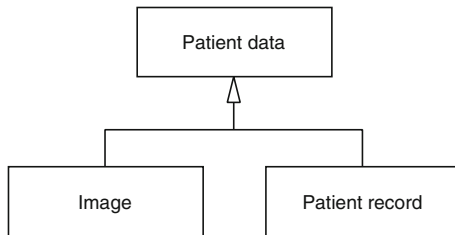


Fig. 9 Patient data object types

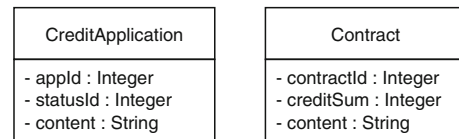


Fig. 11 Data items of the credit application process

4.1.2 Credit application process

Figure 10 shows a credit application process that uses the elements of the SecureObjectFlows package. The model contains swimlanes representing a customer and a bank clerk. Figure 11 shows a UML class diagram that describes the data items used in the credit application process. In addition, Table 4 documents the attribute-value pairs of the corresponding secure object nodes. The activity starts when the SecureActivityParameterNode named *Credit application* passes an object token to the *Check application form* action (see Fig. 10). In this example, the *Credit application* SecureActivityParameterNode is ensuring the data confidentiality and the data integrity of the corresponding object tokens via the AES-192 and SHA-1 algorithms, respectively (see Table 4). Remember that the formal semantics of the respective modeling elements are defined via the OCL constraints from Appendix A.

After completing the *Check application form* action, the creditworthiness of the applicant is checked. If the check fails, the credit application is rejected and the process ends (see Fig. 10). If the creditworthiness check is passed, however, the bank offers a contract to the respective customer. If the credit sum does not exceed the amount of 5000, the applicant is offered a standard contract. Otherwise, a customized contract is negotiated with the client. Because the contents of this contract are confidential, both output pins of the *Standard contract* and *Negotiate contract* actions as well as the input pin of the subsequent action *Approve contract* support confidentiality properties (see also Table 4). In Sects. 5 and 6, we use the credit application example to describe the modeling of (secure) process-driven SOAs and to illustrate our tool support for secure object flows.

5 Modeling process-driven SOAs with UML

In the context of (Web) service modeling, identifying and categorizing services that are based on business process artifacts is an important modeling task. It provides the input for

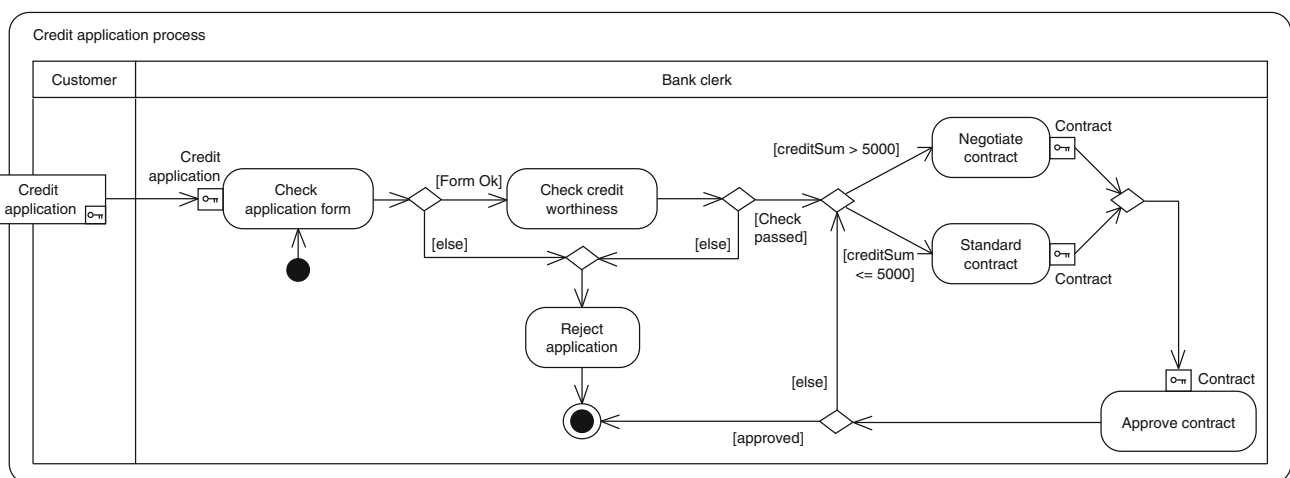


Fig. 10 Credit application process with secure object flows

Table 4 SecureObjectFlows attributes for the credit application process

Object type	SecureObjectFlows attributes
<i>Credit application</i>	confidentialityAlgorithm = Aes192; integrityAlgorithm = Sha1
<i>Contract</i>	confidentialityAlgorithm = Aes192

specifying the services, the service architecture, and the technical (i.e., executable) process descriptions. In the remainder of this paper, we focus on the specification of services and a corresponding service architecture using extended UML activities (see Sect. 4) and other UML models. Below, we describe how we derive the structural specification of a SOA (in terms of a distributed system architecture) from a business process modeled as a UML activity model. The examples given below refer to the credit application process from Fig. 10. For the sake of simplicity, we make the following assumptions:⁴

- Business units and actors involved in a process are modeled as `ActivityPartitions` (swimlanes). Thus, each swimlane indicates task ownership by a single unit or by an actor. In the service modeling step, units and actors may be modeled as SOA participants that provide and consume services. For example, the credit application activity from Fig. 10 contains two swimlanes (and thereby two task owners): The `BankClerk` receives and evaluates the credit application filed by the `Customer`.
- A process model identifies the object flows between tasks. This means that the model specifies the data items (objects) that serve as the input for and as the output of the tasks. For example, the credit application process describes object flows for a credit application and for the respective contract (see Figs. 10, 11). The corresponding data items (objects) enter a service model as the invocation data that are exchanged between the services (e.g., via messages, see [81]).
- A process model may define object flows between tasks owned by a single actor as well as object flows which occur between tasks owned by two (or more) distinct actors. Thus, an object flow within the same swimlane is executed by a single actor (i.e., it does not involve interaction between actors). In a service architecture, such an actor becomes a SOA participant which is responsible for executing a macroflow or a microflow (see also [25]). On the other hand, object flows crossing swimlane boundaries identify interactions between services that are provided and consumed by two (or more) actors (SOA participants). Such interactions effectively turn into

service invocations, and the object flow details (such as object types and security attributes) are contracted via respective service interfaces.

- The data integrity and data confidentiality properties expressed via our `SecureObjectFlows` extension apply to both intra-swimlane and inter-swimlane object flows (of course). In the subsequent step of modeling a service architecture, the secure object flows therefore map to either or both the macroflow/microflow specifications and the service interfaces.

A service specification includes the definition of structural and behavioral views of a service architecture. In particular, we use the Service-oriented architecture Modeling Language (SoaML, [58]) and the SoaML extension UML4SOA (see [38]). The SoaML provides essential modeling primitives for structural views of a service architecture (including participants, collaborations, service contracts and interfaces, as well as messages). The UML4SOA extension is used for modeling macroflow/microflow specifications for the participants of a service architecture. Moreover, we describe how we integrated the `SecureObjectFlows` extension (see Sect. 4) with the SoaML and UML4SOA, respectively. Thereby, we provide a seamless mapping of integrity and confidentiality properties specified at the business process level to the structural and behavioral views of a service architecture.

5.1 Modeling the structure of a process-driven SOA

The SoaML offers an extension of the UML composite structure metamodel: Fig. 12 shows an excerpt from the SoaML metamodel extension. This extension enables a composition of service consumers and providers via a set of interacting entities, referred to as `Participants`. A participant announces its interaction capabilities and requirements through `Service` and `Request` ports, respectively (see Fig. 12). Because the `Service` and `Request` elements are derived from the UML `Port` metaclass, they specify required and provided `Interfaces`.

`Interfaces` can directly be attached to `Service` and `Request` ports or they can be specified for a port via an intermediary construct: a so called `ServiceInterface` (see also [14]). A SoaML `ServiceInterface` is derived from the `Class` metaclass (see Fig. 12) and can be used to define a port protocol. `Participants` are connected via ports. The context of such a connection is modeled

⁴ Note, however, that we only make these assumptions to simplify the following explanations, our approach is independent of these assumptions, of course.

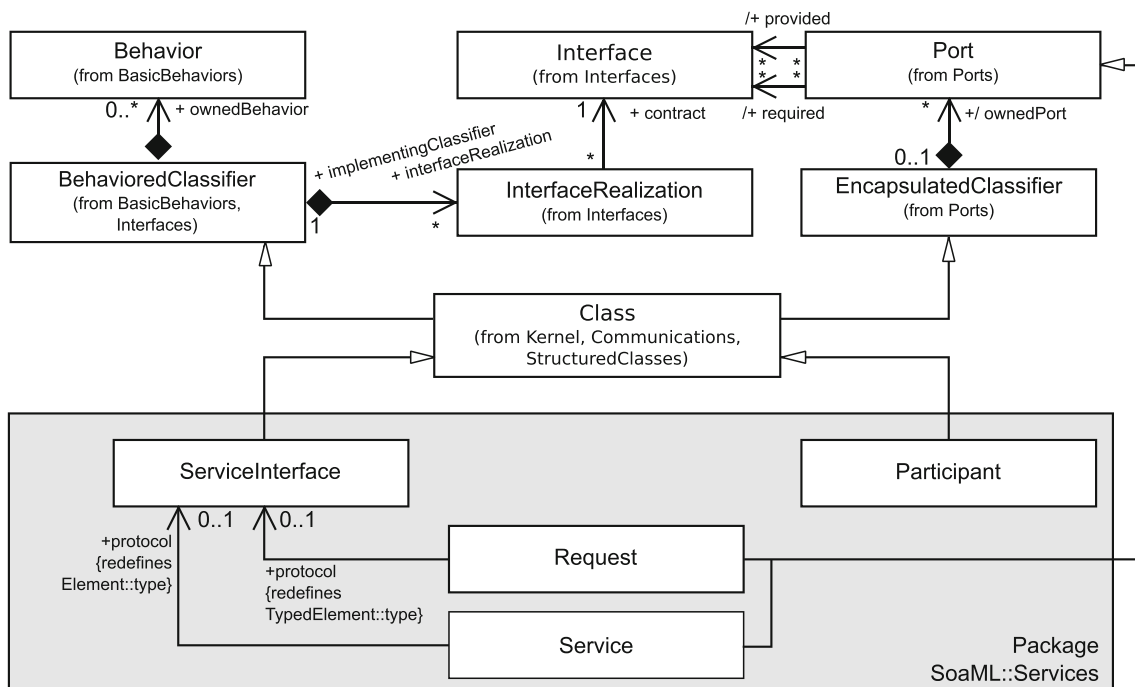


Fig. 12 Excerpt from the SoaML metamodel extension (see [58])

Table 5 Selected SoaML modeling elements

SoaML::Services metaclass	SoaML stereotype	Specialized/Extended metaclass	Description
Participant	«Participant»	Class	Represents a software system, component, or application which provides or consumes services (including process engines)
Request	«Request»	Port	Defines the service interaction point of a participant for consuming services offered by other participants, according to the port's required interfaces
Service	«Service»	Port	Defines the service interaction point of a participant for providing services to other participants, according to the port's provided interfaces
ServiceInterface	«ServiceInterface»	Class	Defines the structural (e.g., the operational interface) and behavioral properties (e.g., the invocation protocol) of a service. It is shared between a pair of request and service ports

as a *ServiceChannel*. A SoaML *ServiceChannel* ensures protocol compliance between a pair of *Service* and *Request* ports. In the structural view, protocol compliance is either expressed by sharing a *ServiceInterface* between corresponding *Request* and *Service* ports, or by directly connecting their required and provided *Interfaces*. Table 5 gives an overview of selected SoaML elements for the definition of composite structures (see also Fig. 12).

Figure 13 shows an example of a composite SOA structure modeled using the SoaML. The example includes two *Participants* (A and B) which define interacting subsystems of the SOA. Participant B acts as the service provider. This is modeled via its «*Service*» port. Participant A is

a service consumer modeled via a «*Request*» port. The requestor and consumer ports are connected through a service channel. The structural dependency realized by the two ports via mutually provided and required operations is specified by the service interface *AService* (see Fig. 13). This service interface describes two participating roles (*roleA* and *roleB*) with each role referring to a corresponding *Interface*. In the context of the given service channel, the requestor port binds *roleA* while the provider port binds *roleB*. This binding indicates that the «*Service*» port implements the provided interfaces (i.e., *InterfaceB*) and uses the required interfaces (i.e., *InterfaceA*; see Fig. 13). The «*Requestor*» port of participant A is also typed by the *AService* interface and

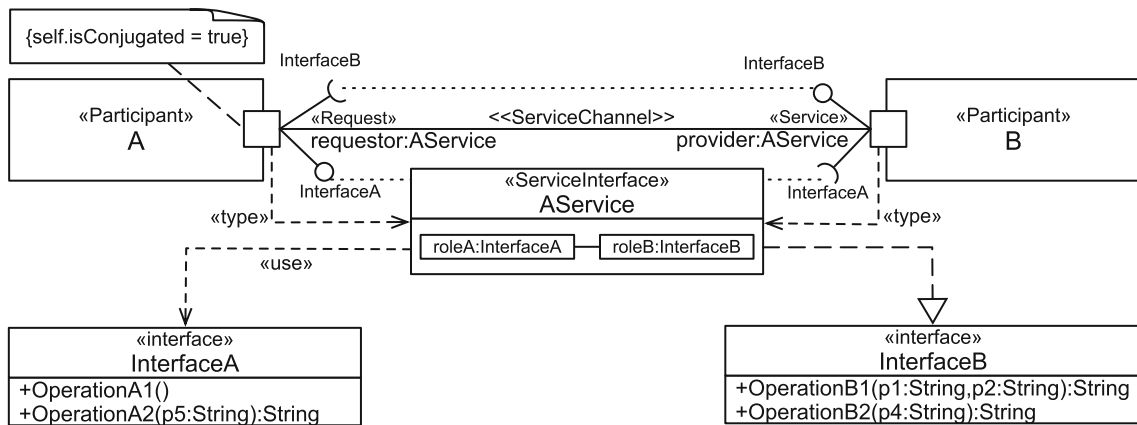


Fig. 13 Example of a composite SOA structure in the SoaML

because it is defined as a *conjugated* port, the meaning of the required and provided interfaces linked to the port is inverted (see Fig. 13). The *requestor* port provides an interface realization for InterfaceA and expects InterfaceB to be implemented by its port *provider* (for further details see [58]).

5.2 Modeling the behavior of a process-driven SOA

After defining a SoaML structure model, we require means to express the object flows resulting from service invocations (as well as their integrity and confidentiality properties). In general, we can distinguish two types of object flows in a process-driven SOA. First, object flows occurring during execution of macroflows/microflows (see [25]). These object flows are internal to a process engine and will be referred to as *process execution data* in the following. Second, object flows resulting from service invocations. Note, however, that neither business nor control data are exclusive to one of these object flow categories. They may rather be involved in both types of object flows. Consider, for example, that the credit application data from Fig. 10 are first reified as a data structure that is associated with a certain process instance and stored by the respective process engine. In a subsequent step, the data are marshaled into a message which is then delivered to a remote service endpoint. Because of this dual character of business and control data in a process-driven SOA, we require two different, yet complementary, behavioral viewpoints to specify secure object flows for these data assets. In particular, it is necessary to incorporate secure object flows in *service orchestration* specifications as well as *service choreography* specifications.

The SoaML provides explicit extension points for attaching behavioral specifications to elements of a composite SOA structure. As far as participants and service interfaces are concerned, the SoaML recommends the use of UML activi-

ties and interactions (see, e.g., [13, 58]). However, the SoaML does not provide any normative guidance for specifying SOA behaviors such as service choreography and service orchestration (see, e.g., [19]).

