Integrity and Confidentiality Annotations for Service Interfaces in SoaML Models

Bernhard Hoisl[1,2]
[1] *Secure Business Austria Research Center*
Vienna, Austria
bernhard.hoisl@sba-research.org

Stefan Sobernig[2]
[2] *Institute for Information Systems and New Media*
Vienna University of Economics and Business
stefan.sobernig@wu.ac.at

***Abstract*—This paper presents an approach for incorporating data integrity and data confidentiality into the model-driven development (MDD) of process-driven service-oriented architectures (SOAs) based on the OMG SoaML. Specifications for service interfaces are extended by UML activities to model invocation protocols. An invocation protocol makes the control and the object flows between service invocations explicit. Integrity and confidentiality attributes are used to annotate the object flows. The annotations serve for generating security-aware execution artefacts (e.g., interface description documents, deployment descriptors, and middleware configurations). We applied the approach prototypically in a Web Services platform environment (WS-BPEL, WSDL, WS-SecurityPolicy).**

***Keywords*-Service-Oriented Architecture, Security Engineering, UML, Web Services, SoaML, Model-Driven Development**

I. Introduction

A service-oriented architecture (SOA) is an architectural style which organises a software system as a composition of distributed software components, or services. Services require and provide callable functions at announced network endpoints. Processable interface descriptions serve for implementing the service interfaces in consumer and provider applications; independent from the communication middleware, the invocation protocols, and the transport protocols used. Modelling support for systems adhering to the SOA architectural style is provided, e.g., by the UML extension SoaML [1]. In a process-driven SOA, one or more components act as process engines and orchestrate service interactions in order to implement business processes [2].

Business processes executed by a process-driven system involve data assets (e.g., personnel or financial records) requiring protection against, e.g., unauthorized or inadvertent disclosure and modification. The need for enforcing security properties required by data assets (e.g., data confidentiality, data integrity) arises also from compliance requirements with legislation (e.g., privacy laws), with industry regulations (e.g., the Basel II Accord), and with security engineering frameworks (e.g., the SOA Security Compendium). Therefore, representing security properties explicitly in business process models based on EPCs [3], BPMN [4], and UML ([5]–[7]) have been proposed. In an earlier contribution [8], we introduced first-class support for expressing integrity and confidentiality properties of object flows in UML activities.

Nevertheless, business process models including security properties must be integrated with design and implementation models of the process-executing software system. For the scope of this paper, we look at the various invocation data processed for realising interactions among services and process engines: endpoint references, the operation names, as well as input and output parameters [9]. Process execution turns data assets into serialised invocation data exchanged between service endpoints. For example, if data assets in a business process require the integrity property, a modeller must express corresponding message integrity constraints over service interfaces. With this, integrity-verifying code (e.g., message interceptors for fingerprinting and signing) and/or configuration data (e.g., for a security component) can be generated. Such a multi-stage mapping helps maintain consistency with corresponding business process models and contributes to a compliant system implementation.

To facilitate the complex mapping task, model-driven development (MDD) approaches ([3]–[7]) have combined modelling support for security properties in business process models, model integration with structural and behavioural views of a process-driven SOA, and automated generation of execution models (and code). However, many existing MDD approaches fall short in two respects:

Common multi-view meta-model — The support for SOA, process, and security property modelling is not based on a meta-modelling environment providing several modelling views (e.g., a process flow view, a composition view, a data flow view) for reducing the overall mapping complexity [10]. Even if adopting a multi-view environment based on a common meta-model, such as the UML, standard extensions providing or refining these views are not adopted; e.g., the SoaML for the UML.

View accuracy — Security concerns are orthogonal to concerns such as service composition, transport handling etc. [9]. Modelling data integrity and data confidentiality properties should not lead to model interactions violating this separation of concerns. For instance, expressing security properties of invocation data at the level of process execution models (e.g., service activities [11]) bears the risk to interweave process flow and invocation handling details which are otherwise orthogonal to each other.

