From Secure Business Process Models to Secure Artifact-Centric Specifications

Mattia Salnitri[1], Achim D. Brucker[2], and Paolo Giorgini[1]

[1] University of Trento, Trento, Italy
{mattia.salnitri, paolo.giorgini}@unitn.it
[2] SAP SE, Karlsruhe, Germany
achim.brucker@sap.com

Abstract. Making today's systems secure is an extremely difficult and challenging problem. Socio and technical issues interplay and contribute in creating vulnerabilities that cannot be easily prevented without a comprehensive engineering method. This paper presents a novel approach to support process-aware secure systems modeling and automated generation of secure artifact-centric implementations. It combines social and technical perspectives in developing secure systems. This work is the result of an academic and industrial collaboration, where SecBPMN2, a research prototype, has been integrated with SAP River, an industrial artifact-centric language.

1 Introduction

Today's systems are more and more similar to complex organizations, where autonomous and independent components interact one another to achieve common and local objectives. An air traffic management system is, for instance, composed of several autonomous elements, such as the communication service provider network, the tower control, the meteorological services provider, the Very High Frequency (VHF) network, the ground management system, and so forth. Some of them can be considered as pure technical components (e.g., satellite communication network or the aircraft router) while others are human/social elements (e.g., the controllers in the control tower, or airport rescue team). In other words, socio and technical elements are components of the same Socio-Technical System (STS) where they interact as autonomous elements.

STSs can easily become complex and hard to control systems, where human factors may introduce an high level of unpredictability. To regulate the system's interactions, process modeling languages are commonly used to design the flow of activities and to prescribe roles and responsibilities. Business Process Management and Notation (BPMN) 2.0 [1] and Business Process Execution Language (BPEL) [2] are well known examples of process-centric modeling languages. The design of a STS cannot leave out, however, the artifacts (entities, data and documents) that are used, consumed and shared within the system. SAP River [3] and Oracle PeopleCode [4] are largely used artifact-centric approaches to model business artifacts and their business logic.

K. Gaaloul et al. (Eds.): BPMDS 2015 and EMMSAD 2015, LNBIP 214, pp. 246–262, 2015.

In STS, security is not exclusively a technical problem, very often it is the combination of socio and technical factors that gives origin to the most critical vulnerabilities of a system [5]. To guarantee desirable levels of security, artifact-centric approaches offer, for example, access control and authentication security controls to constraint the access to the data and related executable functions (business logic) [6]. However, the security strategies that are beyond the usage of such security controls, should be consistent with the security choices adopted in the business process model; namely, any security strategy adopted for the STS should be first implemented into the business process and then, as consequence, coded at level of artifacts. For example, in a payment engine (e.g., SAP Payment Engine [7]), security choices of creating a process that maintains the integrity of the invoice or ensuring the confidentiality of the credit card information, should be enforced on related business artifacts (e.g., implementing authentication controls for accessing the credit card data).

The literature offers a number of process-centric languages for modeling security concerns along the activities' flow of a system. SecBPMN [8] and SecureBPMN [9] are two examples of modeling languages where specific annotations are introduced to extend BPMN with security concepts. However, no approach has been proposed so far to handle with security as a global concern across process-centric and artifact-centric dimensions. For example, SAP proposes SAP River [10] and ABAP [11] as artifact-centric languages without any related support for modeling security at process level.

In this paper, we present an approach to deal with security that combines the advantages of business process modeling with the advantages of artifact development. We implemented our approach using SecBPMN2 [12] as a process-centric modeling language to define the business processes and the security choices and SAP River platform for the artifact-based implementation. The overall approach guarantees that the artifact-based implementation complies to the high-level security-aware process specification.

In more detail, our contributions are three-fold: first, we present an integrated approach for modeling and implementing secure process-aware socio-technical systems. Second, we present a mapping from control-flow-centric business process models to artifact-centric implementations that include the translation of security and compliance properties. And, third, we implemented our approach on an industrial platform.

2 Baseline

In this section, we introduce the foundations of our work: the security aware, process-centric modeling language SecBPMN2 and the artifact-centric framework SAP River.

2.1 SecBPMN2

Among various process-centric modeling languages, SecBPMN2 stands out for its expressiveness, and the possibility to model both business processes and se-