5.2.1 Specifying a choreography via UML activities

Object flows that realize service invocations across ports are constrained by the ports' protocol. This also applies for object flows between corresponding *Service* and *Request* ports that are provided through the same *Participant*. The behavioral part of a port protocol stipulates the choreography of service invocations between the *Interfaces* and it defines the respective service invocation patterns (such as "fire and forget" or "result callbacks"). While the structural part of a port protocol is specified by a *ServiceInterface*, the *ServiceInterface* can be extended via an owned *Behavior* to express the details of a corresponding behavioral protocol. To represent secure object flows at the level of such a choreography specification, we use an extended UML activity. For example, the use of UML activities allows us to integrate the protocol viewpoint with the orchestration viewpoint in terms of *ServiceActivityNodes* (see Fig. 14).

In particular, we use UML activities as owned behavior of *ServiceInterfaces* to model flows of invocation objects. This choice permits us to model the characteristics of service invocation data (see, e.g., [81]) through UML object flows:

- *Invocation data as object nodes*: An *Activity* can model input and output parameter streams of service invocations. This is a prerequisite for applying the *SecureObjectFlows* extension to invocation data at this level.

Fig. 14 Activities as behavioral specifiers

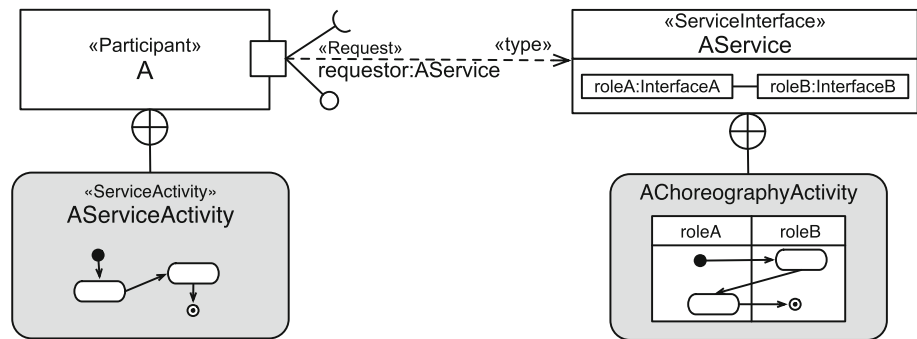
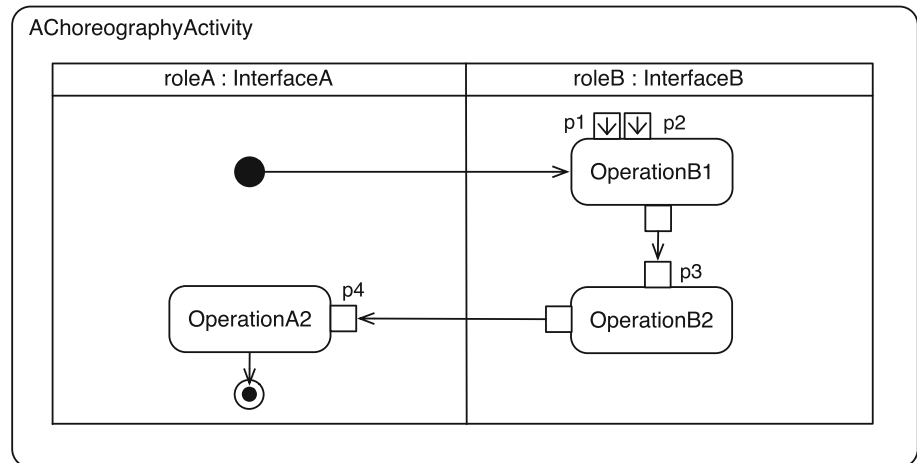


Fig. 15 Example of a choreography activity



- *Choreography roles:* ActivityPartitions can be used for modeling *choreography roles*.⁵ Thus, each ActivityPartition represents an “interface-realizing role” and thereby abstracts from the Participants that use or implement the interfaces. In the compositional view, they correspond to the respective ServiceInterfaces (see roleA and roleB in Figs. 13 and 14). Note that ServiceInterfaces may refer to more than two parts (or choreography roles) and can be used to model multi-directional invocation flows.
- *Duality of invocations:* Choreography roles are typed through the Interfaces that are required and implemented by the corresponding ServiceInterface (for example InterfaceA and InterfaceB in Fig. 13). ActivityPartitions model the providers and consumers of invocations. Thus, the SecureObjectFlows elements in an Activity model consumer-side and provider-side security properties (such as signature mechanisms for messages).
- *Standalone choreography specification:* An Activity that is owned by a ServiceInterface and the security properties that are specified for its object flows are modeled independently of the Participants which consume or implement the respective service endpoints. For example, in Fig. 13 ASerivce applies to any pair of Participants A and B, regardless of whether they act as process engine or service providers.

Figure 15 shows an example choreography activity, where AChoreographyActivity further specifies the ASerivce interface from Fig. 14. Through its requestor port, participant A consumes the ASerivce interface (represented via roleA, see Figs. 13, 14). Thus, the activity defines the operation calls between the Interfaces via actions and a corresponding control flow. Object flows model the flow of input and output parameters between the operation calls (see parameters p1–p4 in Fig. 15).

SecureObjectFlows are independent of different types of invocation patterns. Therefore, we do not further elaborate on the definition of invocation patterns for distributed systems in general (see, e.g., [81]), or for process-driven SOAs (see, e.g., [97]). Note, however, that UML activities can be used to model service invocation patterns such as fire-and-forget invocations, sync-with-server invocations, request-

⁵ Note that such “choreography roles” do only model which participant provides and/or requests specific functions/interfaces. They do not model access control roles. For the definition of process-related access control models, the SecureObjectFlows extension is integrated with the extension presented in [85].

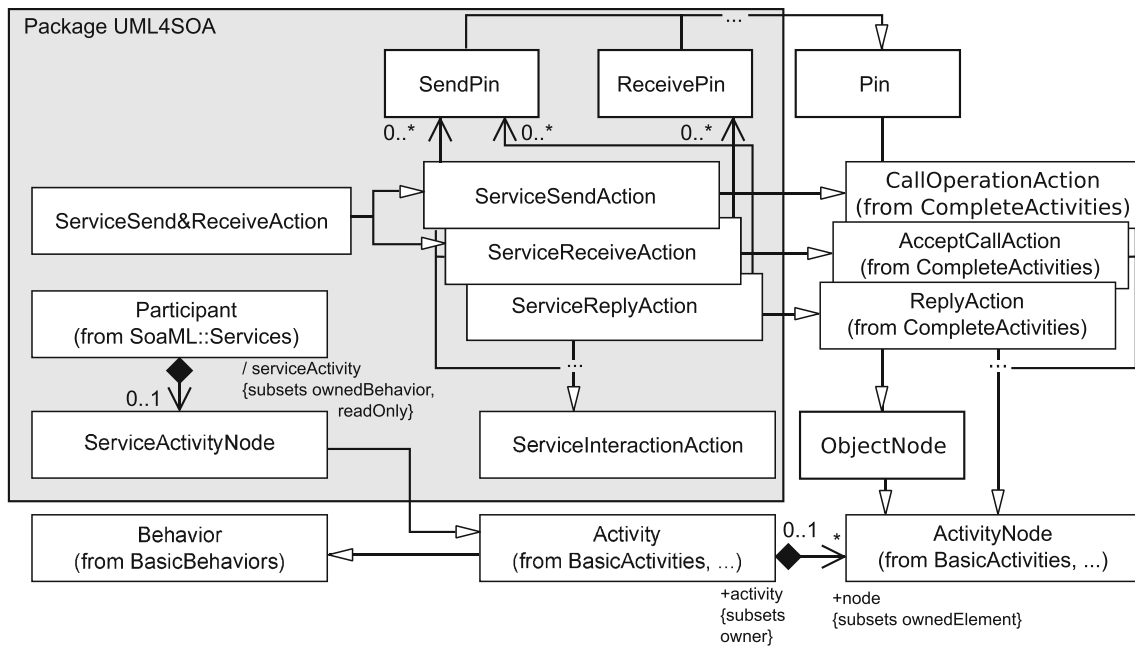


Fig. 16 Excerpt from the UML4SOA metamodel (see [38])

reply invocations, or result callbacks, of course (see, e.g., [97]).

5.2.2 Specifying service orchestrations via UML activities

We use the UML4SOA [38] to model object flows as an integral part of service orchestrations. UML4SOA is a SoaML extension to model process-driven service compositions through orchestration specifications that are defined with UML activities. In a process-driven SOA, one or more Participants act as process engines that invoke the functions of service providers to execute tasks. The corresponding tasks are defined via a composite activity. In UML4SOA, such a composite activity (controlled by a single, orchestrating Participant) is modeled through a *ServiceActivityNode* (also referred to as a “service activity”). A *ServiceActivityNode* is owned by the orchestrating Participant (see Fig. 16 and the example from Fig. 14).

In particular, a service activity defines the control flow through *ServiceInteractionActions* and corresponding UML protocol state machines. For example, a fire-and-forget invocation is modeled using a *ServiceSendAction*. Accessing and changing the process engine’s state can be modeled via *DataHandlingActions*. The internal data flow and data dependencies of a service activity (e.g., the process engine state or the invocation data) are expressed using a set of refined object nodes (*SendPins* and *ReceivePins*) that are linked

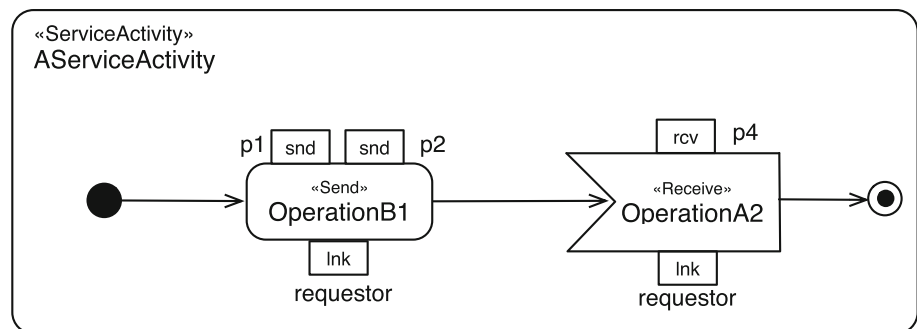
to *ServiceInteractionActions*. These *SendPins* and *ReceivePins* represent the object nodes which form object flows (see Fig. 16). Table 6 provides an overview of the UML4SOA model elements that are relevant for the remainder of this paper.

Figure 17 shows an example of a *ServiceActivityNode*. The orchestration specification *AServiceActivity* (see also Fig. 14) is modeled via a *ServiceActivityNode* that is registered as an owned behavior of the corresponding Participant (participant A from Fig. 14). Process execution starts by sending a call request to an operation *OperationB1* via the consumer port (*requestor*) of participant A. The process instance must provide two input parameters *p1* and *p2* as *SendPins* to the call request (see Fig. 17). The call request does not return out parameters to the process instance (i.e., it models a fire-and-forget invocation). The participant then enters a waiting state, until an inbound call request for the operation *OperationA2* is signalled through the *requestor* port. This inbound call request provides invocation data (*p4*) via a *ReceivePin* (see Fig. 17).

The examples from Figs. 15 and 17 show the complementary nature of the invocation activities and the service activities. Service invocations that are issued or expected by the orchestration specification (see Fig. 17) are reflected in the corresponding choreography specification (see Fig. 15) as the required or the provided operations. While the orchestration specification only considers consuming or providing roles (i.e., the required or implemented Interfaces) of a *single* participant, the choreography specification integrates

Table 6 Selected UML4SOA/SoaML modeling elements

SoaML::Services metaclass	SoaML/UML4SOA stereotype	Specialized/Extended metaclass	Description
ServiceActivityNode	«ServiceActivity»	Activity	A service orchestration specification; specific to a single SoaML participant
SendPin	«Snd»	Pin	Represents outbound invocation data provided to a service in a send action
ReceivePin	«Rcv»	Pin	Represents inbound invocation data received from a service in a receive action
ServiceSendAction	«Send»	CallOperationAction	Models a non-blocking service invocation
ServiceReceiveAction	«Receive»	AcceptCallAction	Models a blocking message listener
ServiceReplyAction	«Reply»	ReplyAction	Represents an invocation which completes a receive action; i.e., the provider-side of a result-callback invocation
ServiceSend&ReceiveAction	«Send&Receive»	ServiceSendAction, ServiceReceiveAction	Models a blocking request-reply invocation
LinkPin	«Lnk»	Pin	A reference to a service endpoint (i.e., a service or a request port)

Fig. 17 Example of a service activity

the consuming and providing roles of two or more participants for the scope of a single service channel.

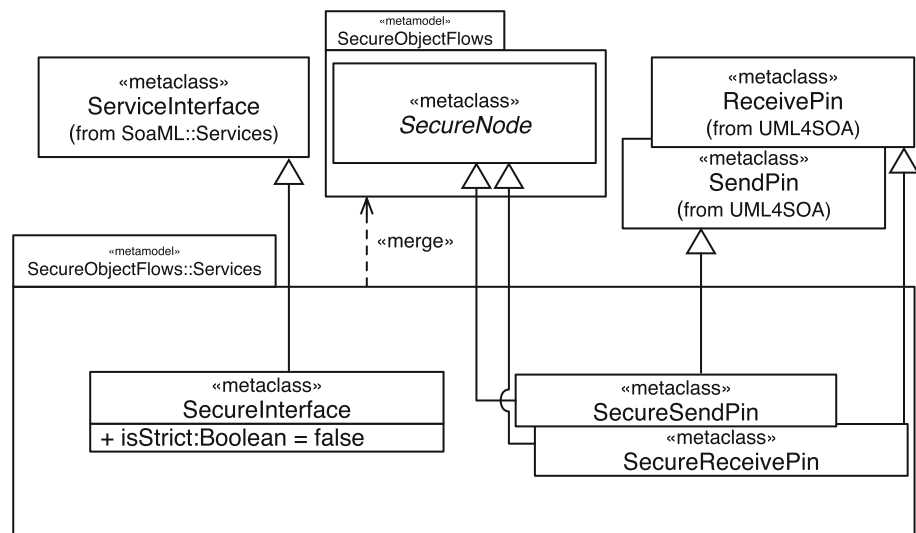
5.3 A SoaML extension for secure object flows

Security concerns such as message confidentiality and message integrity are crosscutting in nature and must be addressed in different types of SOA models (see, e.g., [25, 81]). Business process data (e.g., business and control objects) are exchanged in terms of invocation data. Invocation data include service endpoint references, operation names, input and output parameters, as well as exception data (see, e.g., [81]). At runtime, a process engine controls process instances that include corresponding data objects. Confidentiality and integrity properties of invocation data and process execution data affect data transformation steps at various layers of a SOA. As a result, the invocation processing infrastructure as well as the respective transport handling must be adapted. For example, if we need to ensure the integrity of data assets in a business process, a modeler must define message integrity constraints over the corresponding service interfaces. Afterwards, corresponding source code (such as message interceptors for message signing) and/or

configuration data (e.g. for a security component) can be generated. Therefore, multiple views must be considered to support the definition of secure object flows in a SOA context. In this context, the choreography specifications for service interfaces and the corresponding orchestration specifications are of special importance:

- *Choreography specifications for secure flows of invocation data:* This view includes modeling support for invocation data (such as input and output parameters) which require integrity and/or confidentiality properties. In the compositional view of a SoaML model, the structural characteristics (e.g., the interface signature) of service invocations are specified via *ServiceInterfaces* (see also Fig. 12).
- *Orchestration specifications for secure flows of process execution data:* This view includes modeling support for secure object flows of process execution data as well as invocation data. A UML4SOA *ServiceActivityNode* allows to model specifying flows of process execution and invocation data for a single entity, e.g., a *Participant* or a role in the sense of a *ServiceInterface* (see also Fig. 16).

Fig. 18 The SecureObjectFlows::Services package



Modeling integrity and confidentiality properties in the choreography specifications of *SecureInterfaces* and *ServiceActivityNodes* is complementary. If the *SecureInterface* view was not available, the modeling effort would have to be duplicated for the *ServiceActivityNodes* of all *Participants* which share one or more *ServiceInterfaces*. This is because a *ServiceActivityNode* captures service invocations as patterns of *ServiceInteractionActions* from either the consumer or provider side *only* (see also Sect. 5.2). Moreover, a *Participant* may represent a process engine as well as simple service providers. As a consequence, the *ServiceActivityNode* element may not be available for all *Participants*.