In this paper, we present an approach for modelling

integrity and confidentiality properties of invocation data for standard SoaML models. For this, we constrain service interfaces through UML activities. These activities specify invocation data dependencies as object flows with integrity and confidentiality annotations [8]. By adopting the standard UML and the SoaML extension, existing tool support can be reused for generating execution models (WS-BPEL, WSDL). For the integration of the UML package `SecureObjectFlows` [8] with the SoaML, we provide both, a UML meta-model extension (`SecureObjectFlows::Services`) and a profile extension (`SOF::Services`), with bi-directional mappings available.[1]

The paper is structured as follows: In Section II, we give an overview of MDD for process-driven SOAs and the UML extensions used. The integration steps necessary for modelling secure flows of invocation data in SoaML are elaborated on in Sections III and IV. We proceed by contrasting our approach with closely related work on model-driven security in Section V. In Section VI, we conclude by summarising our contribution and by pointing to future work.

II. Overview

In relevant MDD approaches ([3]–[7]) computational independent models (CIMs) are considered first (e.g., informal process descriptions, structured architectural descriptions, and security engineering guidelines). In a next step, CIMs are formalised into platform independent models (PIMs), providing structural and behavioural views on the technical services and their process-driven composition. PIMs offer different, yet integrated views to capture a SOA, a process description, and security properties. From these different views, model transformations produce a set of platform-specific models (PSMs).

In our approach, we employ the UML for modelling various PIM views on a process-driven SOA. A SOA's structure (e.g., services, service interfaces) is depicted as a set of SoaML/UML models (see Section II-A). Data dependencies between invocations, including their integrity and confidentiality properties, are defined for a given service interface by secure object flows [8] (see Section II-B). Rule-based PIM-to-PSM translation is achieved by operating on the models' XMI representations. The XMI documents are processed into an intermediate object model, to bridge between the graph-based PIMs and the block-based PSMs [12]. Process-oriented transformation steps are supported by the existing Eclipse-based MDD4SOA plugin [13]. The targeted PSMs are interface descriptions (i.e., WSDL) and process execution descriptions (i.e., WS-BPEL 2.0).

[1]All modelling and implementation artefacts are available from http://nm.wu.ac.at/bhoisl.

For the transformation of the security property elements, we extended the MDD4SOA plugin. Additional transformation steps add WS-SecurityPolicy fragments to the generated interface descriptions and deployment descriptors. In addition, our approach allows for specifying parts of invocation data (e.g., single parameters or message elements) to be annotated. These parts turn into selection expressions over messages in WS-SecurityPolicy descriptions (e.g., `EncryptedElements`, `SignedParts`). The attributes of secure property elements specifying the integrity and confidentiality details map to identifiers for algorithm suites as defined by the WS-SecurityPolicy specification. Due to the space limitations, we do not elaborate further on PSM generation in this paper.