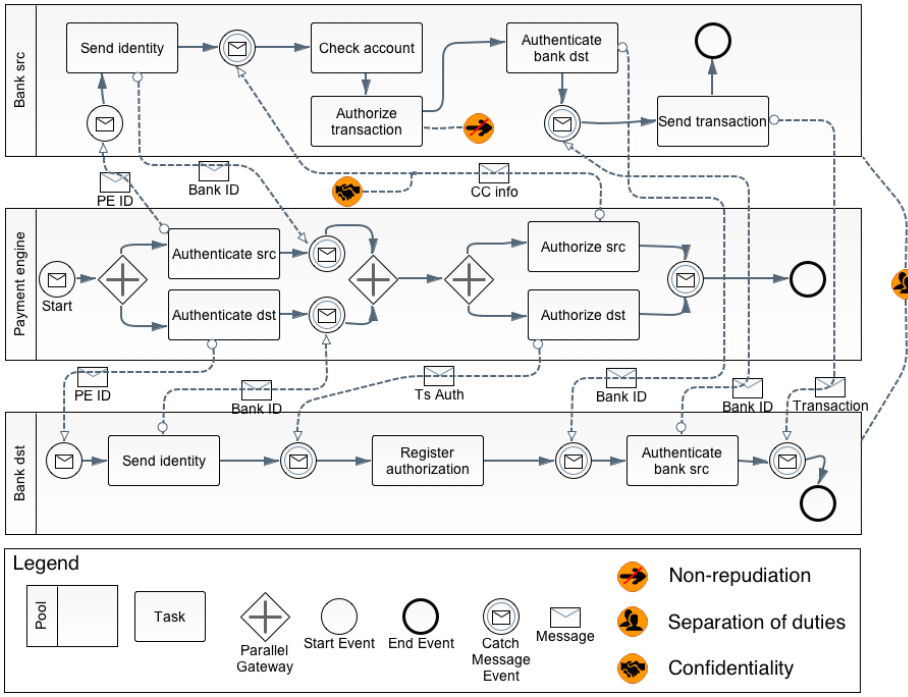


Fig. 1. Example of SecBPMN2-ml model

curity policies. It is composed of SecBPMN2 - modelling language (SecBPMN2-ml), a modeling language that extends BPMN 2.0with security concepts, and SecBPMN2-Query (SecBPMN2-Q), a modeling language for security policies. Figure 1 shows an example of SecBPMN2 concerning the SAP Payment Engine (PE). SAP Payment Engine [7] is a flexible single-payment platform aimed for processing payments into one central hub. It can be used as a single entry point where the company orders/receives payments. It interacts with any bank it is required to be connected using the proper interface and security level required by the bank. The process in Figure 1 starts when a money transfer is executed between two banks.

SecBPMN2 extends BPMN 2.0 adding security choices, represented as eleven security annotations: accountability, auditability, authenticity, availability, integrity, privacy, binding of duties, non-delegation, non-repudiation, separation of duties, and confidentiality. In Figure 1, we have: *non-repudiation*, linked with "Authorize transaction", specifies that "Bank src" cannot be able to deny the execution of that task; *separation of duties* specifies that "Bank src" and "Bank dst" cannot be the same bank; *confidentiality*, linked to the message flow that transmits the "CC info", specifies that only the authorized receivers can read the message. More details can be found in [12].

```
1 @OData
2 type LocalDate { date : UTCTimeStamp; state: String; }
3 application PizzaCloud.SalesApp  {
4     role Approver;
5     export entity SalesOrder accessible by Approver {
6         key element ID : String;
7         element transactionDate : LocalDate;
8         element items : association[0..*] to SalesOrderItems via backlink order;
9     action approveOrder() { [..] } } }
```

Listing 1. A Simple River Example: Modeling a Sales Order

2.2 Artifact-Centric Business Process Modeling

Well known business-process or workflow modeling languages such as BPMN 2.0 or BPEL are based on activity flows: data that is processed within the processes is often an afterthought. In contrast, artifact-centric business process modeling [10, 13] puts the business artifacts (e.g., data, documents) into the center of the process modeling.

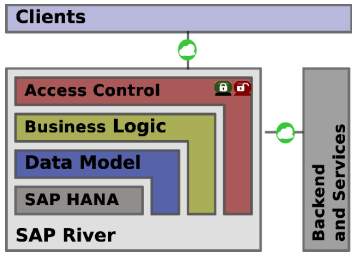


Fig. 2. The SAP River Platform

For our prototype, we use the SAP River, a framework for developing business applications on top of the SAP HANA. Figure 2 shows an high-level overview of SAP River: SAP HANA provides the persistency layer as well as the container in which the enterprise applications are executed. Clients as well as back end systems or external services can communicate with the River platform using standard protocols. The business artifacts (i.e., the data model) and their behavior (i.e., the business logic) as well as the access control are specified in the River Definition Language (RDL) [3].

RDL is an executable specification language that allows to specify declaratively the artifacts (e.g., entities), the relations between them (e.g., associations) as well as the business logic (e.g., actions) on the artifacts. Lst. 1 illustrates an excerpt of a SAP River application. Most importantly, RDL allows for specifying entities (e.g., `SalesOrder`, their attributes (e.g., `transactionDate`), custom types of the attributes (e.g., `LocalDate`), and the relation (associations) to other entities. For example, the entity `SalesOrder` as a bidirectional association (similar to associations in UML [14]) to the entity `SaledOrderItems`. Besides this pure data modeling, RDL also supports to specify the actions (e.g., `approveOrder` in a declarative style. Finally, RDL supports to specify role-based access control restrictions: the actions of the entity `SalesOrder` are only accessible by members of the role `Approver`.

By default, artifacts in RDL are private. To enable access outside of their scope, they need to explicitly marked with the **export** keyword. Moreover, the annotation `@OData` enables remote access using the OData protocol (`www.odata.`

org). Such a remote access is controlled by the same access control restrictions of internal access.