To integrate the *SecureObjectFlows* extension with the SoaML, we provide a UML integration package (see Fig. 18). Our *SecureObjectFlows::Services* package adds SoaML-specific constraints for secure object flows.

5.3.1 SecureObjectFlows::Services abstract syntax

The *SecureObjectFlows::Services* package introduces a specialized *ServiceInterface* called *SecureInterface*. At the SoaML metamodel level, *SecureInterface* extends the *ServiceInterface* metaclass (see Fig. 18; and OCL Constraint 7 in Appendix B). A *SecureInterface* contracts either a strict or a permissive mode. The *permissive mode* is the default mode (i.e., *isStrict* is set to *false*) which allows to include secure object flows as well as ordinary object flows. In contrast, the *strict mode* (i.e., *isStrict* is set to *true*) defines that all invocation data flows (as specified further below) must be secure object flows (see OCL Constraint 8).

To model secure object flows in UML4SOA *ServiceActivityNodes*, we provide two additional

metaclasses: *SecureSendPin* and *SecureReceivePin* (see Fig. 18). They integrate the capabilities of the *SecureNode* metaclass (see Sect. 4) and the *ReceivePin* and *SendPin*, respectively. For all metaclasses provided by the *SecureObjectFlows::Services* package, the notation for the *SecureNode* metaclass applies (see Sect. 4; [27]).

5.3.2 Constraints for the SecureObjectFlows::Services package

In this section, we discuss constraints for an *Activity* that is owned by a *SecureInterface* and for *ServiceActivityNodes* of the *Participants* which are connected by *SecureInterfaces*. The OCL constraints for the *SecureInterface* metaclass are defined over the metaclasses *SecurePin* and *SecureActivityParameterNode*. The OCL constraints for *ServiceActivityNodes* refer to the *SecureSendPin* and *SecureReceivePin* metaclasses (see Figs. 12, 16, 18).

Explicit links between invocations and interfaces: An *Activity* that specifies a choreography must only describe service invocations between *Operations* that are provided or required by the *Interfaces* referenced by the corresponding *SecureInterface* (see OCL Constraint 9). This allows the modeler to express explicit links between *Actions* in a choreography activity and the *Operation* repository represented by these *Interfaces*.

Cross-interface invocations only: *ActivityPartitions* represent “interface-realizing” and “interface-providing” roles with respect to a *SecureInterface* (see also Sect. 5.2). Object flows may occur *within* a single partition or *between* two partitions. Thus, an object flow between two *CallOperationActions*, which are modeled in different *ActivityPartitions*, depicts an

output/input dependency between operations which represent an actual service invocation (see Figs. 12, 16, 18). If two adjacent `CallOperationActions` reside within the same `ActivityPartition`, they are required and provided by the same `Interface`. Thus, secure object flows in the `SecureObjectFlows::Services` package apply to cross-interface invocations (see OCL Constraint 10), and each activity node is assigned to exactly one partition (see OCL Constraint 11).

Activity parameters for initial and intermediary inbound data: A choreography activity captures data dependencies between invocations, i.e., the output data of one invocation serve as the input data for a subsequent invocation. In two important cases, however, the input data may originate from the outside. These cases are *initial* and *intermediary* inbound data.

Initial inbound data are provided through the required `Interface` of the consumer role. These data are contracted by the operation of a `SecureInterface` that triggers the execution of an `Activity` (see Figs. 12, 16, 18). Intermediary inbound data are not the result of previous invocations within the same choreography. The input is rather provided from the outside, such as from a process engine holding process control data which are then used as the input parameters for an operation call.

For specifying secure object flows, however, it is mandatory to model pairs of secure object nodes (see Sect. 4; [27]). This is because the security properties required at either end of an object flow must be compatible (see also Sect. 3). Thus, each secure pin and each secure activity parameter node must be connected to (at least) one object flow (see OCL Constraint 12).

Activity parameters for intermediary and flow-final outbound data: Analogous to initial and intermediary inbound data, output data can describe external data dependencies, i.e., dependencies which do not manifest within the choreography activity alone. For instance, an invocation's output may be stored in a process-persistent variable by a process engine. In this context, each secure pin and each secure activity parameter node must be connected to (at least) one corresponding object flow (see OCL Constraint 12).

Streaming-only intermediary activity parameters: `ActivityParameterNodes` that are used to model intermediary inbound data, and output data represent *streaming* activity parameters (see OCL Constraint 13). Streaming parameters model data which become available in the context of an activity, or which leave this context during execution of the `Activity`. Note that the streaming mode is only mandatory for cases of secure intermediary `InputPins` and `OutputPins` (in the sense of OCL Constraint 12).

Same origin for input data flows: Input data for service invocations, which are represented by `InputPins` on `CallOperationActions`, must have related object nodes

which reside in the same `ActivityPartition`. Different partitions as the origins for input data of an operation are not valid (see OCL Constraint 14). This requirement follows from the intention to model input/output data dependency along a path of operation calls. The input parameters of an operation, i.e., the `InputPins` modeled for a `CallOperationAction`, must either be related to output parameters (`OutputPins`) of the preceding operation call, or to (initial or intermediary) inbound parameters of the choreography activity. In either case, these source object nodes must share their `ActivityPartition` origin.

Explicit links between orchestration and choreography activities: The overlap between the views provided by choreography activities for `SecureInterfaces` and `ServiceActivityNodes` becomes evident through the dual appearance of business and control data—once in terms of process execution data items and once as invocation data items. On the one hand, data returned from service invocations, e.g., out-parameter values, enter the process execution view as input data (e.g., for result callbacks). On the other hand, process execution data that are passed as parameter to invocation requests become in-parameters traveling across object flows in the choreography view. To avoid inconsistent models, it is important to verify these dependencies in the model specification phase. Therefore, OCL constraints 15 and 16 define consistency constraints between the two model views (see also Figs. 12, 16, 18).

5.4 UML profiles for secure object flows

Sections 4 and 5.3 introduced the `SecureObjectFlows` package and the `SecureObjectFlows::Services` package, respectively. Both packages specify UML metamodel extensions at the PIM level and provide native UML elements for the definition of secure object flows in general, and for secure object flows in SOAs in particular. An extension of the UML metamodel allows to define new and specifically tailored UML elements (defined via *new* metaclasses), and it allows to define a customized notation, syntax, and semantics for the new modeling elements. However, the integration of a metamodel extension with software tools most often results in a significant development effort. Therefore, a metamodel extension can be seen as a medium-term and long-term option to extend the UML (or another modeling language) as well as corresponding software tools.

In contrast, UML profiles provide a mechanism for the extension of *existing* UML metaclasses to adapt them for non-standard purposes. However, UML profiles are *not* a first-class extension mechanism (see [59, p. 660]) and are less powerful than metamodel extensions. In particular, UML profiles do not allow for modifying existing metamodels. Nevertheless, most UML tools directly support the definition of profiles. Therefore, it is comparatively easy to integrate

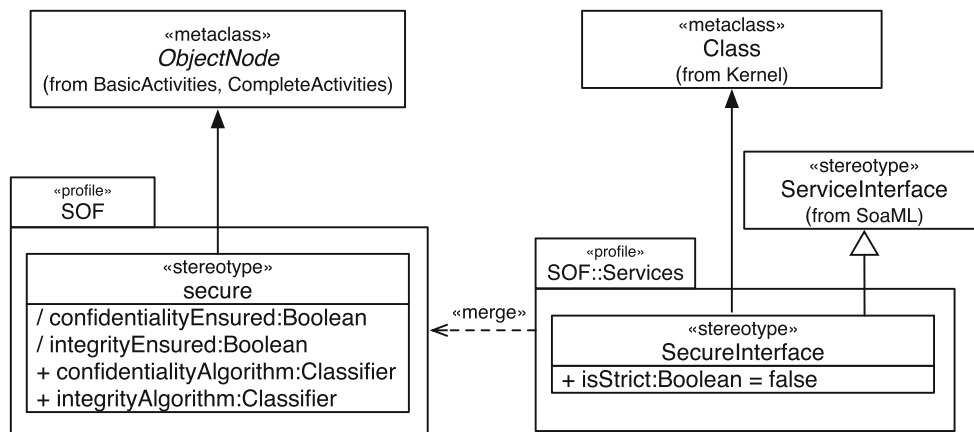


Fig. 19 The UML profile packages SOF and SOF::Services

Table 7 Mappings between the SOF::Services profile and the SecureObjectFlows::Services metamodel extension

M1 model (profile extension)		M1 model (metamodel extension)
<u>:Pin</u>	extension_secure → ← base_ObjectNode	<u>:SecurePin</u>
<u>:ActivityParameterNode</u>	extension_secure → ← base_ObjectNode	<u>:SecureActivityParameterNode</u>
<u>:SendPin</u>	extension_secure → ← base_ObjectNode	<u>:SecureSendPin</u>
<u>:ReceivePin</u>	extension_secure → ← base_ObjectNode	<u>:SecureReceivePin</u>
<u>:Class</u>	extension_SecureInterface → ← base_Class	<u>:SecureInterface</u>

UML profiles in a software tool. For this reason, we introduce two UML profiles for secure object flows. By these means, we provide both a long-term option based on metamodel extensions and a short-term option based on UML profiles.

Below, we describe the UML profile packages Secure Object Flows (SOF) and SOF::Services (see Fig. 19). In Sect. 6, we present our tool support for the definition of secure object flows in process-driven SOAs based on these two profile packages.

The SOF package provides a profile for a (simplified) variant of the SecureObjectFlows package (see Sect. 4), and the SOF::Services package provides a (simplified) variant of the SecureObjectFlows::Services package (see Sect. 5.3).

The secure stereotype provides the integrity and confidentiality attributes of the SecureNode metaclass (see Fig. 19). The OCL constraints for the SecureObjectFlows metamodel (see Appendix A) were adapted for the context of the secure stereotype. Table 7 shows mappings between the SOF::Services profile and the SecureObjectFlows::Services metamodel extension. In particular, the following transformation rules exist: Instances of Pin, ActivityParameterNode, SendPin, and ReceivePin tagged with the «secure» stereotype map to instances of SecurePin, SecureActivityParameterNode, SecureSendPin, and SecureReceivePin, respectively.

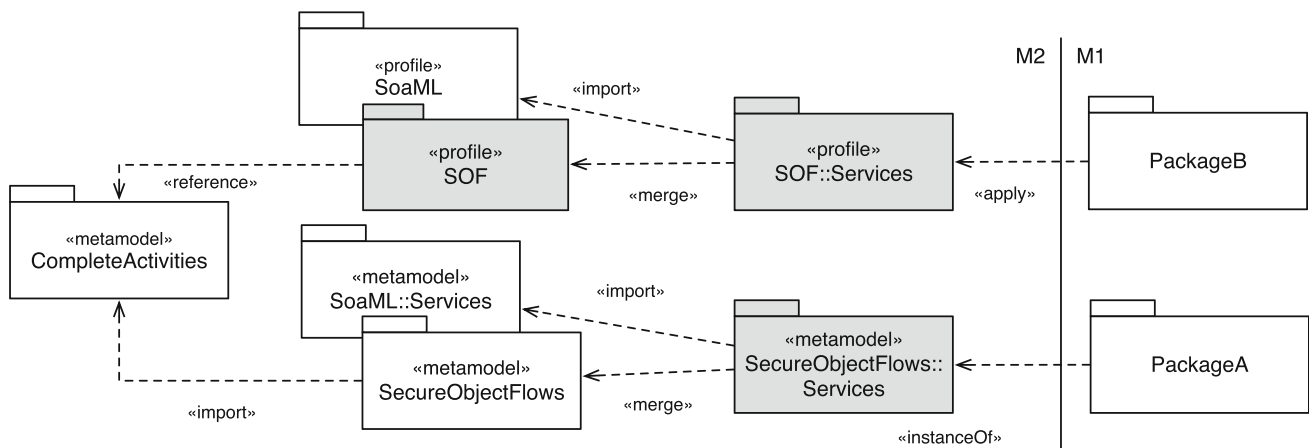


Fig. 20 The metamodel and profile packages for secure object flows

The `SecureInterface` metaclass is represented via the `SecureInterface` stereotype (see Fig. 19). In particular, this stereotype identifies a tagged `ServiceInterface` as a secure interface, and allows for specifying the strict or the permissive mode (see Sect. 5.3.1). The `SecureReceivePin` and `SecureSendPin` metaclasses are included in the SOF profile. However, an important limitation applies: `ServiceActivityNodes` may also contain `LinkPins` for identifying the ports of a given `ServiceInteractionAction`. Because they also represent `ObjectNodes` which are in principle extensible via the `«secure»` stereotype, we define that the `«secure»` stereotype must not be applied to object nodes tagged as `«lnk»` pins:

```

1 context SOF::Services::secure
2 inv: self.base_ObjectNode.getAppliedStereotype('UML4SOA::Services
   ::lnk') = null

```

The definition of secure object flows for the UML and their integration with the SoaML/UML4SOA via a metamodel extension as well as two profiles open up two integration paths (see Fig. 20). If we use the metamodel extension to define secure object flows, we essentially instantiate the corresponding metamodel (see “instanceOf” relation from `PackageA` to the `SecureObjectFlows::Services` package in Fig. 20). In contrast, if we use the profile extension to define secure object flows, we apply the profile to the corresponding UML model (see “apply” relation from `PackageB` to the `SecureObjectFlows::Services` package in Fig. 20). For details concerning the “instanceOf” and “apply” relations see [60].

5.5 An integrated example

Figure 21 shows an example that uses the SOF and SOF::Services profile packages—it extends the example from Sects. 5.1 and 5.2. Remember that in this exam-

ple, participant A acts as process engine and is specified via `AServiceActivity`. The service channel between the process engine and the service provider (participant B) is defined through the `AService` interface which owns `AChoreographyActivity` (see also Figs. 13, 14, 15, 17).

`AChoreographyActivity` is tagged with the `«SecureInterface»` stereotype, requesting the permissive mode (see Fig. 21). Thus, not all invocation object flows in the choreography activity need to be specified as secure object flows (see also Sect. 5.3.1). The choreography activity defines the object flow pointing towards `OperationB1` realizes a secure object flow that uses the `Aes192` as confidentiality mechanism. Moreover, the choreography activity specifies another secure object flow between `OperationB2` and `OperationA2`. This invocation object flow establishes end-to-end message integrity via the `Sha256` integrity algorithm.

`AServiceActivity` defines the orchestration specification of participant A (see Fig. 21). It includes the corresponding secure object flows from the perspective of the process engine (participant A). The two `SendPins` from the orchestration specification map to the input parameters `p1` and `p2` defined on `OperationB1` in the choreography activity. Moreover, the input parameter `p4` from the choreography activity matches the respective `ReceivePin` on the `ServiceReceiveAction` in the orchestration specification.