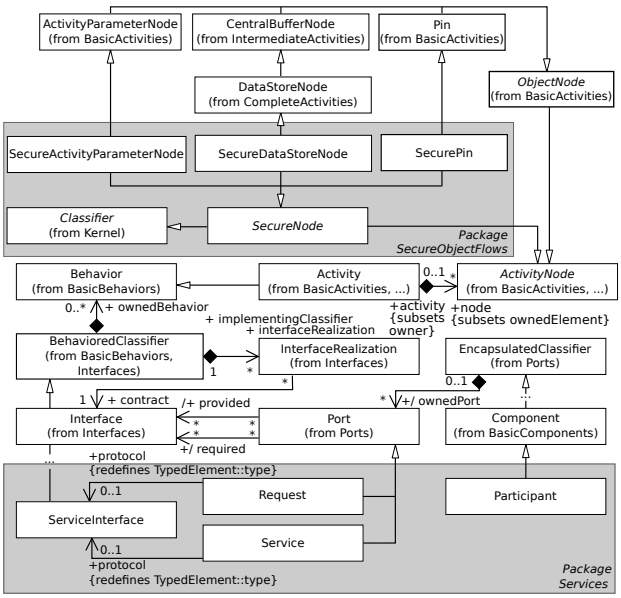

Figure 1: Relevant excerpts from the SoaML and `SecureObjectFlows` meta-models

A. SoaML Concepts

The Service-oriented architecture Modeling Language (SoaML) extends the UML to model SOAs from structural and behavioural viewpoints (see Figure 1). As for the composition of service consumers and providers, the SoaML describes a SOA as a set of interacting components referred to as `Participants`, each announcing interaction capabilities and needs by means of `Service` and `Request` ports, respectively. For this, `Service` and `Request` ports expose required and provided `Interfaces` realising the port's protocol. This port protocol can be further specified by a `ServiceInterface`. `Participants` are connected via their protocol-compliant ports. Protocol compliance is expressed by either sharing a `ServiceInterface` between corresponding `Request` and `Service` ports; or by mating their required and provided `Interfaces` directly.

ServiceInterfaces allow the modeller to express behavioural details of port protocols explicitly. Protocol roles taken by two or more interacting ports for realising a ServiceInterface can be modelled, along with behavioural specifications such as UML activities. An excerpt of a SoaML model is depicted in Figure 3 for later reference.

B. SecureObjectFlows Concepts

The SecureObjectFlows meta-model extends UML activity models with abstract syntax and semantics for modelling data integrity and data confidentiality properties for object flows. Essentially, the SecureObjectFlows package specialises three types of ObjectNodes with integrity and confidentiality properties: SecurePin, SecureDataStoreNode, and SecureActivityParameterNode (see also Figure 1). These specialised nodes inherit all behaviour from their corresponding ObjectNodes and the SecureNode classifier. The abstract SecureNode meta-class allows for specifying additional properties (e.g., cryptographic hash functions, encryption algorithms, encryption key lengths) for its indirect instances and provides model integrity constraints.

III. A Profile for Secure Object Flows

In [8], the SecureObjectFlows package is introduced as a UML meta-model extension to the UML CompleteActivities package. For integration with the SoaML, it is necessary to provide a UML profile variant of the SecureObjectFlows meta-model. This is because, on the one hand, the SoaML is provided both as a meta-model and as a profile, with explicit correspondences defined for them. Both variants are binding compliance points for modellers, CASE tool providers, and extension engineers [1]. On the other hand, a UML profile extension provides immediate advantages. Most importantly, the CASE tool integration available for SoaML can be reused.

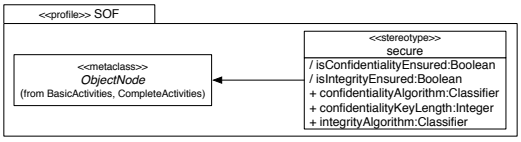

Figure 2: The «secure» stereotype

The profile variant of the SecureObjectFlows package provides a single stereotype «secure» extending the ObjectNode meta-class (see Figure 2). This stereotype provides all integrity and confidentiality attributes available for the SecureNode meta-class. The OCL constraints originally defined for the SecureObjectFlows meta-model were adapted for the context of the «secure» stereotype. The correspondences between the meta-model and the profile are depicted in Table I as instance specifications at the UML level M1.

M1 model (profile extension)			M1 model (meta-model extension)
:Pin	extension_secure / base_ObjectNode	:secure	:SecurePin
:DataStoreNode	extension_secure / base_ObjectNode	:secure	:SecureDataStoreNode
:ActivityParameterNode	extension_secure / base_ObjectNode	:secure	:SecureActivityParameterNode

Table I: Mappings between the SOF profile and the SecureObjectFlows meta-model extension

IV. Secure Interfaces in SoaML

The SecureObjectFlows extension permits modelling the integrity and confidentiality properties as annotations for object nodes in UML activities. We aim at modelling invocation data (e.g., input and output parameters) requiring the integrity and the confidentiality property. In the compositional view of a SoaML model, service invocations are represented by ServiceInterfaces. In Figure 3, AService stipulates the permissible service invocations (e.g., OperationB1) between a Request port (requestor) and a Service port (provider). To describe the behavioural pattern of service invocations between two (or more) ports, a ServiceInterface as a kind of BehavioredClassifier can hold instances of Behavior and, hence, instances of Activities (see the UML excerpt in Figure 1 and AnInvocationProtocol in Figure 3).