3 From Procedural Specification to Business Artifact Specification

Figure 3 provides an overview of our approach: Step 1 consists in defining the procedural specification, where a team composed of domain experts (i.e., the customers and consultants) and security engineers work together to express security needs over the activities' flow of the system. The procedural model is then used to automatically generate a set of River application skeletons (step 2) that will be used by developers to implement the business logic (step 3). Whenever the system changes (e.g., to adapt to new organizational processes or new legislations), a revised version of the SecBPMN2 processes or new specifications for the artifacts can be introduced. Compliance between SecBPMN2 models and artifact specifications has to be checked again and new changes on the process models or artifact specifications are introduced (step 4).

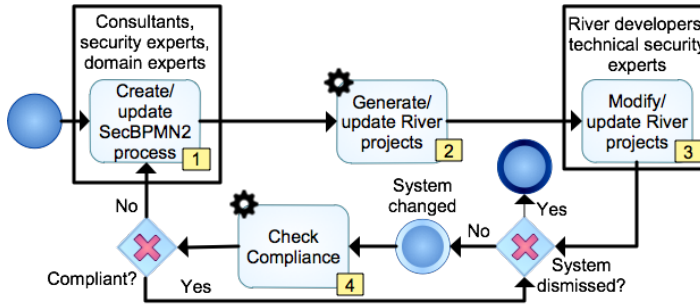


Fig. 3. The process proposed with the approach

The generation of River applications skeleton from SecBPMN2 models requires extra information that are specific of business artifacts and that are not represented in the process models. Particularly, SecBPMN2 data objects are complex data structure with heterogeneous data types in River; for instance, in the procedural specification in Figure 1, the credit card data object is specified only by its name, while in River it will be specified as complex business artifact with information such as the number of the credit card, Card Verification Value (CVV), name of the owner, or the issue date. Companies such as SAP or Oracle have created repositories of templates, that can be reused whenever the same business artifacts are requested. Our approach includes such repositories, so to support the generation of River applications that already incorporate a complete definition of business artifacts and therefore to reduce the amount of work required to River developers to customize the River application once generated. Developers will, then, complete the specification of the business artifacts

and their business logic with domain dependent details. For example, in PE, the business logic related to the credit card strictly depends on the context in which the system applies.

Moreover, since River applications can enforce only SecBPMN2 security choices regarding information handled in the process, they cannot be the only enforcement point for all security choices defined in the procedural specification. We include, as an output of the approach, a "security specification" document which contains a list of the security controls that cannot be implemented in River applications.

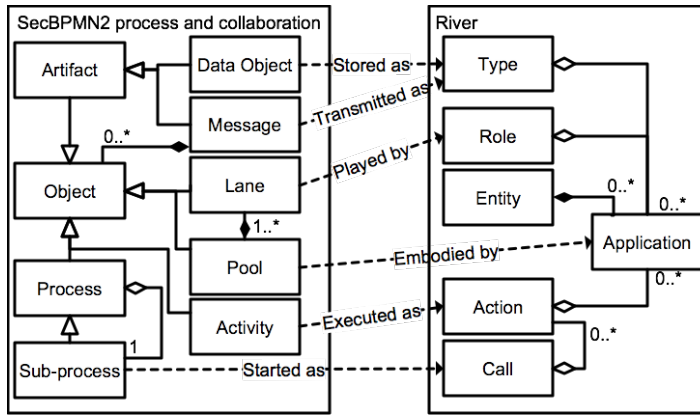


Fig. 4. Part of meta-models of SecBPMN2 and River and the mapping relations

Figure 4 shows the mapping relations between part of SecBPMN2 and River meta-models. The six mapping relations are described below.

Stored as "Data object", which represents a set of information, is "stored as" a River "Type", which represents the structure of the information of an element in entity.

Transmitted as "Message", that represents a set of information sent between pools, is "transmitted as" a River "Type", which represents the structure of a message sent.

Embodied by "Pool", which defines a company or an actor such as a buyer or a manufacturer, is "embodied by" an "Application", that represents a set of business artifacts, which can be accessed only using the APIs, and their business logic. From an artifact-centric perspective, a pool and a River application are use to identify organizations or a well defined parts of them.

Executed as A "Task" represents an operation performed by a participant. "Action" represents the business logic linked to a data structure, i.e., they are the operations executed to set/maintain some properties of the business objects. The operations represented by a task are implemented and executed in a River action.

```
application paymentEngine
{
        role controller;
        role validator;
        type VCNUM {
                element ClientIDs : Integer;
                element PaymentCardType : String;
                element CardNumber : String;
                [..]}
        export entity VCNUMEntity {
                key element id: Integer;
                element VCNUMData: VCNUMType; }
        export namespace VCNUMnamespace accessible by sap.hana.All {
                export action ValidateCreditCard() {}
                export action FilterSensitiveInformation() {} }  }
```

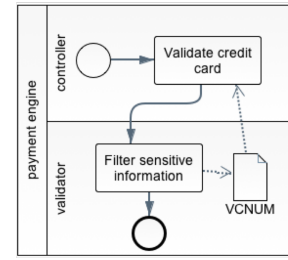


Fig. 5. A SecBPMN2 model representing part of a PE business process

Played by A "Lane" represents a participant, i.e., a person, a service or a set of them. A"Role" represents a set of authorizations assigned to a (set of) physical entity(ies), i.e., the it represents any entity that can receive an authorization. A lane is "played by" a River role because any entity represented by a role can perform all the actions required by the tasks in the lane.