6 Tool support for secure object flows in SOAs

Our tool support for the definition of secure object flows in process-driven SOAs is based on the Eclipse 3.6 Model Development Tools (MDT; [11]) and the Eclipse Papyrus visual UML editor [12]. Moreover, we use the SoaML and

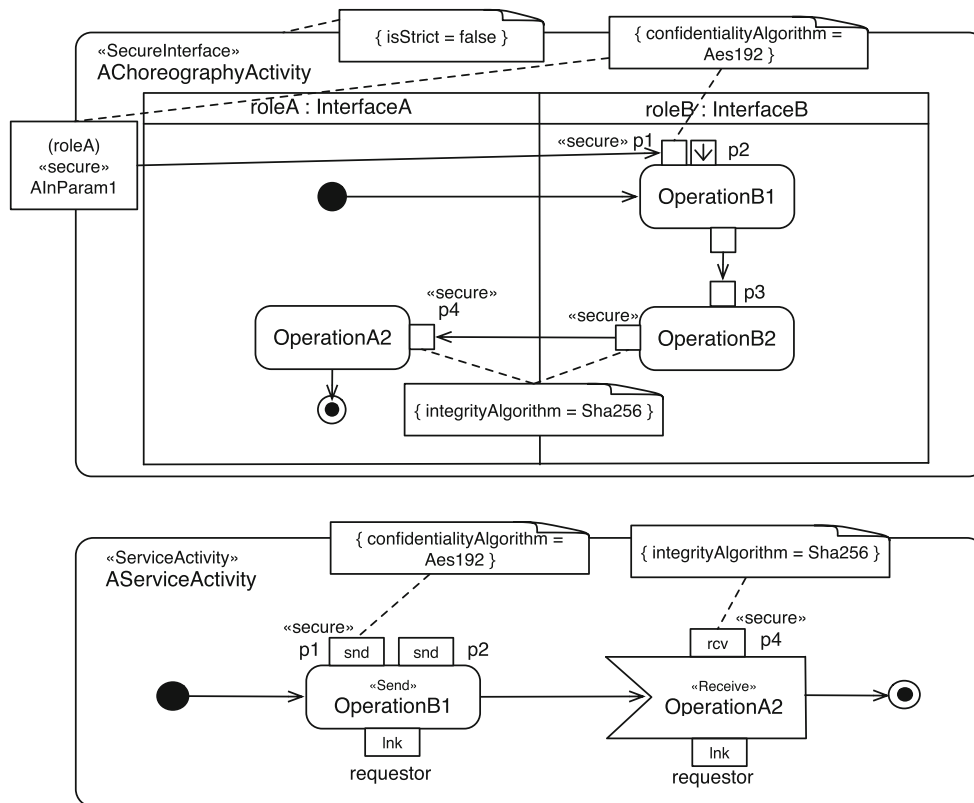


Fig. 21 Integrated views on secure object flows: orchestration and choreography specifications

UML4SOA profile definitions for MagicDraw 17.0 [51] to define service specifications. Model integrity checking based on the OCL constraints for our UML packages (see Appendices A, B) is performed in the Eclipse MDT environment.⁶

Our tool support enables automated model transformations for secure object flows that are defined via platform-independent models (PIMs). The model transformations produce corresponding platform-specific models (PSMs). The generated PSM artifacts include WSDL interface descriptions [92] and WS-BPEL process descriptions [61]. A major challenge of this model transformation step was to bridge the gap between the graph-based PIMs (defined via extended UML activities; see Sect. 5) and the block-based PSMs (defined via BPEL specifications; see, e.g., [41]). In particular, we extended the MDD4SOA Eclipse plugin [36] to support the corresponding model transformations for secure object flows. Additional transformation steps add WS-SecurityPolicy statements [63] to the generated interface descriptions and deployment descriptors. Moreover, our approach allows to add security properties to invocation data

(e.g., single parameters or message elements). These security properties are transformed into WS-SecurityPolicy descriptions (e.g., EncryptedElements, SignedParts). Modeling elements that specify integrity and confidentiality requirements of object flows map to identifiers for algorithm suites (as defined by the WS-SecurityPolicy specification [63]).

In the following, we use the credit application example from Sect. 4.1 to describe our extension for the Eclipse MDT tool chain in detail. Figure 22 gives an overview of the different steps and the resulting artifacts.

For the sake of simplicity, and in order to emphasize the details relevant for this paper, we focus on the object flow which triggers the processing of a credit application (see Sect. 4.1.2). The corresponding object flow is also depicted in the topmost diagram of Fig. 22 (PIM, Business Level, Business Activity). This object flow is defined as a secure object flow and models the submission of a credit application document, that is, it specifies integrity and confidentiality properties for the transferred document.

6.1 Modeling the SOA structure

The credit application is submitted by a customer software component and received by a bank clerk software component

⁶ All modeling and implementation artifacts are available from <http://nm.wu.ac.at/modsec>.

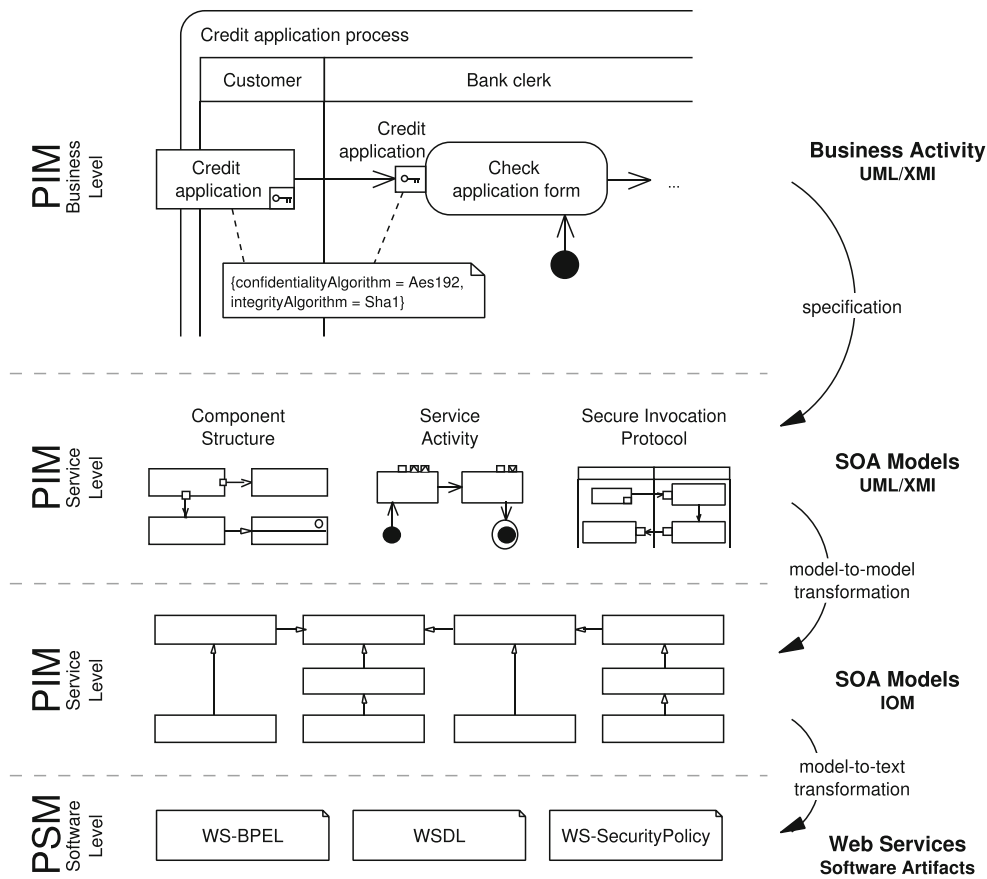


Fig. 22 Different modeling levels supported by the tool chain

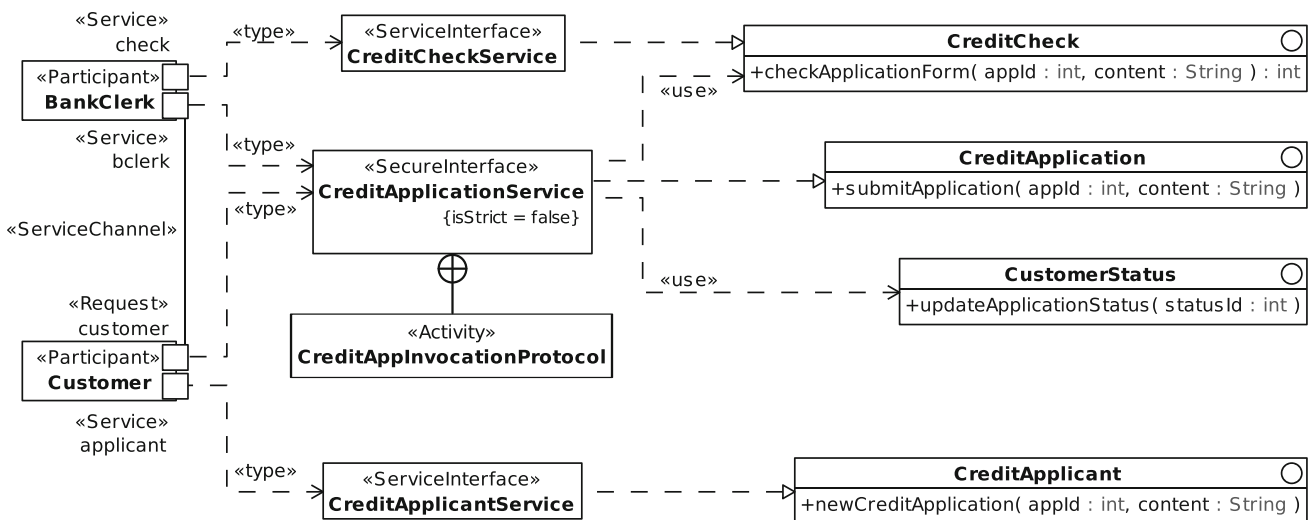


Fig. 23 Static structure of the credit application SOA

(see Fig. 22). In the structural viewpoint of a SOA, this translates into two interacting participants named BankClerk and Customer. A corresponding SoaML composite structure model is shown in Fig. 23.

The Customer and BankClerk participants are connected through a ServiceChannel which carries credit application submissions. The channel connects the two participants through their Request and Service ports.

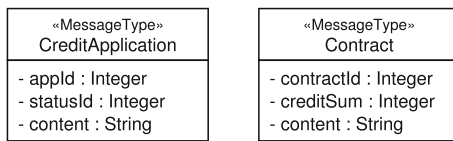


Fig. 24 Business objects as SoaML message types

The customer request port represents the Customer's consuming interaction point. The `bclerk` service port represents the corresponding providing interaction point of the `BankClerk` participant (see Fig. 23).

The details of service invocations for submitting credit applications are negotiated by the ports' protocol, i.e., via an instance of a `SecureInterface` named `CreditApplicationService`. This `SecureInterface` defines the required and provided interfaces for service invocations traveling through the customer and `bclerk` ports. The structural part of a respective protocol identifies a single operation `submitApplication` that is defined via the `CreditApplication` interface. This operation realizes the application submission and must be implemented by the `BankClerk` participant via its `bclerk` port (see Fig. 23).

While this section highlights the specification of one secure object flow only, the composite structure model from Fig. 23 includes additional `ServiceInterfaces` that result from other tasks of the credit application process (see Fig. 10). For example, the `BankClerk` participant provides the `CreditCheckService` interface via its `check` service port.

The `CreditApplicationInterface` is defined as a `SecureInterface`. Corresponding confidentiality and integrity properties are specified via a UML activity attached to the `SecureInterface`. The «`SecureInterface`» slot value for `isStrict` is set to `false` (see Fig. 23). Therefore, the `SecureInterface` is said to be *permissive* (see also Sect. 5.3.1). This is necessary because service invocations other than the secured application submission are specified via the same service interface (e.g., the `updateApplicationStatus` function provided through the `CustomerStatus` interface, see Fig. 23).

In addition, we must define a structural model of the business objects via SoaML message type specifications. In the credit application example, we have two types of business objects (the `CreditApplication` and the `Contract`, see also Sect. 4.1.2). Figure 24 shows the respective SoaML message type specification for these object types. Note that in the subsequent steps of our example, we do not use a document-centric service invocation style, but rather a procedural invocation style (also referred to as RPC-style). As a result, message type features (such as `appId` or `statusId`)

are included in the operation signatures rather than in the messages.

6.2 Modeling the SOA behavior

After defining the structural model, the control flow and the object flow for processing credit applications are added to the SoaML model. In Sect. 6.2.1, we specify the choreography of the `CreditApplicationService` interface. Subsequently, Sect. 6.2.2 defines the corresponding orchestration specification.

6.2.1 Choreography specification

`ServiceInterface` instances can include behavioral protocol specifications for the corresponding ports (see also Sect. 5). A behavioral protocol specification defined through a UML activity can describe various characteristics of a service channel. It identifies the `Interfaces` (grouped by a `ServiceInterface`) which depend on each other, the order and the targets of consecutive operation calls, as well as the respective invocation patterns (i.e., the choreography). In addition, input and output dependencies between the operation calls can be specified via object flows.

For the `CreditApplicationService`, we must define a choreography over four `Interfaces` and their operations: `CreditCheck`, `CreditApplication`, `CustomerStatus`, and `CreditApplicant` (see Fig. 23). Figure 25 shows a UML activity that models the corresponding choreography. The choreography identifies five data-level dependencies in terms of object flows between the call operation actions. When invoking the `newCreditApplication` operation, the `appId` and `content` parameters must be provided. Furthermore, the output parameters of the `newCreditApplication` (`content` and `appId`) are expected as input parameters for the subsequent `submitApplication` call. The fifth dependency specifies that the output parameter of the `checkApplicationForm` (`statusId`) is expected as the input parameter for the `updateApplicationStatus` action.

To model secure object flows, we tag the corresponding input and output pins of `newCreditApplication` and of `submitApplication` (see Fig. 25). The other object flows in the example are ordinary object flows. According to the constraints for the `SecureObjectFlows` package (see Appendix A), any secure target node must support the same security properties as its source node(s) (see OCL Constraint 5). Therefore, both secure object flows from Fig. 25 (and all four secure object nodes) support the same confidentiality and integrity algorithms (in this example AES-192 and SHA-1).

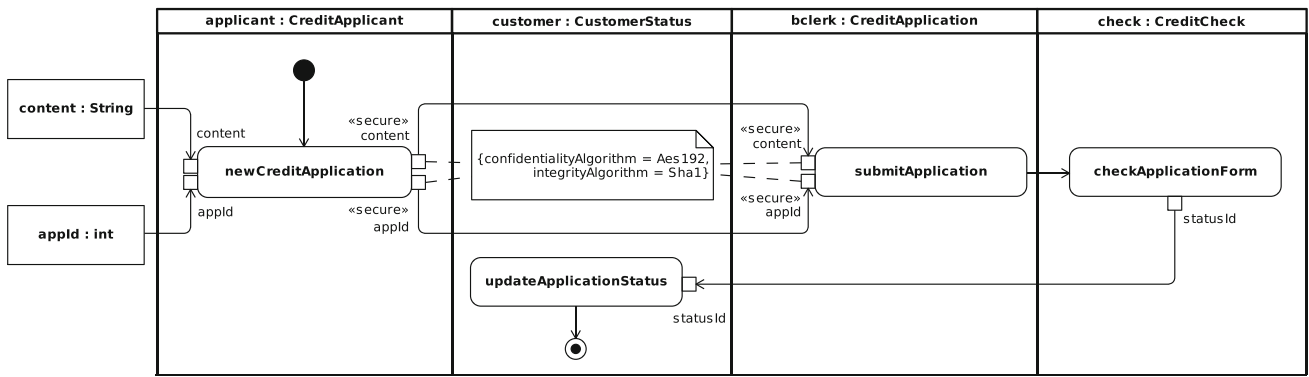


Fig. 25 Secure choreography activity

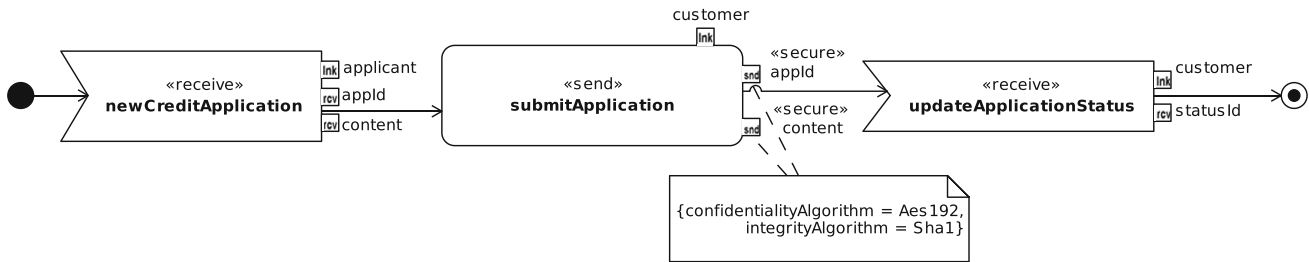


Fig. 26 Consumer (customer) service activity

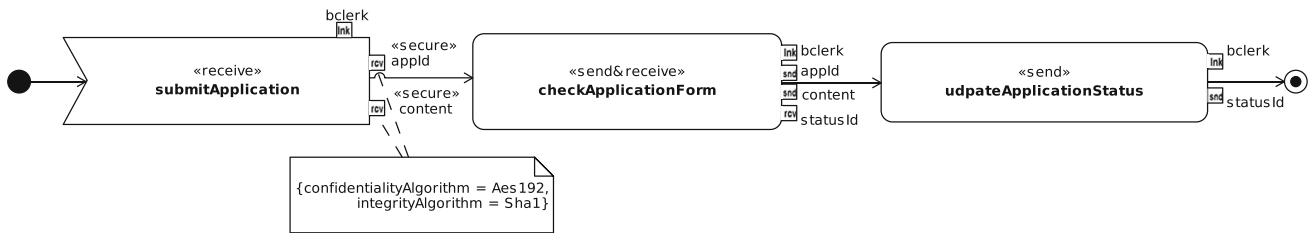


Fig. 27 Provider (bank clerk) service activity

6.2.2 Orchestration specification

A `ServiceActivityNode` models the behavior of a Participant across all service and request ports (see Sect. 5; [38]). While the choreography for `ServiceInterfaces` lays out the order of service invocations for a service channel, a `ServiceActivityNode` orchestration specification defines the control and object flow between service invocations that are controlled by a particular Participant. In Figs. 26 and 27, we show the `ServiceActivities` (i.e., the orchestration specifications) of the `BankClerk` and `Customer` participants.