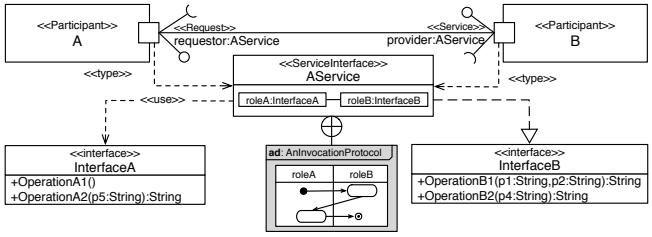

Figure 3: Activities on ServiceInterfaces

The availability of an Activity as owned behaviour of a ServiceInterface provides an important extension point[2] for defining security properties in invocation protocols:

- *Invocation data as object nodes*: An Activity, with its actions denoting invocations, allows for object flows to reflect input and output parameter streams for invocations. This is the major representational prerequisite for

[2]Note that the SoaML specification points to the usage of Activities for modelling *control flows* between operations which are required or provided by Interfaces defined on a ServiceInterface [1].

applying the concepts of the SecureObjectFlows package.

- *Protocol roles*: ActivityPartitions can be used for modelling *protocol roles*. Each ActivityPartition represents an interface-realising role, abstracting from the actual Participants using or implementing the interfaces. Their compositional correspondences are the parts defined for the activity-owning ServiceInterface (see roleA and roleB in Figure 3). ServiceInterfaces may refer to more than two parts (or protocol roles) and can so model multi-directional invocation flows.
- *Duality of invocations*: The roles are typed by the Interfaces required and implemented by the ServiceInterface (i.e., InterfaceA and InterfaceB in Figure 3). With ActivityPartitions denoting these roles, ActivityPartitions reflect the provider and consumer sides of invocations in a single model element. The integrity and confidentiality annotations in the Activity can so capture consumer- and provider-side capabilities (e.g., signature mechanisms).
- *Standalone invocation protocol*: A refined Activity owned by a ServiceInterface and the security properties specified for its object flows are modelled independently from the concrete Participants consuming or implementing the service endpoints. In Figure 3, for instance, AService and so AnInvocationProtocol apply to any pair of Participants, whether process engines or service providers.

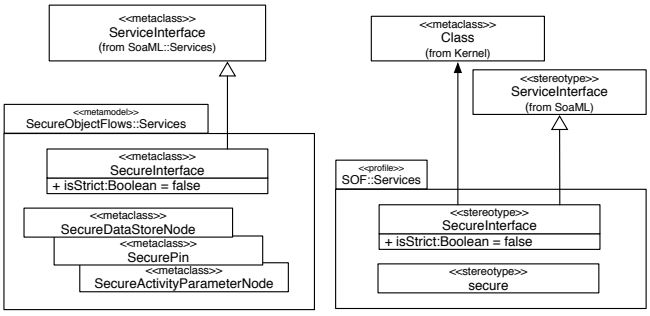

Figure 4: The UML packages for integration

To connect the SecureObjectFlows extension to ServiceInterfaces in SoaML, we provide two UML packages (see Figure 4). The SecureObjectFlows::Services package applies to the SecureObjectFlows meta-model extension. The SOF::Services package glues the SOF profile package and the SoaML profile. In the following, we outline their additions to the abstract syntax and to the semantics of secure object flows.

A. Abstract Syntax

As for the abstract syntax, we introduce a specialised ServiceInterface named SecureInterface. A SecureInterface adds to the ServiceInterface's capabilities by requiring exactly one Activity to be set as owned behaviour. At the SoaML meta-model level, SecureInterface extends the ServiceInterface meta-class. In the profile mapping, the SecureInterface meta-class is represented as a distinct stereotype specialising the ServiceInterface stereotype; and extending the Class meta-class (see Figure 4; see also OCL Constraint 1 in the Appendix).