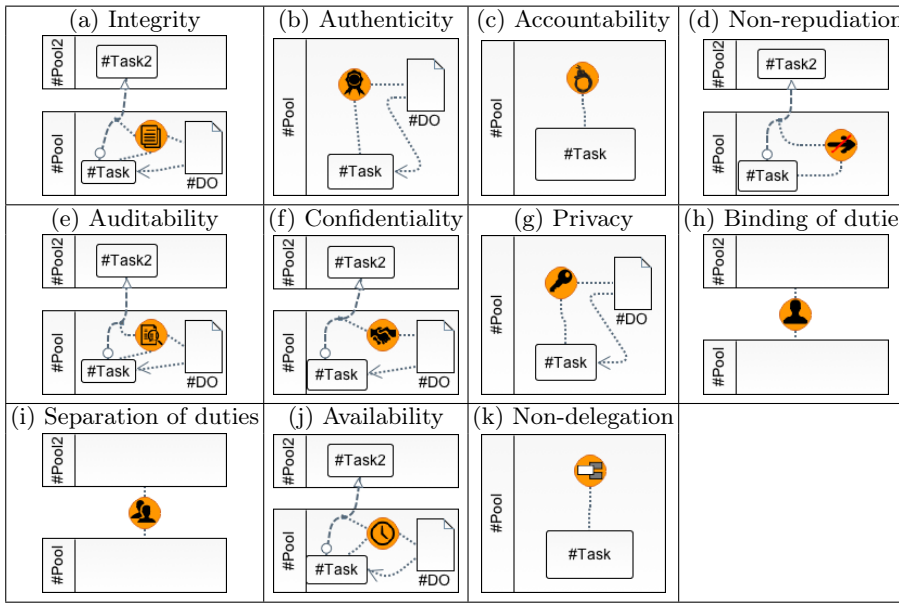
Started as A "Sub-process" is a task that encapsulates a business process, which contains a whole new set of SecBPMN2 elements. Similarly a "Call" is a reference to another River application, which, in turn, contains a whole set of business artifacts. A sub-process is "started as" a call because a call starts a new River application.

The creation of River skeletons is based on generation rules that follow the mapping relations defined in Figure 4. Events and gateways elements are, however, not part of the model transformation since they are used to define the control flow. The main generation rule specifies that a River entity and a River type are generated for each data object and for each message in the SecBPMN2 model. Each entity contains one element of the type generated together with the entity. Each task linked to the data object is transformed in an action, that is placed in an ad-hoc namespace, created for the data object (see Sec. 5).

Figure 5 shows an example of generation. The name of the application reflects the name of the pool and two roles, which correspond to the lanes in the business process, are specified. From the data object "VCNUM" are generated: (i) a type "VCNUM" that contains the structure of the data that makes tangible the information; (ii) an entity "VCNUMEntity" that contains the actual information; (iii) a namescape "VCNUMnamespace" that contains all the actions derived from the SecBPMN2 tasks linked to the "VCNUM" data object. The structure of "VCNUM" is retrieved from the SAP repository, indeed "VCNUM" is a template for the information related to credit cards.

The fourth step of process described in Figure 3 consists in checking if the security requirements are satisfied in SecBPMN2 models and River applications. The former control can be performed using SecBPMN2 verification engine, while the latter may be perform using software verification techniques.

Table 1. SecBPMN2 security annotations



4 Security Enforcement Rules

In this section, we present a set of rules that are used to enforce the security choices of SecBPMN2 into River applications. Tab. 1 shows SecBPMN2 security annotations. To simplify, security annotations shown in the table are linked to all elements they can be linked to; however, SecBPMN2 allows for only one link (except for separation of duties and binding of duties, that can be linked to two pools). In this paper we report only the enforcement rules for process and collaboration models, while details about choreographies can be found in [12]. For each SecBPMN2 security annotation, we briefly describe in the following its meaning and the corresponding Enforcement Rule (ER).

ER1: Integrity. It requires a system to ensure completeness, accuracy and absence of unauthorized modifications in all its components [8]. It can be linked to one task, data object or message flow (Tab. 1-a). Although, it can be partially enforced by filtering the users who can access the River entities (i.e. using authentication and access control), backup mechanisms should be used to avoid loosing potentially precious information (when linked to a data object), or loosing functionalities offered by the system (when linked to the message flow or to an activity). Since, such configurations cannot be specified in a River application, they are enlisted in the security specification document.