The `Customer` service activity is triggered through the `newCreditApplication` operation call (see Fig. 26). As a result of the call request, two variables are stored in the corresponding `ReceivePins` (`appId` and `content`). After receiving the `newCreditApplication` call, the `Customer` executes the `submitApplication` operation. The `submitApplication` action specifies two

`SendPins` (`appId` and `content`) which hold the input parameters that are transmitted in the corresponding operation call. The participant then enters a waiting state, until it receives the `updateApplicationStatus` request through its `customer` port (see Fig. 26).

To ensure consistency between the models, the `SecureObjectFlows` extension requires that `SendPins` and `ReceivePins`, which model `SecureNodes` in the choreography (see Fig. 25), also include corresponding security properties in the respective service activity (see Fig. 26). Therefore, the `SendPins` of the `submitApplication` action (`appId` and `content`) include confidentiality and integrity properties which refer to the `OutputPins` of the `newCreditApplication` action in the associated invocation protocol (see Figs. 25, 26).

The orchestration specification for the `BankClerk` participant is shown in Fig. 27. This specification stipulates a blocking `Send&ReceiveAction` on the `checkApplicationForm` operation. Thus, after sending

the request data (i.e., the `appId` and `content`), the action waits for a response which is then stored in the `ReceivePin` (`statusId`). Note that the `submitApplication` `ReceiveAction` includes secure object flow annotations (see Fig. 27). Again, the security properties for the `submitApplication` action establish a consistency link between the choreography activity and the respective service activity (see Figs. 25, 27).

6.3 Intermediary model transformations

After modeling the SOA structure and behavior (see Sects. 6.1, 6.2), the respective models are transformed into an intermediary PIM. We extended the MDD4SOA Eclipse plug-in [36,39] to automate this step for models that include secure object flows.

The corresponding processing steps perform model-to-model transformations via customized rule-based translations (see, e.g., [42]). In particular, the XMI representation (XML Metadata Interchange [55]) of the SoaML models is transformed into an intermediate object model (IOM). The primary objective of the SoaML-to-IOM transformation is to bridge between the graph-based UML PIMs (i.e., extended UML activities, see Sects. 4, 5) and the block-based PSMs defined via WS-BPEL (see, e.g., [41]). For example, in the UML-based PIMs loops are modeled via specific control nodes and control flow edges between nodes. However, because our PSM process execution format (WS-BPEL) does not allow for the definition of graphs that contain cycles, control flow loops at the PIM-level must be translated to block-structured loops in the PSMs (see, e.g., [35]).

The XMI representation of the SoaML structure and behavior models (see Sects. 6.1, 6.2) serves as input for the intermediary model transformation. Listing 1 shows an excerpt from the XMI representation for the secure choreography from Fig. 25. In particular, it shows the assignment of the activity as `ownedBehavior` to the `CreditApplicationService` class. In addition, Listing 1 includes the `appId` and `content` `ActivityParameterNodes` (lines 12–17). An example of a «secure» stereotype instance and the respective slot values are shown in line 30. It refers to the `ActivityParameterNode` named `content` (referenced via `id = 4149`, see lines 15–17).

The transformation maps the XMI representation of the SoaML model to the IOM which is implemented on top of Eclipse's Ecore facility. The Ecore metamodel is based on the essential meta object facility (EMOF, [53]) standard and supported by the Eclipse Modeling Framework (EMF, [83]) project.

The Ecore model of the MDD4SOA IOM provides all stereotypes that are required by the UML4SOA profile via Ecore classes (EClass). The MDD4SOA infrastruc-

ture defines three Ecore packages for the intermediary metamodel: `Statik`, `Behaviour`, and `Data` (see [37]). The `Statik` Ecore package contains intermediary abstractions corresponding to SoaML's composite structure metamodel. The structural abstractions include Ecore classes for participants, service endpoints, and so on. Similarly, the `Behaviour` Ecore package provides an EClass for service activities. In order to transform `SecureInterfaces` (see Sects. 5.3, 5.4) and their choreography specifications into any IOM representation we had to extend the `Statik` package. The `Data` and `Behaviour` Ecore packages did not need to be changed. In particular, we had to address the following requirements to integrate the SOF package (see Sect. 5) into the IOM.

```

1 <xmi:XMI [...]>
2 [...]
3 <packageElement xmi:type="uml:Class" xmi:id="3709" name="
  CreditApplicationService" clientDependency="7807"
  classifierBehavior="3841">
4   <ownedBehavior xmi:type="uml:Activity" xmi:id="3841" name="
     SecureInvocationProtocol" isReentrant="true" partition="
     1975">
5     [...]
6     <ownedParameter xmi:id="4081" name="appId" visibility="public"
       isStream="true">
7       <type xmi:type="uml:PrimitiveType" href="pathmap://
         UML_LIBRARIES/JavaPrimitiveTypes.library.uml#int"/>
8     </ownedParameter>
9     <ownedParameter xmi:id="4150" name="content" visibility="
     public" isStream="true">
10      <type xmi:type="uml:PrimitiveType" href="pathmap://
        UML_LIBRARIES/UMLPrimitiveTypes.library.uml#String"/>
11    </ownedParameter>
12    <node xmi:type="uml:ActivityParameterNode" xmi:id="4080" name
      ="appId" visibility="public" outgoing="1817"
      inPartition="3917" parameter="4081">
13      <type xmi:type="uml:PrimitiveType" href="pathmap://
        UML_LIBRARIES/JavaPrimitiveTypes.library.uml#int"/>
14    </node>
15    <node xmi:type="uml:ActivityParameterNode" xmi:id="4149" name
      ="content" visibility="public" outgoing="1801"
      inPartition="3917" parameter="4150">
16      <type xmi:type="uml:PrimitiveType" href="pathmap://
        UML_LIBRARIES/UMLPrimitiveTypes.library.uml#String"/>
17    </node>
18    <node xmi:type="uml:InitialNode" xmi:id="4295" name=""
      visibility="public" outgoing="1831" inPartition="3917"/>
19    [...]
20    <node xmi:type="uml:CallOperationAction" xmi:id="1593" name="
      " visibility="public" outgoing="1767" inPartition="3917"
      operation="4236">
21      <argument xmi:id="1604" name="statusId" visibility="public"
        incoming="2359" inPartition="3917">
22        <type xmi:type="uml:PrimitiveType" href="pathmap://
          UML_LIBRARIES/JavaPrimitiveTypes.library.uml#int"/>
23      </argument>
24    </node>
25    [...]
26  </ownedBehavior>
27  [...]
28 </packageElement>
29 [...]
30 <SOF:secure xmi:id="q1kQ" confidentialityAlgorithm="2366"
  confidentialityEnsured="true" integrityAlgorithm="2053"
  integrityEnsured="true" base_ObjectNode="4149"/>
31 [...]
32 </xmi:XMI>

```

Listing 1 XMI excerpt from a secure choreography activity

Transformation Requirement 1 *Make all UML stereotypes defined in the SOF profile package available as model elements in the Ecore-based IOM.*

To meet this requirement, we added the EClass `Secure` to the `Statik` Ecore package (see Fig. 28). The `Secure` EClass represents the «secure» stereotype (see Sect. 5.4). Instances of the `Secure` EClass describe instances of the IOM's `InterfaceParameter` (see Fig. 28). The `InterfaceParameter` EClass represents input and output parameters for `InterfaceOperations`. Because

Fig. 28 Ecore-based IOM of the SOF extension

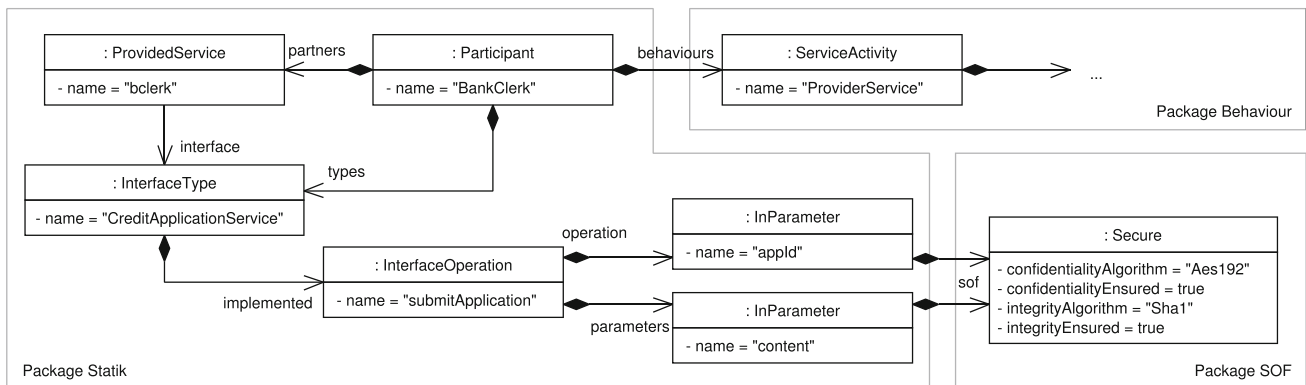
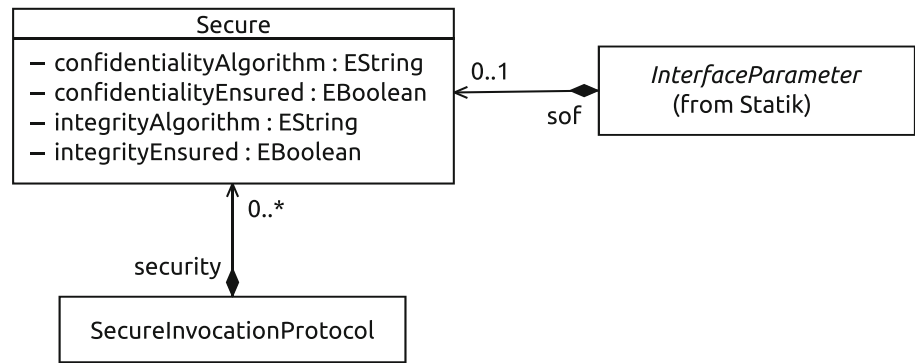


Fig. 29 Excerpt from the IOM object model for the credit application example

the `Pin` types used in `ServiceActivityNodes` and `SecureInterface` choreography activities map to the input and output parameters of the corresponding UML operations, the `InterfaceParameter` EClass qualifies as an appropriate extension point for attaching secure object flow properties at the IOM level.

Transformation Requirement 2 *Define transformation rules for converting an XMI-based into an IOM-based representation of secure object flows.*

We implemented a converter component to transform secure object flows into an IOM structure. Figure 29 shows an excerpt from the IOM model that the converter produced for the credit application example. In principle, the conversion from SoaML/XMI artifacts into IOM instances involves three steps: First, we identify the `SecureInterfaces` and their owned activities (i.e., the choreography specifications) from the SoaML/XMI document. Second, the secure object flows from the choreography specifications that involve an invocation between two entities (e.g., two distinct ports) are selected for further processing. Then, the `SecureNodes` included in these object flows are mapped to IOM instances (in particular instances of type `InterfaceParameter`, see also Fig. 28). Third, the resulting IOM is serialized into its Ecore/XMI representation.

6.4 Platform-specific model transformations

After the intermediary model transformations, our extended MDD4SOA plug-in creates platform-specific models (PSMs) for a selected execution platform. While our approach is not limited to this platform family, the MDD tool chain described in this paper integrates with Web services communication middleware. In particular, the IOM representations of the SOA composite structure (i.e., the service interfaces and message type specifications) are transformed into WSDL interface descriptions [92], and the behavioral parts (i.e., the service activities) are translated into WS-BPEL execution specifications [61]. For our credit application example, we generate WS-BPEL specifications for the Apache orchestration director engine (ODE, [2]). The corresponding secure object flow properties are transformed into WS-SecurityPolicy statements [63]. Finally, supplemental artifacts such as deployment descriptors for Apache ODE are created.

Transformation Requirement 3 *Secure object flow properties of the PIM must be mapped to the PSM.*

The WS-SecurityPolicy standard [63] allows to define a number of security binding properties. In particular, it provides a list of security algorithm suites for cryptographic operations with symmetric or asymmetric encryption mech-

Table 8 Examples for WS-SecurityPolicy algorithm suites

Encryption	Aes128	Aes192	Aes256	TripleDes
Digest	Sha1	Sha1	Sha1	Sha1
<i>Algorithm suite</i>	<i>Basic128</i>	<i>Basic192</i>	<i>Basic256</i>	<i>TripleDes</i>

anisms. To be compliant with the WS-SecurityPolicy standard, Web service communication middleware must provide software support for these security bindings (message-level encryption). Each algorithm suite specifies the actual algorithm and the respective key lengths [63]. Table 8 shows encryption algorithms and hash functions which can be applied in the credit application example.

Thus, our extended MDD4SOA plug-in must perform an automated mapping of secure object flow properties (i.e., confidentiality and integrity attributes) to the algorithm suites specified by the WS-SecurityPolicy standard. In the credit application example, the following security properties have been defined (see also Figs. 25, 26, 27): confidentialityAlgorithm = Aes192 and integrityAlgorithm = Sha1. Both, encryption algorithm and integrity algorithm, correspond to the *Basic192* algorithm suite of the WS-SecurityPolicy standard (see Table 8).

Transformation Requirement 4 *Implement transformation rules for converting the IOM/XMI representation into the PSMs.*

Because our PSMs are Web service artifacts, we map the Ecore representation of the Statik IOM package (e.g., the EClasses Service, InterfaceOperation, InterfaceParameter; see Sect. 6.3) to WSDL descriptions. The artifacts from the Behaviour package are translated to WS-BPEL definitions (e.g., ServiceActivityNode or ServiceProtocol; see [37]). The EClass from our security extension in the IOM SOF package is transformed into WS-SecurityPolicy fragments.

The WS-SecurityPolicy specification allows to define nested policy assertions (see [94]). In principle, the WS-SecurityPolicy standard defines three attachment points for policies (called *policy subjects*, see [93]):

Endpoint policy subject A policy that applies to the service at the endpoint level.

WSDL attachment points: wsdl:binding and wsdl:port.

Operation policy subject A policy on a per-operation basis. WSDL attachment points: wsdl:binding/wsdl:operation.

Message policy subject A policy at the message level.

WSDL attachment points: wsdl:binding/wsdl:operation/wsdl:input, wsdl:binding/wsdl:operation/wsdl:output, and wsdl:binding/wsdl:operation/wsdl:fault.

A benefit of the SecureObjectFlows extension is its ability to model data security properties at the level of individual object flows. Therefore, we are able to model confidentiality and integrity requirements of single invocation parameters, if necessary. This level of detail can be expressed through the *Message Policy Subject* of the WS-SecurityPolicy specification (see above).