A SecureInterface contracts either a strict or a permissive mode. In strict mode, all object flows qualifying as invocation data flows (as specified further below) must be annotated. The permissive mode, the default, does not impose a minimum number of secured object flows. The reader is referred to OCL Constraint 2 for the profile realisation of the strict/permissive mode.

B. Constraints

The following constraints impose integrity requirements upon the Activity owned by a SecureInterface. In addition, they add semantics to model flows of invocation data more accurately. For the meta-model integration, the OCL constraints are defined over the meta-classes SecureDataStoreNode, SecurePin, and SecureActivityParameterNode. As for the profile, they apply to the context of the «secure» stereotype. The OCL constraints for the SOF profile package are given as listings in the Appendix.

1) Traceability between Invocations and Interfaces: An Activity may only contain Actions representing Operations owned by the Interfaces implemented or used by a given ServiceInterface. We realise this by requiring all Actions to be instances of CallOperationAction. In addition, all CallOperationActions in an ActivityPartition must link to an Operation of the Interface represented by this ActivityPartition (see OCL Constraint 3).

2) Cross-Interface Invocations only: Depending on the partitioning of an Activity, object flows may occur *within* a single partition or *between* two partitions. In Figure 5, for instance, the object flow between OperationB1 and OperationB2 depicts an output/input dependency between operations owned by the same Interface. Such service invocations are traded within the same Service or Request port and do not travel between two service endpoints (see also Figure 3). We consider such service invocations bypassing most steps of invocation processing and being served within process or machine boundaries [9].

Therefore, we limit the applicability of secured object nodes to cross-interface invocations (see OCL Constraint

4). To distinguish between inner- and cross-interface object flows throughout an invocation protocol activity, all object nodes must be assigned to a single partition (see OCL Constraint 5). According to these constraints, the object flow between `OperationB2` and `OperationA2` in Figure 5 can be annotated, for instance.

3) Activity Parameters for Initial and Intermediary Inbound Data: An invocation protocol activity captures data dependencies between invocations, i.e., output data of one invocation serving as input data for a subsequent invocation. In two important cases, however, input data originates from the outside. These cases are *initial* and *intermediary* inbound data.

Initial inbound data is provided by the consumer triggering the execution of the `Activity` (see e.g. `p1` in Figure 5). Intermediary inbound data is not the result of previous invocations within the same protocol. The input is rather provided from the outside, such as from a process engine holding process control data (see, e.g., `p5`).

For specifying secure object flows, however, it is mandatory to model pairs of secure object nodes [8]. This is because the security properties required at either end of an object flow might deviate from each other. Consider, for instance, two `Participants` providing different confidentiality algorithm suites. In order to model this heterogeneity while maintaining consistency between endpoints, an additional object node as an explicit counterpart must be modelled (see OCL Constraint 6). By using `ActivityParameterNodes` as counterparts, external input and output dependencies of the `Activity` are expressed.

4) Activity Parameters for Intermediary and 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 invocation protocol. For instance, an invocation's output is to be stored as a process-persistent variable by a process engine. If secured, such object nodes require a corresponding object node, e.g., an `ActivityParameterNode` (see OCL Constraint 6).

5) Streaming-only Activity Parameters: As a result of the prior two constraint sets, the `ActivityParameterNodes` used to depict the counterparts of intermediary inbound data (e.g., `AInParam2` in Figure 5) and output data must stand for *streaming* activity parameters (see OCL Constraint 7). Streaming parameters represent data which become available in the context of a given activity, or which leave this context, during execution of the `Activity`. Note that the streaming mode is only mandatory for cases of secure intermediary `Input-` and `OutputPins` (in the sense of OCL Constraint 6). Corresponding object nodes of `Pins` (e.g., `AnInParam1`) held by the first and the last `CallOperationActions` are exempted so that they can model global start and stop conditions for the invocation protocol activity.

6) Same Origin for Input Data Flows: Input data for service invocations, represented by `InputPins` on `CallOperationActions`, must have corresponding object nodes which all reside in the same `ActivityPartition`. By corresponding, we mean the initial source nodes of an object flow. Different partitions as origins for input data for an operation are invalid (see OCL Constraint 8). In Figure 5, for example, we find that the `InputPins` of `OperationA2` have corresponding object nodes in the `roleB` partition.