ER2: Authenticity. It is defined as the ability of a system to verify identity and to establish trust in a third party and in information it provides [8]. It can be connected to one task or data object (Tab. 1-b). When it is linked to #Task, it

```
1  application #Pool {
2    entity repositorySignatures {
3      signature : String;
4      user : String; }
5    export entity SignatureLogs {
6      element signature: Association to repositorySignatures;
7      element date: UTCTimestamp;
8      element #DO : Association to #DO; }
9    export namespace #DOnamespace accessible by sap.hana.All {
10     export action #Task() {
11       let newSignatureLogs : SignatureLogs = SignatureLogs{
12         date : sap.hana.utils.dateTime.currentUTCTimestamp(),
13         signature : SELECT ONE repositorySignatures FROM repositorySignatures
14                     WHERE user = sap.hana.services.session.getUserName(), #DO : this };
15         Add newSignatureLogs to SignatureLogs; } } }
```

Listing 2. Enforcement of accountability, implementing signature security control

can be enforced with an authenticity security control that verifies the identities of users who execute the action generated from #Task. When it is linked to #DO, an element which contains the hash of the type generated from the #DO will be included in the type itself.

ER3: Accountability. It is defined as the ability of a system to hold users responsible for their actions (e.g., misuse of information) [8]. It can be linked to a task, as shown in Tab. 1-c. It is enforced using the signature security control, which stores the private key of the user who performs the action generated from to #Task. We used private key to unequivocally identify users that performed the action.

Lst. 2 shows part of the River template used to generate the River code for the signature security control. A River template is a piece of River application with placeholders, marked with a "#", that are substituted with an appropriate string. For example, in Lst. 2, #Pool will be substituted with the name of the pool in SecBPMN2. The entity in lines 2–4 contains the private keys associated to the users, while the entity defined in lines 5–8 stores the link to the signature of the user who executed the action, the date in which the action is performed and the link to the entity that contains the action performed. Lines 10–15 show how the signature security control is implemented in each action generated from #Task: a new entry is inserted in the entity "SignatureLogs".

ER4: Non-repudiation. It is defined as the ability of a system to prove (with legal validity) occurrence/non-occurrence of an event or participation/non-participation of a party in an event [8]. It can be connected to one activity or one message flow (Tab. 1-d). We use the signature security control to enforce the non-repudiation security choice. If the security annotation is linked to a message flow, every time send and receive actions are executed, the information about the execution is inserted in the signature entity. If the security annotation is linked to #Task, the information is inserted whenever the action, generated from #Task, is executed.

```
1  application #Pool {
2      type ActionType : enum { READ; WRITTEN; SENT; RECEIVED; }
3      export entity ActionLogs#DO {
4          element date: UTCTimestamp;
5          element actionType : ActionType;
6          element user : String; }          [..]
7      export namespace #DOnamespace accessible by sap.hana.All {
8          export action get#DO(idEntity: Integer) : #Pool.#DOType { [..]
9              let log:ActionLogs#DO = ActionLogs#DO{
10                 user : sap.hana.services.session.getUserName(),
11                 actionType : ActionType.READ,
12                 date : sap.hana.utils.dateTime.currentUTCTimestamp() };
13             log.save(); }
14         export action #Task() { [..] } } }
```

Listing 3. Enforcement of auditability, implementing logging security control

ER5: Auditability. It is defined as the ability of a system to conduct persistent, non-by-passable monitoring of all actions performed by humans or machines within the system [8]. It can be linked to one task, data object or message flow (Tab. 1-e). It is enforced with the logging security control, which stores information of the actions performed. If the security annotation is linked to #DO, all actions in the entity that contains the type generated from #DO are logged; if it is linked to #Task, only the calls to the action that is generated from #Task are stored; if it is linked to a message flow only the actions send/receive, generated from the message flow, are stored.

Lst. 3 shows part of the River template for the logging security control. The type "ActionType" (line 2) defines the type of actions. The entity in lines 3–6 contains: type of the action, date of execution and user who performed the action. Lines 9–13 show how the information about the execution of an action is stored in the entity actionLog#DO. If the security annotation is linked to #DO, information about the execution is inserted in actionLog#DO every time an action, defined in an entity that contains the type generated from #DO, is performed; if the security annotation is linked to #Task then the information is stored every time the action, generated from #Task, is performed; if the security annotation is linked to the message flow, then the information is stored every time the send and receive actions, generated from the message flow, are executed.

ER6: Confidentiality. It requires a system to ensure that only authorized users access information [8]. It is a security annotation that is linked to one message flow or one data object (Tab. 1-f). We enforced it using authentication, access control and implementing encryption security control.