We implemented a converter which adds WS-SecurityPolicy statements to the WSDL interface descriptions (see Listing 2 for an example). In particular, we need to find all InterfaceParameters from the IOM which include confidentiality and integrity properties. For every secured InterfaceParameter, we must identify the corresponding WS-SecurityPolicy algorithm suite (see Table 8). Next, we have to generate a policy assertion for each interface parameter with the corresponding SignedElements, EncryptedElements, ContentEncryptedElements, and AlgorithmSuite (see Listing 2, lines 3–22). Subsequently, each message parameter in the binding definition of the WSDL document is extended to contain a reference to the corresponding policy assertion in terms of a PolicyReference statement (see Listing 2, line 30).

```

1 <definitions [...] name="bclerk" [...]>
2 [...]
3 <wsp:Policy wsu:Id="sp_submitApplication_input_appId">
4 <wsp:ExactlyOne>
5 <wsp:All>
6 <sp:SignedElements>
7 <sp:XPath>/Envelope/Body//appId</sp:XPath>
8 </sp:SignedElements>
9 <sp:EncryptedElements>
10 <sp:XPath>/Envelope/Body//appId</sp:XPath>
11 </sp:EncryptedElements>
12 <sp:ContentEncryptedElements>
13 <sp:XPath>/Envelope/Body//appId</sp:XPath>
14 </sp:ContentEncryptedElements>
15 <sp:AlgorithmSuite>
16 <wsp:Policy>
17 <sp:Basic192/>
18 </wsp:Policy>
19 </sp:AlgorithmSuite>
20 </wsp:All>
21 </wsp:ExactlyOne>
22 </wsp:Policy>
23 [...]
24 <binding name="bclerk_binding" type="this:bclerk">
25 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
26 <operation name="submitApplication">
27 <soap:operation soapAction="submitApplication"/>
28 <input name="submitApplication_input">
29 <soap:body namespace="http://www.mdd4soa.eu/generated/BankClerk/" use="literal"/>
30 <wsp:PolicyReference URI="#sp_submitApplication_input_appId"
    required="true"/>
31 [...]
32 </input>
33 </operation>
34 [...]
35 </binding>
36 [...]
37 </definitions>

```

Listing 2 Excerpt from a WSDL document with WS-SecurityPolicy assertions

Transformation Requirement 5 *Execution of generated PSMs in a selected environment.*

The credit application example was deployed in the Apache ODE 1.3.5. Apache Axis2 [1] serves as the integration layer for the communication over Web services. Apache Rampart (the security module of Axis2; [3]) enforces the corresponding WS-SecurityPolicy specifications. With the automated creation of deployment descriptors, we realize the integrated model-driven specification of secure object flows from the CIM and PIM levels down to the execution of the PSM in a specific platform environment (here: a SOA environment).

7 Related work

We use three main characteristics to review related approaches: 1) whether a *formal and generic metamodel* (CIM) is provided, 2) whether *modeling support* (for PIMs) is provided, and 3) whether an *integrated tool chain* is available (including automated model transformations into PSMs). For each of these characteristics, we further consider multiple sub-criteria. Table 9 shows an overview of related work on modeling of secure object flows for process-driven SOAs. With respect to the concepts and artifacts specified in Sects. 3–6, we use a \surd if a related approach provides similar or comparable support for a certain concept; and a Δ if a related approach provides at least partial support for a particular concept.

Wolter et al. [89,90] presented an approach to model security goals in business processes. The security goals are mapped to a workflow model, and finally to executable Web service specifications. Wolter et al. provide generic metamodels for security properties that are expressed via concept diagrams which loosely resemble UML class diagrams, whereas the workflow models use BPMN (see also [91]). Informal semantics can be derived from UML-like security policy diagrams and the assisting comments. However, they do not provide an integrated metamodel for the business process view and crosscutting security views (neither at the CIM nor at PIM level). As a result, Wolter et al. only sketch their model transformation framework. In particular, they do not describe how the model transformations actually merge the security goals and the BPMN process descriptions to produce platform-specific Web service artifacts. Furthermore, the behavioral model view (BPMN) only provides model annotations for security properties without processable or formalized semantic constraints. Therefore, the approach does not support the specification of security properties for object flows at various abstraction levels (such as process assets, process execution data, invocation data, messages). Regarding tool support, the authors provide an overview of a transformation framework based on annotated

XMI representations of BPMN models. However, a detailed discussion of how the security properties are integrated with the XMI representation is missing.

Gilmore et al. [20] discuss modeling of non-functional aspects for service-oriented systems. They define a UML profile based on the SoaML that supports the definition of non-functional properties (performance, reliable messaging, and security). Similar to our work, the approach of Gilmore et al. uses the UML4SOA extension [36,38,39]. However, in contrast to [20] our work is based on a formal and generic metamodel (CIM) that integrates the process flow view with our security extension (see also [85]). Thus, in principle, our CIM for secure object flows can be used to extend arbitrary process modeling languages. Moreover, Gilmore et al. treat security specifications along with non-functional properties such as QoS and emphasize a (predominantly) structural view on security properties. The structural view especially includes non-functional contracts that are associated with SoaML interfaces. However, this design choice also limits the expressiveness, in particular, security properties cannot be specified at different abstraction levels (such as message parts, scope of single invocations, or context dependencies). Gilmore et al. define non-functional properties as part of a service interface. Therefore, a contract specifies requirements for every interaction between the corresponding participants. Thus, it is not possible to define security properties at the level of individual service invocations or at the level of separate message parameters. In contrast, our secure object flows provide means for a much more fine-grained specification of security properties from the CIM to the PSM level. However, the approach from [20] could be integrated with our approach. The level of service contracts could add an additional, global specification scope for integrity and confidentiality properties.

Jürjens et al. present UMLsec [32,33], a UML profile that supports the definition of various security properties. For example, UMLsec is used to define and verify cryptographic protocols. Moreover, data confidentiality and integrity stereotypes can define dependencies in static structure or component diagrams. While UMLsec does not explicitly target SOAs, it could be integrated with other UML-based approaches such as the SoaML with moderate effort. Jürjens et al. offer tool support for running static and behavioral checks and a permission analyzer for access control mechanisms (see [78]). The original UMLsec approach was defined using UML 1.5 [32]. Recently, they introduced a corresponding UML 2.3 profile [78] as well as Eclipse-MDT-based tool support [88]. However, regarding model-driven development UMLsec does not (yet) provide a model transformation framework and, consequently, does not address platform integration issues. Therefore, UMLsec is well-suited to be integrated with our approach. For example, UMLsec models can be supported by our model transformation framework to target other (non-SOA) platforms.

Table 9 Secure object flows for process-driven SOAs: related work

	Integrity and confidentiality	Formal and generic metamodel (CIM)	Integrated metamodel (CIM, PIM, PSM)	Formal semantics	Separation of concerns	Multiple model views	Multiple refinement levels	Tool support	Model transformation framework
Model-driven sec. requirement spec. [89,90]	✓	△						△	✓
Non-functional properties in MDD [20]	✓		✓	△	△	△		△	✓
Model-driven security [5]		✓	△	✓	△			✓	△
Secure sys. development with UML [32,33]	✓	✓	△	✓	△	✓	✓	△	
Inter-org. workflow sec. [21–24,40]	✓	△	✓	△	△	✓	✓	✓	✓
Sec. for WS-based business processes [31]	✓					✓		△	✓
Secure business process model spec. [66,67]	✓	△	△	△				△	△
MDD of security aspects [65]		△	△		△			△	✓
MDA approach to access control spec. [15]		△	✓	△				△	△
Generative arch. for model-driven sec. [68]	✓	△			△		✓	△	✓
Model-driven security based on WS [44,86]	✓		△					△	△
Secure object flows (our approach)	✓	✓	✓	✓	✓	✓	✓	✓	✓

The SECTET framework [21–24, 40] aims at supporting the design and implementation of security-critical inter-organizational workflows based on Web services. The approach includes a metamodel consisting of multiple UML class diagrams. However, SECTET is not based on a formal CIM. Yet, because SECTET is based on the UML, the respective PIMs are (implicitly) based on a common metamodel (i.e., the UML metamodel). Nevertheless, SECTET does not use SOA-related UML extensions (such as the SoaML). Formal semantics for SECTET models are specified via a custom, OCL-like constraint language. A global workflow view is used for specifying message exchange contracts between service providers and service consumers. However, security properties of object flows between service invocations are not covered.

In [31], Jensen and Feja extend a proprietary MDD software tool for modeling SOA-based security properties. The approach uses event-driven process chains (EPCs), but does not provide a formal and generic CIM. Security models are defined separately from process models and add security-related notation elements to different workflow elements. Security and process models are integrated via a model transformation step. However, the approach does not provide formal semantics for the process-related security properties. In addition, the security properties are defined for the scope of a single process engine (rather than for a collaboration of service partners).

Rodríguez et al. [66, 67] present a UML profile to model security requirements via UML activities. Separation of concerns and multi-level specification of security properties are not supported. The approach provides a single process-related view, yet, it could be extended with additional views provided in our approach. Rodríguez et al. describe PIM transformations to use case and class diagrams. However, they do not define PSM transformations. The PIM transformation rules are specified via QVT [54], but they are not integrated with the overall tool chain. In [67], they describe the modeling of security requirements in BPMN and the corresponding model-to-model transformations into UML activity diagrams. Rodríguez et al. do not provide a formal and generic metamodel (CIM).

Reznik et al. [65] address the automatic generation of security-critical applications for different middleware platforms. The approach defines a UML profile for an adapted subset of the UML metamodel. Corresponding PIMs are mapped to a security-extended CORBA component model. While their PIMs are based on a UML subset, Reznik et al. redefine some parts of the UML metamodel. As a result, the corresponding subset it is not compliant with the UML standard. Moreover, because their UML profile is based on the adapted UML subset it is not supported by standard UML tools. The approach of Reznik et al. does neither provide a

CIM nor does it discuss multi-view modeling or refinement of security properties.

In [68], Sanchez et al. present a model-driven approach for the definition of different security properties. The approach is based on multi-stage, automatic model transformations. The generic metamodel of the approach is defined as a UML M1 class diagram. Sanchez et al. do not provide a CIM, they emphasize that their approach is limited to a specific modeling language. In [68], they describe a custom modeling language that uses the Eclipse Ecore facility. Hence, regarding syntax and semantic, their generic UML-based metamodel does not integrate seamlessly with their custom modeling language.

Nakamura et al. [44] and Tatsubori et al. [86] present a toolkit for generating Web service security configurations. UML class models are applied to provide a structural view on a SOA. In addition, they provide stereotypes to define selected security properties. However, the corresponding UML profile is defined in an ad-hoc fashion and does not conform to the UML standard. Process views are not considered, but could be added via an extension of their approach. Nakamura et al. and Tatsubori et al. neither define a formal or generic metamodel (CIM) nor do they discuss formal semantics or issues regarding separation of concerns.

In addition to the approaches described above, multiple other model-driven security approaches exist that do not address the modeling of secure object flows but focus on other process-related security aspects. For example, Fink et al. [15] propose to generate access control specifications from MOF-based models, and Basin et al. [5] specify access control properties via domain-specific UML profiles.

8 Conclusion

In this paper, we presented an integrated approach to model and to enforce secure object flows in process-driven SOAs. In particular, we provide a generic metamodel (CIM) for secure object flows, a corresponding UML extension to define platform-independent models (PIM), a model transformation framework for PIM-to-PSM transformations for Web service artifacts, as well as corresponding tool support. Our approach enables the continuous specification and the enforcement of confidentiality and integrity properties for object flows in business processes that are executed in a distributed system. Moreover, secure object flows are a part of the Business Activities framework and are thereby directly integrated with extensions for the specification, for checking, and for the enforcement of other security properties, such as access control or audit rules (see, e.g., [6, 28, 30, 71–76, 84, 85]).

Our approach follows the model-driven development paradigm. At the PIM level, we provide both a metamodel

extension for secure object flows (see Sect. 4) as well as a corresponding UML profile (see Sect. 5). For both extensions, we provide OCL constraints that define consistency requirements for the corresponding modeling artifacts. Therefore, each valid secure object flow must conform to the respective OCL constraints. Our metamodel extension provides specifically tailored UML elements. It can be used to extend future versions of UML tools with native modeling support for secure object flows. In contrast, a UML profile adapts *existing* UML metaclasses for non-standard purposes. In our case, it extends the SoaML/UML4SOA with modeling support for secure object flows. Because most UML tools directly support the definition of profiles, it is comparatively easy to integrate UML profiles in a software tool. Our tool support for secure object flows in process-driven SOAs is based on the Eclipse Model Development Tools (see Sect. 6). Our model transformation framework generates WS-BPEL, WSDL, and WS-SecurityPolicy artifacts. The generated artifacts were deployed in the Apache orchestration director engine (ODE). Apache Axis2 serves as the integration layer for the communication over Web services. Apache Rampart enforces the corresponding WS-SecurityPolicy specifications. However, note that our approach is generic and does neither depend on the UML nor on a specific software tool or runtime environment. Thus, it can also be applied to extend other modeling languages or other software tools.

In addition to the main contributions of this paper, we also gained numerous other experiences and insights. Some of which are specific to certain design artifacts. For example, during the tool integration, it turned out that the UML4SOA extension [38] deviates from the UML standard by using an AcceptCallAction with the «Receive» stereotype for defining an event or message listener (i.e., a ServiceReceiveAction). Moreover, it demands that a specific stereotype is defined on the corresponding input pins. However, the resulting constraint is not compliant with the UML standard. This small yet important anomaly directly influenced the tool integration and corresponding PIM-to-PSM transformations. In general, such small anomalies may always occur if one integrates different modeling extensions in a consolidated tool environment.

In our future work, we will extend the Business Activities framework to provide an integrated environment for the model-driven specification, checking, deployment, and enforcement of secure business processes in distributed systems.

Appendix A: Constraints for the SecureObjectFlows package

This section provides the complete list of OCL-expressions for the UML extension specified in Sect. 4.

OCL Constraint 1 *The confidentialityEnsured attribute of the SecureNode classifier is derived from the confidentialityAlgorithm attribute and evaluates to true if a confidentiality-related security property is supported.*

```
1 context SecureObjectFlows::SecureNode::confidentialityEnsured : Boolean
2 derive: confidentialityAlgorithm->notEmpty()
```

OCL Constraint 2 *The integrityEnsured attribute of the SecureNode classifier is derived from the integrityAlgorithm attribute. It evaluates to true if an integrity-related security property is supported.*

```
1 context SecureObjectFlows::SecureNode::integrityEnsured : Boolean
2 derive: integrityAlgorithm->notEmpty()
```

OCL Constraint 3 *A secure object node must ensure either or both the confidentiality and the integrity.*

```
1 context SecureObjectFlows::SecureNode
2 inv: self.confidentialityEnsured or
3 self.integrityEnsured
```

OCL Constraint 4a *The successor object node of a secure object flow must also be a secure object node.⁷*

```
1 context SecureObjectFlows::SecureNode
2 def: allSuccessors(node : ActivityNode) : Set(ActivityNode) =
  node.outgoing.target->collect(x |
3 allSuccessors(x))->asSet()->union(node.outgoing.target)
4 inv: allSuccessors(self)->select(oclIsKindOf(ObjectNode))->
  forAll(node |
5 node.oclIsKindOf(SecureNode))
```

OCL Constraint 4b *The successor object node of a secure object flow must also be a secure object node.⁸*

```
1 context SecureObjectFlows::SecureNode
2 def: allSuccessors(node : ActivityNode) : Set(ActivityNode) =
  node.outgoing.target->closure(x |
3 x.outgoing.target)->asSet()->union(node.outgoing.target)
4 inv: allSuccessors(self)->select(oclIsKindOf(ObjectNode))->
  forAll(node |
5 node.oclIsKindOf(SecureNode))
```

OCL Constraint 5 *The successor secure object nodes must support the same security properties as the corresponding source secure object node.*

```
1 context SecureObjectFlows::SecureNode
2 inv: allSuccessors(self)->select(oclIsKindOf(SecureNode))->
  forAll(node |
3 node.oclAsType(SecureNode).confidentialityEnsured implies
  node.oclAsType(SecureNode).confidentialityAlgorithm =
  self.confidentialityAlgorithm)
4 inv: allSuccessors(self)->select(oclIsKindOf(SecureNode))->
  forAll(node |
5 node.oclAsType(SecureNode).integrityEnsured implies
6 node.oclAsType(SecureNode).integrityAlgorithm = self.
  integrityAlgorithm)
```

OCL Constraint 6a *All secure object nodes having the same target secure object node must support identical security properties.⁹*

```
1 context SecureObjectFlows::SecureNode
2 def: allPredecessors(node : ActivityNode) : Set(ActivityNode) =
  node.incoming.source->collect(x |
3 allPredecessors(x))->asSet()->union(node.incoming.source)
4 inv: allPredecessors(self)->select(oclIsKindOf(SecureNode))->
  forAll(node |
5 node.oclAsType(SecureNode).confidentialityEnsured implies
6 node.oclAsType(SecureNode).confidentialityAlgorithm =
  self.confidentialityAlgorithm)
7 inv: allPredecessors(self)->select(oclIsKindOf(SecureNode))->
  forAll(node |
8 node.oclAsType(SecureNode).integrityEnsured implies
9 node.oclAsType(SecureNode).integrityAlgorithm = self.
  integrityAlgorithm)
```

OCL Constraint 6b *All secure object nodes having the same target secure object node must support identical security properties.¹⁰*

⁷ This constraint conforms to the OCL standard version 2.2 [56].