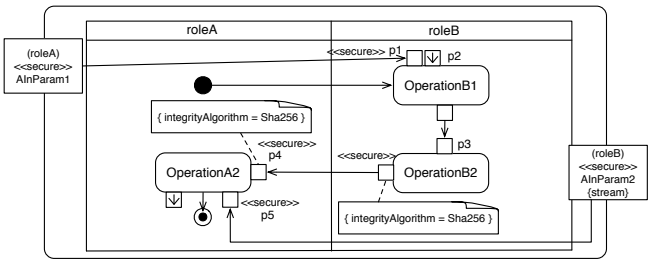

Figure 5: An invocation protocol activity with secure object flows

V. Related Work

In [3], Jensen and Feja extend a proprietary MDD software tool for modelling SOA security properties (access control, data integrity, and confidentiality). The target PSMs are WS-SecurityPolicy specifications. EPCs are used at the business process modelling level. The security model view, while sharing the EPC meta-model, is separate from the process model view. Both are maintained separately and mapped to each other to form an amalgam model. As for view accuracy, the security properties are only captured for the scope of a single process engine (rather than for a collaboration of service partners).

Wolter et al. present in [4] an approach for modelling security goals visually, including model transformations into corresponding security policy specifications. Security-annotated BPMN process models are mapped to WS-BPEL service descriptions and WS-Security policies. As the security models are specified in the UML, model transformations must be applied between the UML and the BPMN process description. A common meta-model for all views is, therefore, not realised. The security properties covered are access control (authentication and authorisation), confidentiality, and integrity at a per-message level.

Another approach is presented by Basin and Doser [5]. Basing the work on UML 1.4 class models, the authors integrate their security modelling language (SecureUML) with a custom defined process language. The SecureUML is extended to adopt an RBAC scheme, with RBAC constraints being expressed over process model instances. The PSM target is code for Java Servlet containers, instrumenting

the container's access control mechanisms. The approach is limited in its extensibility because UML M1 class models define the shared meta-model. The state-transition semantics of the process models only cover a single view on a process-driven system. SOA-related views are not provided at all.

Nakamura et al. describe in [6] a toolkit for generating web services security configurations, covering properties such as authentication, integrity, non-repudiation, and confidentiality. UML class models provide a structural view on a SOA, with stereotypes representing selected security properties. Although using the UML meta-model has potential for adopting existing UML extensions, the authors limit themselves to a custom, ad hoc profile definition. Process views are not considered. The target PSMs are IBM WS-Security specifications.

In [7], Hafner et al. present a model-driven security approach for incorporating security requirements (integrity, confidentiality, and non-repudiation) into PIMs. The UML is used, on the one hand, to model process descriptions as activity diagrams and, on the other hand, to add security annotations to `ObjectNodes` using OCL-like statements. Two orthogonal views are presented: a workflow and an interface view. Although the PIMS are based on a common, i.e., the UML, meta-model, SOA-related extensions are not reused. Secure document flows between two participants can be expressed in the workflow view only, security properties of object flows between service invocations are not covered. From the PIMs, transformations generate BPEL, WSDL, and XACML artefacts.

VI. Concluding

In this paper, we outlined an approach for model-driven security of invocation data in process-driven SOAs. With an extension of SoaML service interfaces based on UML activities, we provide means to model integrity and confidentiality in invocation protocols. We emphasise the reuse of existing modelling extensions (SoaML, SecureObjectFlows), as well as existing MDD software artefacts (MDD4SOA). A UML profile (`SOF::Services`), formally described by a suite of OCL constraints, is available for adoption.

In contrast to likeminded MDD approaches, we provide separate views for security properties of invocation data, on the one hand, and process descriptions, on the other hand. As security requirements are orthogonal to, for instance, service orchestration requirements, these views should not interfere with each other. This separation of concerns is achieved for the UML as a common meta-model.