Lst. 4 shows part of the River template for the encryption security control. In lines 5 and 6 the encryption and decryption functions are defined. For the sake of brevity, the algorithms used to encrypt and decrypt data are not shown. Lines 8–11 show how the functions are used to enforce confidentiality: the content of the entity/message is decrypted when retrieved/received and encrypted when is stored/sent. Therefore, the content of entity/message will be visible only in the River application. The encryption/decryption functions are inserted in the

```
1  application #Pool {
2      type #DO {[..]}
3      export entity #DOEntity {
4          [..]
5          action encrypt(data :#DO): #DO { [ENCRYPTION ALGORITHM] }
6          action decrypt(data :#DO): #DO { [DECRYPT ALGORITHM] } }
7      export namespace #DOnamespace accessible by  sap.hana.All {
8          export action get#DO(idEnt: Integer) : #DOEntity {
9              let #DOInst: #DOEntity = SELECT * FROM #DOEntity WHERE id == idEnt;
10             #DOInst.#DOData = #DOInst.decrypt(#DOInst.#DOData);
11             return #DOInst.#DOData; } } }
```

Listing 4. Enforcement of confidentiality, implementing encryption security control

send/receive actions when the security annotation is linked to a message flow, while are inserted in getters and setters of the entity which contains the type generated from #DO.

ER7: Privacy. It requires a system to obey privacy legislation and it should enable individuals to control, where feasible, their personal information (user-involvement) [8]. It is linked to one message flow or one data object (Tab. 1-g). With authentication and access control, we restrict the access to authorized users. We further enforce it, encrypting the content of the entity that contains the type generated from #DO (when the security annotation is connected to #DO), or encrypting the entities sent and received (when the security annotation is linked to a message flow).

ER8: Binding of duties. It requires the same person to be responsible for the completion of a set of tasks [15]. It is linked to two pools (Tab. 1-h). It is enforced using authentication and access-control, ad-hoc security controls. Lst. 5 shows the template for the enforcement of binding of duties. Element "BoDUser" in line 3 contains the first user who accesses the entity and, therefore, the only one that is authorized to access the entity(s) contained in the River applications generated from #Pool and #Pool2. In lines 4–14 the function "CheckBoD" is defined: it checks if the variable "BoDUser" is set locally and in the application generated from #Pool2. If the variable is not set, it sets the variable both local and remotely, otherwise it checks if the user who is executing the action in which the "CheckBoD" method is called, is the same as the one memorized in the variable. The "CheckBoD" method will be called in any action of the entities contained in the applications generated from #Pool and #Pool2 (lines 16–19).

ER9: Separation of duties. It requires two or more different people to be responsible for the completion of a set of tasks [16]. It is linked to two pools (Tab. 1-i). Static separation of duties [17] is enforced using authentication and access control, while dynamic separation of duties [17] is enforced with authentication, access control and ad-hoc security controls. The template for enforcing dynamic separation of duty is similar to the one for binding of duty (Lst. 5).

ER10: Availability. It requires a system to ensure that all its components are available and operational when they are required by authorized users [8]. Tab. 1-j

```
application #Pool{ [..]
  export entity #DOEntity {
    element BoDUser : String;
    action checkBoD(): Boolean {
      if (BodUser is null && getBodUser('#poolURL',id) is null) {
        BodUser = sap.hana.services.session.getUserName();
        setBodUser('#poolURL', id, sap.hana.services.session.getUserName());
        return true; }
      else
        if (BodUser == sap.hana.services.session.getUserName() &&
          getBodUser('#poolURL',id) == sap.hana.services.session.getUserName())
          return false;
        else
          return true; } }
  export namespace #DOnamespace accessible by  sap.hana.All {
    export action get#DO(idEnt: Integer) : #DOEntity {
      let #DOInst: #DOEntity = SELECT * FROM #DOEntity WHERE id == idEnt;
      if (!#DOInst.checkBoD())  return null;
        return #DOInst.#DOData; }
    export action setBoDUser(urlPool: String, idEnt: String, BodUser: String){[..]}
    export action getBoDUser(urlPool: String, idEnt: String): String{[..]}
} }
```

Listing 5. Implementation of dynamic binding of duties

shows its graphical representation. It cannot be enforced in a River application because it requires configuration of the system (e.g., the configuration of the data-base management system), so the specification of using backup mechanism for #DO will be added to the security specification document.

ER11: Non-delegation. It requires the system to ensure that the actions are performed only by indicated actor(s). It can be linked to one task (Tab. 1-k). It is enforced using access control: when #Task is transformed in an action in River, it is executed by the roles authorized to access the tenancy/server where the River application is deployed. Once the action is implemented, it will not be anymore delegated to a third party.

5 Implementation and Evaluation

5.1 Prototypical Implementation

Our prototype (available from [12]) tool takes as inputs an XML specification of the SecBPMN2 model, a repository of business artifact definitions and, optionally, a set of enforcement rules. Using the generation rules described earlier, the prototype generates the River skeletons form templates using Freemarker (`http://freemarker.org`): a Java template library.

Alg. 1 shows the generation of the River skeletons. It follows the generation and enforcement rules described in Sec. 3 and 4. It uses the function GENERATE that retrieves the Freemarker templates and instantiate them using the information contained in the SecBPMN2 model. For each pool of the SecBPMN2 model (line 1), the algorithm creates a new River application (line 2) and it adds, to the application generated, all roles generated from all lanes contained

in the pool (lines 3–5). For each data object, it creates a River type, entity and namespace, and add them to the application (lines 6–10). After that, for each task in the pool, it generates the corresponding action and adds it to the entity(ies) that is(are) generated from the data object that is linked to the task (lines 11–16). The RETRIEVE function checks for this link. If no data object is linked the task, the generated action is added to the application. The last part of the algorithm is for the enforcement of the security annotations: for each security annotation in the pool, GENERATESC instantiates the Freemarker template for the corresponding security control(s) and after that GERNERATESP generates the security specifications that are added to the security specification document.