⁸ This constraint conforms to the OCL standard version 2.3.1 [57].

⁹ This constraint conforms to the OCL standard version 2.2 [56].

¹⁰ This constraint conforms to the OCL standard version 2.3.1 [57].

```

1 context SecureObjectFlows::SecureNode
2 def: allPredecessors(node : ActivityNode) : Set(ActivityNode) =
  node.incoming.source->closure(x |
3     x.incoming.source)->asSet()->union(node.incoming.source)
4 inv: allPredecessors(self)->select(oclIsKindOf(SecureNode))->
  forAll(node |
5     node.oclAsType(SecureNode).confidentialityEnsured implies
6     node.oclAsType(SecureNode).confidentialityAlgorithm =
  self.confidentialityAlgorithm)
7 inv: allPredecessors(self)->select(oclIsKindOf(SecureNode))->
  forAll(node |
8     node.oclAsType(SecureNode).integrityEnsured implies
9     node.oclAsType(SecureNode).integrityAlgorithm = self.
  integrityAlgorithm)

```

Appendix B: Constraints for the SecureObjectFlows::Services package

This section provides the complete list of OCL constraints for the UML extension specified in Sect. 5.

OCL Constraint 7 A SecureInterface must own an Activity instance as its owned behavior.

```

1 context SecureObjectFlows::Services::SecureInterface
2 inv: self.ownedBehavior->one(oclIsKindOf(Activity))

```

OCL Constraint 8 In strict mode all cross-interface object flows must be secured.¹¹

```

1 context SecureObjectFlows::Services::SecureInterface
2 def: allPredecessors(node : ActivityNode) : Set(ActivityNode) =
  node.incoming.source->collect(x |
3     allPredecessors(x))->asSet()->union(node.incoming.source)
4 inv: self.isStrict implies
  self.ownedBehavior.oclAsType(Activity).node->select(
5     oclIsKindOf(ObjectNode))->forAll(node |
6     allPredecessors(node)->select(incoming->isEmpty()->
  forAll(s |
7         s.inPartition << node.inPartition implies
8         s.oclIsKindOf(SecureNode) and node.oclIsKindOf(
  SecureNode)))

```

OCL Constraint 9 All Actions must be instances of CallOperationAction and each CallOperationAction's operation enclosed by a given partition must correspond to an Operation owned by the Interface denoted by this partition.

```

1 context SecureObjectFlows::Services::SecureInterface
2 inv: self.ownedBehavior.oclAsType(Activity).node->select(
  oclIsKindOf(Action))->forAll(a |
3     a.oclIsKindOf(CallOperationAction) and
4     self.part->any(name = a.inPartition->any(true).name).type
  .oclAsType(Interface).ownedOperation->
5     includes(a.oclAsType(CallOperationAction).operation))

```

OCL Constraint 10 Corresponding secure object nodes must reside in different partitions.¹²

```

1 context SecureObjectFlows::Services::SecureNode
2 def: allPredecessors(node : ActivityNode) : Set(ActivityNode) =
  node.incoming.source->collect(x |
3     allPredecessors(x))->asSet()->union(node.incoming.source)
4 inv: allPredecessors(self)->select(incoming->isEmpty() and
  oclIsKindOf(SecureNode))->forAll(s | s.inPartition << self
5     .inPartition)

```

OCL Constraint 11 All activity nodes must be assigned to and must be contained by exactly one and only one activity partition.

¹¹ Here, an OCL 2.3.1 compliant definition is omitted. For an OCL 2.3.1 compliant definition of allPredecessors() see OCL Constraint 6b in Appendix A.

¹² Here, an OCL 2.3.1 compliant definition is omitted. For an OCL 2.3.1 compliant definition of allPredecessors() see OCL Constraint 6b in Appendix A.

```

1 context SecureObjectFlows::Services::SecureInterface
2 inv: self.ownedBehavior.oclAsType(Activity).node->forAll(
  inPartition->size() = 1)

```

OCL Constraint 12 Only InputPins, OutputPins, and ActivityParameterNodes can be secured. All secured InputPins must have an incoming object flow; all secured OutputPins must have an outgoing object flow. Secured ActivityParameterNodes must either be connected to an incoming object flow, to an outgoing object flow, or to both; depending on the parameter direction.

```

1 context SecureObjectFlows::Services::SecureNode
2 inv: self.oclIsKindOf(InputPin) or self.oclIsKindOf(OutputPin)
  or self.oclIsKindOf(ActivityParameterNode)
3 inv: self.oclIsKindOf(InputPin) implies self.incoming->notEmpty()
4 inv: self.oclIsKindOf(OutputPin) implies self.outgoing->
  notEmpty()
5 inv: self.oclIsKindOf(ActivityParameterNode) implies
6     (self.oclAsType(ActivityParameterNode).parameter.
  direction = ParameterDirectionKind::in or
7     self.oclAsType(ActivityParameterNode).parameter.direction
  = ParameterDirectionKind::inout implies
8     self.incoming->notEmpty()) and
9     (self.oclAsType(ActivityParameterNode).parameter.
  direction = ParameterDirectionKind::out or
10    self.oclAsType(ActivityParameterNode).parameter.direction
  = ParameterDirectionKind::inout or
11    self.oclAsType(ActivityParameterNode).parameter.direction
  = ParameterDirectionKind::return implies
12    self.outgoing->notEmpty())

```

OCL Constraint 13 All ActivityParameterNodes which are not initial or final nodes in a control and data flow but counterparts of intermediary InputPins and OutputPins must refer to a streaming Parameter.¹³

```

1 context SecureObjectFlows::Services::SecureInterface
2 def: isFirstNode(a : ActivityNode) : Boolean =
  a.owner.oclAsType(Activity).node->select(oclIsKindOf(
3     InitialNode))->exists(outgoing.target->any(true) =
  a) or
4     a.owner.oclAsType(Activity).node->select(oclIsKindOf(
  ActivityNode) and incoming->isEmpty()->includes(a)
5     isLastNode(a : ActivityNode) : Boolean =
  a.owner.oclAsType(Activity).node->select(oclIsKindOf(
6     ActivityFinalNode))->exists(incoming.source->any(
  true) = a) or
7     a.owner.oclAsType(Activity).node->select(oclIsKindOf(
  ActivityNode) and outgoing->isEmpty()->includes(a)
8     def: allSuccessors(node : ActivityNode) : Set(ActivityNode) =
  node.outgoing.target->collect(x |
9     allSuccessors(x))->asSet()->union(node.outgoing.target)
10    inv: self.ownedBehavior.oclAsType(Activity).node->select(
  oclIsKindOf(ActivityNode))->forAll(an |
11    (not isFirstNode(an) implies
12    an.input->forAll(ipin | allPredecessors(ipin)->select(
  oclIsKindOf(ActivityParameterNode))->forAll(
13    oclAsType(ActivityParameterNode).parameter.isStream))
  ) and
14    (not isLastNode(an) implies
  an.output->forAll(opin | allSuccessors(opin)->select(
15    oclIsKindOf(ActivityParameterNode))->forAll(
  oclAsType(ActivityParameterNode).parameter.isStream))
16    ))

```

OCL Constraint 14 All source object nodes of a set of InputPins owned by a CallOperationAction must be assigned to the same activity partition.

```

1 context UML::ObjectNode
2 inv: self.activity.owner.oclIsKindOf(SecureInterface) and
3     self.oclIsKindOf(InputPin) implies
4     self.oclAsType(InputPin).owner.oclAsType(
  CallOperationAction).input->forAll(ipin |
5     allPredecessors(ipin)->select(incoming->isEmpty() and
  oclIsKindOf(ObjectNode))->forAll(on1,on2 |
6     on1.inPartition = on2.inPartition))

```

OCL Constraint 15 If provided for a Participant, the ServiceActivityNode must contain a corresponding and compatible SecureSendPin for each secured InputPin in a choreography activity; provided that a) there is a choreography activity in

¹³ Here, an OCL 2.3.1 compliant definition is omitted. For an OCL 2.3.1 compliant definition of allSuccessors() see OCL Constraint 4b in Appendix A.

the first place, and that b) the CallOperationAction owning the InputPin and the ServiceInteractionAction owning the SecureSendPin share the Operation (required from the same Interface).

```

1 context UML4SOA::ServiceActivityNode
2 inv: self.node->select((oclIsKindOf(ServiceSendAction) or
   oclIsKindOf(ServiceReplyAction)) and
3   oclAsType(ServiceInteractionAction).input->select(
   oclIsKindOf(SecureSendPin))>forall(sa |
4     let p : Port =
5       if oclIsKindOf(ServiceSendAction)
6         then sa.oclAsType(ServiceSendAction).target.oclAsType(
           Port)
7         else sa.oclAsType(ServiceReplyAction).returnInformation
           .oclAsType(Port)
8     endif
9   in
10    p.type.oclAsType(SecureInterface) and
11    p.type.oclAsType(SecureInterface).ownedBehavior.
   oclAsType(Activity).node->select(
12      let op : Operation =
13        if oclIsKindOf(ServiceSendAction)
14          then sa.oclAsType(ServiceSendAction).operation
15          else sa.oclAsType(ServiceReplyAction).replyToCall.
   oclAsType(CallEvent).operation
16        endif
17      in
18        oclIsKindOf(CallOperationAction) and
19        oclAsType(CallOperationAction).operation = op->
   forall(
20          oclAsType(CallOperationAction).input->select(
   oclIsKindOf(SecureNode))>forall(i |
21            sa.input->exists(
22              name = i.name and
23              type = i.type and
24              (i.oclAsType(SecureNode).integrityEnsured =
   oclAsType(SecureNode).integrityEnsured)
   implies
25                i.oclAsType(SecureNode).integrityAlgorithm
   = oclAsType(SecureNode).
   integrityAlgorithm and
26              (i.oclAsType(SecureNode).
   confidentialityEnsured = oclAsType(
   SecureNode).confidentialityEnsured)
   implies
27                i.oclAsType(SecureNode).
   confidentialityAlgorithm = oclAsType(
   SecureNode).confidentialityAlgorithm
   )))

```

OCL Constraint 16 *If provided for a Participant, the ServiceActivityNode must contain a corresponding and compatible SecureReceivePin for each secured OutputPin in a choreography activity; provided that a) there is a choreography activity in the first place, and that b) the CallOperationAction owning the OutputPin and the ServiceReceiveAction owning the SecureReceivePin share the Operation (required from the same Interface).*

```

1 context UML4SOA::ServiceActivityNode
2 inv: self.node->select(oclIsKindOf(ServiceReceiveAction) and
3   oclAsType(ServiceReceiveAction).output->select(oclIsKindOf(
   SecureReceivePin))>forall(sa |
4     sa.oclAsType(ServiceReceiveAction).returnInformation.
   oclAsType(Port).type.oclAsType(SecureInterface) and
5     sa.oclAsType(ServiceReceiveAction).returnInformation.
   oclAsType(Port).type.oclAsType(SecureInterface).
   ownedBehavior.oclAsType(
6     Activity).node->select(
7       oclIsKindOf(CallOperationAction) and
8       oclAsType(CallOperationAction).operation = sa.
   oclAsType(ServiceReceiveAction).trigger->
9       any(true).oclAsType(CallEvent).operation)>forall(
10        oclAsType(CallOperationAction).input->select(
   oclIsKindOf(SecureNode))>forall(i |
11          sa.output->exists(
12            name = i.name and
13            type = i.type and
14            (i.oclAsType(SecureNode).integrityEnsured =
   oclAsType(SecureNode).integrityEnsured)
   implies
15              i.oclAsType(SecureNode).integrityAlgorithm
   = oclAsType(SecureNode).
   integrityAlgorithm and
16            (i.oclAsType(SecureNode).
   confidentialityEnsured = oclAsType(
   SecureNode).confidentialityEnsured)
   implies
17              i.oclAsType(SecureNode).
   confidentialityAlgorithm = oclAsType(
   SecureNode).confidentialityAlgorithm
   )))

```

References

1. Apache Software Foundation (ASF): Apache Axis2. <http://axis.apache.org/axis2/java/core/> (2012)
2. Apache Software Foundation (ASF): Apache ODE. <http://ode.apache.org> (2012)
3. Apache Software Foundation (ASF): Apache Rampart—Axis2 Security Module. <http://axis.apache.org/axis2/java/rampart/> (2012)
4. Axenath, B., Kindler, E., Rubin, V.: AMFIBIA: a meta-model for the integration of business process modelling aspects. In: Leymann, F., Reising, W., Thatte, S., van der Aalst, W. (eds.) The Role of Business Processes in Service Oriented Architectures, Dagstuhl Seminar Proceedings (2006)
5. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: from UML models to access control infrastructures. ACM Transact. Softw. Eng. Methodol. (TOSEM) **15**(1), 39–91 (2006)
6. Baumgrass, A., Baier, T., Mendling, J., Strembeck, M.: Conformance checking of RBAC policies in process-aware information systems. In: Proceedings of the Workshop on Workflow Security Audit and Certification (WfSAC), Lecture Notes in Business Information Processing (LNBIP), vol. 100. Springer, Berlin (2011)
7. Cannon, J., Byers, M.: Compliance deconstructed. ACM Queue **4**(7), 30–37 (2006)
8. Committee on National Security Systems (CNSS): National Information Assurance (IA): glossary. http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf (2010)
9. Damianides, M.: How does SOX change IT? J. Corp. Account. Finance **15**(6), 35–41 (2004)
10. Eclipse Foundation: Eclipse IDE. <http://www.eclipse.org> (2012)
11. Eclipse Foundation: Eclipse model development tools (MDT). <http://www.eclipse.org/modeling/mdt/> (2012)
12. Eclipse Foundation: Eclipse Papyrus. <http://www.eclipse.org/modeling/mdt/papyrus/> (2012)
13. Elvesæter, B., Berre, A.-J., Sadovykh, A.: Specifying services using the service oriented architecture modeling language (SoaML)—a baseline for specification of cloud-based services. In: Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER'11), pp. 276–285. SciTePress (2011)
14. Elvesæter, B., Carrez, C., Mohagheghi, P., Berre, A.-J., Johnsen, S., Solberg, A.: Model-driven service engineering with SoaML. In: Service Engineering—European Research Results, pp. 25–54. Springer, Berlin (2011)
15. Fink, T., Koch, M., Pauls, K.: An MDA approach to access control specifications using MOF and UML profiles. In: Electronic Notes in Theoretical Computer Science, pp. 161–179 (2006)
16. International Organization for Standardization (ISO): Information technology: security techniques—code of practice for information security management, ISO/IEC 27002:2005, Stage: 90.92. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50297 (2008)
17. International Organization for Standardization (ISO): Information technology: security techniques—information security management systems—requirements, ISO/IEC 27001:2005, Stage: 90.92. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=42103 (2008)
18. International Organization for Standardization (ISO): Information technology—security techniques—information security management systems—overview and vocabulary, ISO/IEC 27000:2009, Stage: 60.60. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=41933 (2009)
19. Foster, H., Gönczy, L., Koch, N., Mayer, P., Montangero, C., Varró, D. UML extensions for service-oriented systems. In: Wirsing, M., Hölzl, M. (eds.) Rigorous Software Engineering for Service-