Future work will focus on integrating the invocation data view with the process flow and business process views. Candidates are *service activities* in UML4SOA [11] and the *process flow models* in [14]. Moreover, we plan to mature and document our adaptations of the Eclipse plugin MDD4SOA; and then place them into the public domain.

References

[1] Object Management Group, "Service oriented architecture Modeling Language (SoaML) – Specification for the UML Profile and Metamodel for Services (UPMS) – Version 1.0, Beta 2," Available at: http://www.omg.org/spec/SoaML/1.0/Beta2/PDF, 2009.

[2] C. Hentrich and U. Zdun, "A Pattern Language for Process Execution and Integration Design in Service-Oriented Architectures," *Transactions on Pattern Languages of Programming*, vol. 1, pp. 136–191, 2009.

[3] M. Jensen and S. Feja, "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*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 340–347.

[4] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel, "Model-Driven Business Process Security Requirement Specification," *Journal of Systems Architecture*, vol. 55, no. 4, pp. 211–223, 2009.

[5] D. Basin, J. Doser, and T. Lodderstedt, "Model Driven Security: From UML Models to Access Control Infrastructures," *ACM Transactions on Software Engineering and Methodology*, vol. 15, pp. 39–91, January 2006.

[6] Y. Nakamura, M. Tatsubori, T. Imamura, and K. Ono, "Model-Driven Security Based on a Web Services Security Architecture," in *Proceedings of the IEEE International Conference on Services Computing*. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 7–15.

[7] M. Hafner, R. Breu, B. Agreiter, and A. Nowak, "SECTET: An Extensible Framework for the Realization of Secure Inter-Organizational Workflows," *Journal of Internet Research*, vol. 16, no. 5, pp. 491–506, 2006.

[8] B. Hoisl and M. Strembeck, "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), Springer, forthcoming.

[9] S. Sobernig and U. Zdun, "Invocation Assembly Lines: Patterns of Invocation and Message Processing in Object Remoting Middleware," in *Proceedings of 14th Annual European Conference on Pattern Languages of Programming (EuroPLoP 2009)*, A. Kelly and M. Weiss, Eds. Irsee, Germany, July 8-12, 2009: CEUR-WS.org (Vol-566), March 2009.

[10] H. Tran, U. Zdun, and S. Dustdar, "VbTrace: Using View-based and Model-driven Development to Support Traceability in Process-driven SOAs," *Software & System Modeling*, vol. 10, no. 1, pp. 5–29, 2009.

[11] P. Mayer, N. Koch, A. Schröder, and A. Knapp, "The UML4SOA Profile," Available at: http://www.uml4soa.eu/wp-content/uploads/uml4soa.pdf, 2010.

[12] J. Mendling, K. B. Lassen, and U. Zdun, "On the Transformation of Control Flow between Block-Oriented and Graph-Oriented Process Modeling Languages," *International Journal of Business Process Integration and Management*, vol. 3, no. 2, pp. 96–108, 2008.

[13] P. Mayer, A. Schroeder, and N. Koch, "MDD4SOA: Model-Driven Service Orchestration," in *Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference*. IEEE Computer Society, 2008, pp. 203–212.

[14] U. Zdun, C. Hentrich, and S. Dustdar, "Modeling Process-Driven and Service-Oriented Architectures Using Patterns and Pattern Primitives," *ACM Trans. Web*, vol. 1, no. 3, pp. 23–50, September 2007.

APPENDIX

CONSTRAINTS FOR THE SOF::SERVICES PROFILE

The following OCL expressions are specific to the Eclipse 3.6.2 MDT/OCL engine.

OCL Constraint 1: A `SecureInterface` must own an `Activity` instance as its owned behaviour.

```
context SOF::Services::SecureInterface
inv:    self.base_Class.ownedBehavior->one(oclIsKindOf(Activity))
```

OCL Constraint 2: In strict mode all cross-interface object flows must be secured.

```
context SOF::Services::SecureInterface
def:    allPredecessors(objNode : ActivityNode) : Set(ActivityNode) =
        objNode.incoming.source->collect(x |
          allPredecessors(x))->asSet()->union(objNode.incoming.source)
inv:    self.isStrict implies
          self.base_Class.ownedBehavior.oclAsType(Activity).node->select(
          oclIsKindOf(ObjectNode))->forAll(objNode |
            allPredecessors(objNode)->select(incoming->isEmpty())->forAll(s |
              s.inPartition <> objNode.inPartition implies
                s.getAppliedStereotype('SOF::Services::secure') <> null and
                objNode.getAppliedStereotype('SOF::Services::secure') <> null))
```

OCL Constraint 3: 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.