Algorithm 1 Algorithm for generation of River applications

```
GENERATERIVERAPPLICATIONS(SecBPMN2 model)
 1  for each pool ∈ model
 2  do riverApplication ← GENERATE(pool)
 3     for each lane ∈ pool
 4     do riverRole ← GENERATE(lane)
 5        riverApplication.ADD(riverRole)
 6     for each dataObject ∈ pool
 7     do riverType, riverEntity, riverNamespace ← GENERATE(dataObject)
 8        riverApplication.ADD(riverType)
 9        riverApplication.ADD(riverEntity)
10        riverApplication.ADD(riverNamespace)
11     for each task ∈ pool
12     do riverAction ← GENERATE(task)
13        if LINKEDDO(task)
14          then riverEntity ← RETRIEVE(DataObject)
15               riverEntity.ADD(riverAction)
16          else riverApplication.ADD(riverAction)
17     securitySpecificationDoc ← NEW()
18     for each securityAnnotation ∈ pool
19     do securityMeachanisms ← GENERATESC(securityAnnotation)
20        riverApplication.ADD(securityMechanisms)
21        securitySpecifications ← GENERATESP(securityAnnotation)
22        securitySpecificationDoc ← ADD(securitySpecification)
```

Due to lack of resources, our prototype is currently limited to public and private process models and collaboration models. We do not foresee any fundamental problem in extending the prototype to support SecBPMN2 choreography models.

5.2 Evaluation and Discussion

To evaluate our approach, we used the framework to generate River applications for the PE system. Following the process proposed in Figure 3, we modeled

the business processes and then we generated the River applications, using the software tool we implemented. The choice of SecBPMN2 was appropriate since the modeling language was expressive enough to specify the business processes and the security choices. The generation required no effort, and its execution took only few seconds. We successfully deploy the generated River applications on a River server sandbox. The usage of our approach saved lot of effort and time in the first part of the implementation phase, where River skeletons are defined. While the overall evaluation of the approach was very positive, we also observed several limitations that need to be addressed before a commercialization is possible. In particular, we identified the following restrictions:

Manual written code: we generate skeletons of River applications. Although, we try to minimize the human intervention after the generation of the River applications, we believe that, with the current technologies, is hardly possible to completely remove the intervention of developers after the generation. The price to pay in order to automatically generate complete River applications is to collect all the information required, for example the actual implementation of the business logic, before the generation. This would only move the effort required to developers before the generation and, nevertheless, it would lead to a less intuitive, and less flexible framework.

Choice of security controls: Security constraints can often be fulfilled by different security controls. For example, a confidentiality requirement can be implemented by role-based access control or by encryption based access-control. In our prototype, we decided to limit the choice of security controls to, first, increase the usability, and, second, to be able to ensure the compositionality of the security controls. For applying our approach to further application domains, we would need to guide a security expert in selecting the most suitable security control as part of the generation.

Limitations of implementing "security-by-design": While our framework is designed with "security-by-design" in mind, due to technical and fundamental limitations, it cannot be fully achieved. First, there are security controls that require a run-time configuration (e.g., access control) and, second, security controls that are part of the generated implementation could be modified during the development process. With respect to the first item, we are generating requirements documents that need to be fulfilled while configuring the actual system. With respect to the second item, we would need to integrate consistency checks that ensure that the generated source is not modified during development. Moreover, the generated security controls require that manually developed parts to not violate the security requirements. To ensure this, we envision to implement static source code checks (see [18] for a first work in this area).

Cross-organizational security constraints: Currently, our approach has only very limited support for cross-organizational security requirements such as separation-of-duties across multiple organizations. This is a challenge which is out of scope of our work, as it requires collaborations between the organizations on the overall system level, e.g., by using a federated identify management

systems. As soon as such federated security systems are used, our framework will support cross-organizational requirements.

Not all of those restrictions are limiting the wide-spread use of our approach similarly. For example, relying on a "honest developer" is not considered to be a roadblock as the current framework already advanced the state of the art with respect to building secure process-aware systems significantly.

5.3 Related Work

In the area of secure process-aware systems, a variety of BPMN-based approaches have been proposed for modeling security, privacy, and compliance aspects (e.g., [9,19,20]). The BPMN meta-model is extended with new attributes and properties, and different selections of security, privacy, or compliance properties are considered. However, none of these proposals provide support to map BPMN models into artifact centric implementations. The approaches to implement and enforce security properties mainly focus on integrating security control mechanisms (e.g., access control infrastructures) into business process execution engines [21]. Lohmann et al. [22, 23] discuss also the integration of compliance aspects into artifact centric business processes.