- Oriented Systems, Lecture Notes in Computer Science (LNCS), pp. 35–60. Springer, Berlin (2011)
20. Gilmore, S., Gönczy, L., Koch, N., Mayer, P., Tribastone, M., Varró, D.: Non-functional properties in the model-driven development of service-oriented systems. *Softw. Syst. Model.* **10**(3), 287–311 (2011)
 21. Hafner, M., Alam, M., Breu, R.: Towards a MOF/QVT-based domain architecture for model driven security. In: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems (MODELS 2006), Lecture Notes in Computer Science (LNCS), pp. 275–290. Springer, Berlin (2006)
 22. Hafner, M., Breu, R.: Security Engineering for Service-Oriented Architectures, 1st edn. Springer, Berlin (2009)
 23. Hafner, M., Breu, R., Agreiter, B., Nowak, A.: SECTET: an extensible framework for the realization of secure inter-organizational workflows. *Internet Res.* **16**(5), 491–506 (2006)
 24. Hafner, M., Memon, M., Alam, M.: Modeling and enforcing advanced access control policies in healthcare systems with SECTET. In: Giese, H. (ed.) Models in Software Engineering, pp. 132–144. Springer, Berlin (2008)
 25. Hentrich, C., Zdun, U.: A pattern language for process execution and integration design in service-oriented architectures. In: Noble, J., Johnson, R. (eds.) Transactions on Pattern Languages of Programming I, Lecture Notes in Computer Science (LNCS), pp. 136–191. Springer, Berlin (2009)
 26. Hoisl, B., Sobernig, S.: Integrity and confidentiality annotations for service interfaces in SoaML models. In: Proceedings of the International Workshop on Security Aspects of Process-aware Information Systems (SAPAIS2011), pp. 673–679. IEEE (2011)
 27. Hoisl, B., Strembeck, M.: Modeling support for confidentiality and integrity of object flows in activity models. In: Proceedings of the 14th International Conference on Business Information Systems (BIS2011), Lecture Notes in Business Information Processing (LNBIP), pp. 278–289. Springer, Berlin (2011)
 28. Hoisl, B., Strembeck, M.: A UML extension for the model-driven specification of audit rules. In: Proceedings of the 2nd International Workshop on Information Systems Security Engineering (WISSE), Lecture Notes in Business Information Processing (LNBIP). Springer, Berlin (2012)
 29. Huhns, M., Singh, M.: Service-oriented computing: key concepts and principles. *IEEE Internet Comput.* **9**, 75–81 (2005)
 30. Hummer, W., Gaubatz, P., Strembeck, M., Zdun, U., Dustdar, S.: An integrated approach for identity and access management in a SOA context. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT) (2011)
 31. Jensen, M., Feja, S.: A security modeling approach for web-service-based business processes. In: Proceedings of the 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 340–347. IEEE (2009)
 32. Jürjens, J.: UMLsec: extending UML for secure systems development. In: Proceedings of the 5th International Conference on The Unified Modeling Language, pp. 412–425. Springer, Berlin (2002)
 33. Jürjens, J.: Secure Systems Development with UML. Springer, Berlin (2005)
 34. Kim, S., Burger, D., Carrington, D.: An MDA approach towards integrating formal and informal modeling languages. In: Proceedings of the International Symposium of Formal Methods Europe, Lecture Notes in Computer Science (LNCS), vol. 3582, pp. 448–464. Springer, Berlin (2005)
 35. Kopp, O., Martin, D., Wutke, D., Leymann, F.: The difference between graph-based and block-structured business process modelling languages. *Enterp. Model. Inf. Syst.* **4**(1), 3–13 (2009)
 36. Mayer, P.: Model-driven development for service-oriented computing—transformers. <http://mdd4soa.eu/transformers/> (2008)
 37. Mayer, P.: MDD4SOA—model-driven development for service-oriented architectures. PhD thesis, Ludwig Maximilian University of Munich, Faculty of Mathematics, Computer Science and Statistics (2010)
 38. Mayer, P., Koch, N., Schröder, A., Knapp, A.: The UML4SOA profile. <http://www.uml4soa.eu/wp-content/uploads/uml4soa.pdf> (2010)
 39. Mayer, P., Schröder, A., Koch, N.: MDD4SOA: model-driven service orchestration. In: Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference, pp. 203–212. IEEE (2008)
 40. Memon, M., Hafner, M., Breu, R.: SECTISSIMO: a platform-independent framework for security services. In: Proceedings of the Modeling Security Workshop in Association with MODELS 2008 (2008)
 41. Mendling, J., Lassen, K., Zdun, U.: On the transformation of control flow between block-oriented and graph-oriented process modeling languages. *Int. J. Business Process Integr. Manag.* **3**(2), 96–108 (2008)
 42. Mens, T., van Gorp, P.: A taxonomy of model transformation. *Electron. Notes Theor. Comput. Sci.* **152**, 125–142 (2006)
 43. Mishra, S., Weistroffer, H.: A framework for integrating Sarbanes-Oxley compliance into the systems development process. *Commun. Assoc. Inf. Systems (CAIS)* **20**(1), 712–727 (2007)
 44. Nakamura Y., Tatsubori M., Imamura T., Ono K.: Model-driven security based on a web services security architecture. In: Proceedings of the IEEE International Conference on Services Computing, pp. 7–15. IEEE (2005)
 45. National Institute of Standards and Technology (NIST): An Introduction to Computer Security: The NIST Handbook. Special Publication 800–12. <http://csrc.nist.gov/publications/nistpubs/800-12/handbook.pdf> (1995)
 46. National Institute of Standards and Technology (NIST): Data Encryption Standard (DES). Federal Information Processing Standards Publication 46–3. <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf> (1999)
 47. National Institute of Standards and Technology (NIST): Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197. <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf> (2001)
 48. National Institute of Standards and Technology (NIST): Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180–3. http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf (2008)
 49. National Institute of Standards and Technology (NIST): Recommended Security Controls for Federal Information Systems and Organizations. NIST Special Publication 800–53, Revision 3. http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final_updated-errata_05-01-2010.pdf (2009)
 50. National Security Agency (NSA): Information assurance technical framework. <http://handle.dtic.mil/100.2/ADA393328> (2000)
 51. No Magic, Inc.: MacigDraw. <https://www.magicdraw.com> (2012)
 52. Object Management Group: OMG Business Process Model and Notation (BPMN) Specification, Version 2.0, formal/2011-01-03. <http://www.omg.org/spec/BPMN> (2011)
 53. Object Management Group: OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1, formal/2011-08-07. <http://www.omg.org/mof> (2011)
 54. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1, formal/2011-01-01. <http://www.omg.org/spec/QVT> (2011)
 55. Object Management Group: OMG MOF 2 XMI Mapping Specification, Version 2.4.1, formal/2011-08-09. <http://www.omg.org/spec/XMI> (2011)

56. Object Management Group: OMG Object Constraint Language (OCL) Specification, Version 2.2, formal/2010-02-01. <http://www.omg.org/spec/OCL> (2010)
57. Object Management Group: OMG Object Constraint Language (OCL) Specification, Version 2.3.1, formal/2012-01-01. <http://www.omg.org/spec/OCL> (2012)
58. Object Management Group: OMG Service oriented architecture Modeling Language (SoaML) Specification, Version 1.0 Beta 2, ptc/2009-12-09. <http://www.omg.org/spec/SoaML> (2009)
59. Object Management Group: OMG Unified Modeling Language (OMG UML): superstructure, Version 2.4.1, formal/2011-08-06. <http://www.omg.org/spec/UML> (2011)
60. Object Management Group: OMG Unified Modeling Language (OMG UML): infrastructure, Version 2.4.1, formal/2011-08-05. <http://www.omg.org/spec/UML> (2011)
61. Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language, Version 2.0. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf> (2007)
62. Organization for the Advancement of Structured Information Standards (OASIS): Reference Architecture Foundation for Service Oriented Architecture, Version 1.0. <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf> (2009)
63. Organization for the Advancement of Structured Information Standards (OASIS): WS-SecurityPolicy 1.3. <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf> (2009)
64. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: state of the art and research challenges. *IEEE Comput.* **40**, 38–45 (2007)
65. Reznik, J., Ritter, T., Schreiner, R., Lang, U.: Model driven development of security aspects. *Electron. Notes Theo. Comput. Sci.* **163**, 65–79 (2007)
66. Rodríguez, A., Fernández-Medina, E., Trujillo, J., Piattini, M.: Secure business process model specification through a UML 2.0 activity diagram profile. *Decis. Support Syst.* **51**(3), 446–465 (2011)
67. Rodríguez, A., García-Rodríguez de Guzmán, I., Fernández-Medina, E., Piattini, M.: Semi-formal transformation of secure business processes into analysis class and use case models: an MDA approach. *Inform. Softw. Technol.* **52**, 945–971 (2010)
68. Sánchez, Ó., Molina, F., García-Molina, J., Toval, A.: ModelSec: a generative architecture for model-driven security. *J. Univ. Comput. Sci.* **15**(15), 2957–2980 (2009)
69. Sandhu, R.: On five definitions of data integrity. In: Proceedings of the IFIP WG11.3 Working Conference on Database Security VII (1993)
70. Scheer, A.-W.: *ARIS: Business Process Modeling*. Springer, Berlin (2000)
71. Schefer, S., Strembeck, M.: Modeling process-related duties with extended UML activity and interaction diagrams. In: Proceedings of the International Workshop on Flexible Workflows in Distributed Systems, Electronic Communications of the EASST (2011)
72. Schefer, S., Strembeck, M.: Modeling support for delegating roles, tasks, and duties in a process-related RBAC context. In: Proceedings of the International Workshop on Information Systems Security Engineering (WISSE), Lecture Notes in Business Information Processing (LNBIP), vol. 83. Springer, Berlin (2011)
73. Schefer, S., Strembeck, M., Mendling, J.: Checking satisfiability aspects of binding constraints in a business process context. In: Proceedings of the Workshop on Workflow Security Audit and Certification (WfSAC), Lecture Notes in Business Information Processing (LNBIP), vol. 100. Springer, Berlin (2011)
74. Schefer, S., Strembeck, M., Mendling, J., Baumgrass, A.: Detecting and resolving conflicts of mutual-exclusion and binding constraints in a business process context. In: Proceedings of the 19th International Conference on Cooperative Information Systems (CoopIS), Lecture Notes in Computer Science (LNCS), vol. 7044. Springer, Berlin (2011)
75. Schefer-Wenzl, S., Strembeck, M.: An approach for consistent delegation in process-aware information systems. In: Proceedings of the 15th International Conference on Business Information Systems (BIS), Lecture Notes in Business Information Processing (LNBIP). Springer, Berlin (2012)
76. Schefer-Wenzl, S., Strembeck, M.: Modeling context-aware RBAC models for business processes in ubiquitous computing environments. In: Proceedings of the 3rd International Conference on Mobile, Ubiquitous and Intelligent Computing (MUSIC) (2012)
77. Schmidt, D.: Model-driven engineering: guest editor's introduction. *IEEE Comput.* **39**(2), 25–31 (2006)
78. Schmidt, H., Jürjens, J.: Connecting security requirements analysis and secure design using patterns and UMLsec. In: Proceedings of the 23rd International Conference on Advanced Information Systems Engineering (CAiSE), Lecture Notes in Computer Science (LNCS), pp. 367–382. Springer, Berlin (2011)
79. Selic, B.: The pragmatics of model-driven development. *IEEE Softw.* **20**(5), 19–25 (2003)
80. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. *IEEE Softw.* **20**(5), 42–45 (2003)
81. Sobernig, S., Zdun, U.: Invocation assembly lines: patterns of invocation and message processing in object remoting middleware. In: Kelly, A., Weiss, M. (eds.) Proceedings of 14th Annual European Conference on Pattern Languages of Programming (EuroPLoP 2009), CEUR-WS.org, vol. 566. (2009)
82. Stahl, T., Völter, M.: *Model-Driven Software Development*. Wiley, New York (2006)
83. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: *EMF: Eclipse Modeling Framework*. Addison-Wesley, Boston (2008)
84. Strembeck, M., Mendling, J.: Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In: Proceedings of the 18th International Conference on Cooperative Information Systems (CoopIS), Lecture Notes in Computer Science (LNCS), vol. 6426. Springer, Berlin (2010)
85. Strembeck, M., Mendling, J.: Modeling process-related RBAC models with extended UML activity models. *Inform. Softw. Technol.* **53**(5), 456–483 (2011)
86. Tatsubori, M., Imamura, T., Nakamura, Y.: Best-practice patterns and tool support for configuring secure web services messaging. In: Proceedings of the IEEE International Conference on Web Services, pp. 244–251. IEEE (2004)
87. Warner, J., Atluri, V.: Inter-instance authorization constraints for secure workflow management. In: Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT) (2006)
88. Wenzel, S.: CARISMA. <http://vm4a003.itmc.tu-dortmund.de/carisma/web/doku.php> (2012)
89. Wolter, C., Menzel, M., Meinel, C.: Modelling security goals in business processes. In: Modellierung 2008, Lecture Notes in Informatics (LNI), pp. 197–212 (2008)
90. Wolter, C., Menzel, M., Schaad, A., Miseldine, P., Meinel, C.: Model-driven business process security requirement specification. *J. Systems Archit.* **55**(4), 211–223 (2009)
91. Wolter, C., Schaad, A.: Modeling of task-based authorization constraints in BPMN. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) Proceedings of the 5th International Conference on Business Process Management (BPM), volume 4714 of Lecture Notes in Computer Science (LNCS), pp. 64–79. Springer, Berlin (2007)
92. World Wide Web Consortium (W3C): Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl> (2001)
93. World Wide Web Consortium (W3C): Web Services Policy 1.5, Attachment. <http://www.w3.org/TR/ws-policy-attach/> (2007)

94. World Wide Web Consortium (W3C): Web Services Policy 1.5, Framework. <http://www.w3.org/TR/ws-policy/> (2007)
95. Zdun, U.: Patterns of component and language integration. In: Manolescu, D., Völter, M., Noble, J. (eds.) *Pattern Languages of Program Design 5* (2006)
96. Zdun, U., Dustdar, S.: Model-driven and pattern-based integration of process-driven SOA models. *Int. J. Business Process Integr. Manag. (IJBPIIM)* 2(2), 109–119 (2007)
97. Zdun, U., Hentrich, C., Dustdar, S.: Modeling process-driven and service-oriented architectures using patterns and pattern primitives. *ACM Transact. Web* 1(3), 14:1–14:44 (2007)



Stefan Sobernig is a Post-doc Researcher and Lecturer at the New Media Lab, Institute for Information Systems, Vienna University of Economics and Business (WU Vienna). In his research, he works in the fields of feature-oriented software development, domain-specific language engineering, software patterns, and communication middleware.

Author Biographies



Bernhard Hoisl holds a MSc degree in Information Systems from the Vienna University of Economics and Business (WU Vienna) and a MSc degree in Computer Science Management from the Vienna University of Technology. Currently, he works at the New Media Lab, Institute for Information Systems, WU Vienna on a dissertation fellowship project. Since April 2010 he is also part-time researcher at Secure Business Austria (SBA). His PhD research interests are

focused on model-driven software development, domain-specific languages, and process modeling with a special emphasis on the specification of security aspects.



Mark Strembeck is an Associate Professor of Information Systems at the Vienna University of Economics and Business (WU Vienna), Austria. His research interests include secure business systems, business process management, model-driven software development, and the modeling and management of dynamic software systems. He received his doctoral degree as well as his Habilitation degree (*venia docendi*) from WU Vienna. He is a key researcher at Secure Business Austria (SBA), and the Vice Institute Head of the Institute for Information Systems at WU Vienna.