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

OCL Constraint 4: Only corresponding object nodes residing in different partitions may be tagged by the «`secure`» stereotype.

```
context SOF::Services::secure
def:    allPredecessors(objNode : ActivityNode) : Set(ActivityNode) =
        objNode.incoming.source->collect(x | allPredecessors(x))->asSet()->
        union(objNode.incoming.source)
inv:    allPredecessors(self.base_ObjectNode)->select(
          incoming->isEmpty() and
          oclIsKindOf(ObjectNode) and
          getAppliedStereotype('SOF::Services::secure') <> null)->forAll(s |
            s.inPartition <> self.inPartition)
```

OCL Constraint 5: All activity nodes must be assigned to and must be contained by exactly one and only one activity partition.

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

OCL Constraint 6: All secured `InputPins` must have an incoming object flow; all secured `OutputPins` must have an outgoing object flow.

```
context SOF::Services::secure
inv:    self.base_ObjectNode.oclIsKindOf(InputPin) implies
          self.base_ObjectNode.incoming->notEmpty()
inv:    self.base_ObjectNode.oclIsKindOf(OutputPin) implies
          self.base_ObjectNode.outgoing->notEmpty()
```

OCL Constraint 7: All `ActivityParameterNodes` which are not initial or final nodes in a control and data flow but counterparts of intermediary `Input-` and `OutputPins` must refer to a streaming `Parameter`.

```
context SOF::Services::SecureInterface
def:    isFirstNode(a : ActivityNode) : Boolean =
        a.owner.oclAsType(Activity).node->select(
          oclIsKindOf(InitialNode))->exists(outgoing.target->any(true) = a) or
        a.owner.oclAsType(Activity).node->select(
          oclIsKindOf(ActivityNode) and incoming->isEmpty())->includes(a)
def:    isLastNode(a : ActivityNode) : Boolean =
        a.owner.oclAsType(Activity).node->select(
          oclIsKindOf(ActivityFinalNode))->exists(
          incoming.source->any(true) = a) or
        a.owner.oclAsType(Activity).node->select(
          oclIsKindOf(ActivityNode) and outgoing->isEmpty())->includes(a)
def:    allSuccessors(objNode : ActivityNode) : Set(ActivityNode) =
        objNode.outgoing.target->collect(x |
          allSuccessors(x))->asSet()->union(objNode.outgoing.target)
inv:    self.base_Class.ownedBehavior.oclAsType(Activity).node->select(
        oclIsKindOf(ActivityNode))->forAll(an |
          (not isFirstNode(an) implies
            an.input->forAll(ipin |
              allPredecessors(ipin)->select(
              oclIsKindOf(ActivityParameterNode))->forAll(
                oclAsType(ActivityParameterNode).parameter.isStream))) and
          (not isLastNode(an) implies
            an.output->forAll(opin |
              allSuccessors(opin)->select(
              oclIsKindOf(ActivityParameterNode))->forAll(
                oclAsType(ActivityParameterNode).parameter.isStream))))
```

OCL Constraint 8: All source object nodes of a set of `InputPins` owned by a `CallOperationAction` must be assigned to the same activity partition.

```
context SOF::Services::secure
inv:    self.base_ObjectNode.oclIsKindOf(InputPin) implies
          self.base_ObjectNode.oclAsType(InputPin).owner.oclAsType(
          CallOperationAction).input->forAll(ipin |
            allPredecessors(ipin)->select(
              incoming->isEmpty() and
              oclIsKindOf(ObjectNode) and
              getAppliedStereotype('SOF::Services::secure') <> null)->forAll(
                inPartition = self.inPartition))
```