In the area of mapping or transforming control-flow centric business process specifications to artifact based models, the number of existing proposals is surprisingly small. Estañol et al. [24] present a mapping of BPMN to UML models with OCL constraints. The data model of the target language, i.e., UML/OCL class models, is conceptual very close to the River language. The pure mapping of business process artifacts results very similar to our approach. In contrast to our work, Estañol et al. [24] do not discuss security at all. Moreover, their approach is not supported by an actual implementation.

6 Conclusions and Future Work

To our knowledge, this paper presents the first automated framework for translating security-aware control-flow-centric business-process-models to a secure artifact-centric implementation of process aware systems. While our prototype used SecBPMN2 and SAP River, the underlying approach is generic and can be applied as well to other security-aware business process languages as well as other artifact-centric frameworks and languages. Adapting the approach to a different security-aware business process languages, e.g., SecureBPMN [9], changes the set of supported security properties, which might require the development of new mappings. Adapting the mapping to different artifact-centric frameworks, e.g., ABAP (used by the SAP Business Suite) or PeopleCode (used by Oracle PeopleSoft) that already support access control is easy.

We plan to extend our approach along at three lines of research: (i) automated generation of validation checks to be executed after each update of security-related configurations; (ii) as preliminary discussed in [18], automated check for the implementation validation; (iii) integration with monitoring and process mining frameworks.

Acknowledgments. This research was partially supported by the ERC advanced grant 267856, 'Lucretius: Foundations for Software Evolution', www.lucretius.eu.

References

1. OMG: BPMN 2.0. OMG. (Jan 2011) www.omg.org/spec/BPMN/2.0.
2. OASIS: Web Services Business Process Execution Language. OASIS. (Apr 2007) docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.
3. SAP SE: SAP River Developer Guide. (2014) Document Version 1.0, SAP HANA SPS 08.
4. Doolittle, J.: PeopleSoft Developer's Guide for PeopleTools and PeopleCode. McGraw-Hill Osborne Media (2008)
5. Paja, E., Dalpiaz, F., Giorgini, P.: Managing security requirements conflicts in socio-technical systems. In Ng, W., Storey, V.C., Trujillo, J., eds.: ER'13. LNCS 8217, Springer (2013) 270–283
6. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems – Challenges, Methods, Technologies. Springer (2012)
7. SAP SE: SAP Payment Engine Website. www.sap.com/services-support/svc/custom-app-development/cnsltg/prebuilt/payment-engine/ Last visited Mar. 28th, 2015.
8. Salnitri, M., Dalpiaz, F., Giorgini, P.: Modeling and verifying security policies in business processes. BPMDS '14 (2014) 200–214
9. Brucker, A.D.: Integrating security aspects into business process models. it – Information Technology **55**(6) (2013) 239–246
10. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. IBM Syst. J. **42**(3) (July 2003) 428–445
11. Keller, H., Krüger, S.: ABAP Objects. SAP PRESS (2007)
12. SecBPMN Website. www.secbpmn.disi.unitn.it Last visited Mar. 28th, 2015.
13. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32**(3) (2009) 3–9
14. OMG: OMG Unified Modeling Language, Infrastructure, V2.1.2 (2007) www.omg.org/spec/UML/2.1.2/Infrastructure/PDF.
15. Wainer, J., Barthelmess, P., Kumar, A.: W-RBAC - a workflow security model incorporating controlled overriding of constraints. Int. J. Cooperative Inf. Syst. **12**(4) (2003) 455–485
16. Simon, R., Zurko, M.: Separation of duty in role-based environments. In: CSFW '97, IEEE Computer Society (1997) 183–194
17. Ferraiolo, D., Kuhn, R.: Role-based access control. In: 15th NIST-NCSC National Computer Security Conference. (1992) 554–563
18. Brucker, A.D., Hang, I.: Secure and compliant implementation of business process-driven systems. In: SBP '12. LNBIP 132, Springer (2012) 662–674
19. Mülle, J., von Stackelberg, S., Böhm, K.: A security language for BPMN process models. Technical report, University Karlsruhe (KIT) (2011)
20. Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN extension for the modeling of security requirements in business processes. IEICE - Trans. Inf. Syst. '07 **E90-D** 745–752
21. Brucker, A.D., Hang, I., Lückemeyer, G., Ruparel, R.: SecureBPMN: Modeling and Enforcing Access Control Requirements in Business Processes. In Atluri, V., Vaidya, J., Kern, A., Kantarcioglu, M., eds.: SACMAT '12, ACM (2012) 123–126

22. Lohmann, N.: Compliance by design for artifact-centric business processes. Information Systems **38**(4) (2013) 606–618
23. Lohmann, N., Nyolt, M.: Artifact-centric modeling using BPMN. In Pallis, G., Jmaiel, M., Charfi, A., Graupner, S., Karabulut, Y., Guinea, S., Rosenberg, F., Sheng, Q.Z., Pautasso, C., Mokhtar, S.B., eds.: ICSOC '12 Workshops. LNCS 7221. Springer (2011) 54–65
24. Estañol, M., Queralt, A., Sancho, M., Teniente, E.: Artifact-centric business process models in UML. In Rosa, M.L., Soffer, P., eds.: BPM '12. LNBIP 132, Springer (2012) 292–303