Modelling and Transforming Security Constraints in Privacy-Aware Business Processes

Jutta Mülle, Silvia von Stackelberg and Klemens Böhm
Karlsruhe Institute of Technology (KIT) 76131 Karlsruhe, Germany

Abstract—Security and privacy mechanisms are essential for business processes (BPs). In particular, BPs dealing with PII (personally-identifiable information) require mechanisms to counter misuse of PII and give data owners control over their data. Currently, business-process-management systems (BPMSs) provide relatively little security support, and programmers must be familiar with interfaces of security mechanisms not part of BPMSs. This leads to high implementation and maintenance costs. Our approach in turn is a wholistic one, providing security support from the modelling to the runtime phase of a BP lifecycle. As current security-modelling approaches lack features important for BPs in service-oriented architectures, we propose a sophisticated language to formulate respective constraints. In addition, we considerably ease how data owners can control their preferences at process runtime. Our security language is embedded in BFMN. We have extended an open-source BPMMS on the modelling and configuration layer. The BPMS extensions transform BPMN schemas with security constraints into executable secure processes in a versatile manner. Our language is sufficiently rich because we have implemented all identified requirements in a complex, real-world scenario.

Keywords—process modelling; secure business process management; security; privacy; model-driven development

I. MOTIVATION

The support of security goals like authorization, authentication, and confidentiality of data is essential for business processes (BPs) in service-oriented architectures (SOA). In addition, BPs dealing with personally-identifiable information (PII) have to support explicit control of data by their owners, the use of trusted services to process PII, etc. Currently, business-process-management systems (BPMSs) do not support these aspects in an adequate way. From the functional perspective, BPMSs offer some basic security mechanisms, but not the full range [1]. Thus, programmers tend to embed security functionality (e.g., auditing) into BPMSs by hand. However, BPMSs should offer integrated security mechanisms which are easy to use and maintain.

Moreover, BPs dealing with PII have to protect private data from misuse. Data owners will not accept the handling of their personal data without appropriate mechanisms. Further, legal regulations (e.g., European laws on sensitive data [2]) constrain the maintenance of data with respect to security and privacy. In particular, this holds in domains such as e-health or recruitment, where access to personal data must be restricted to specific role holders (e.g., physicians or placement officers) and for specific purposes (e.g., processing of data according to terms and conditions).

Problem Statement: Wolter et al. [3], among others, propose to attach security constraints to BP models. This provides a shared communication base for managers, system designers, security experts, and programmers. But the security vocabularies proposed so far are not sufficiently comprehensive. For instance, one cannot represent delegation in BP for data-access rights. Further, the vocabularies facilitate the specification of high-level security goals, but rarely allow for detailed constraints within the process model. To illustrate their shortcomings, some approaches allow to represent confidentiality as a security goal, but there usually is no way to specify which realization alternative to use.

A further issue is that BPs must consider users’ specified preferences. With conventional approaches, users specify preferences a priori, such as “who is allowed to access their data” or the trust level of service providers which may process their data. However, they should be able to specify their preferences at process runtime, a feature we refer to as user involvement. This is because preferences vary for different instances of a BP, because their contexts are different. Think of a user involved in selecting a service provider to process his PII. Services available depend on the application. But it is tedious to model user involvements explicitly, and it requires specific knowledge about security, trust, privacy and the related mechanisms available. For example, to specify the selection of a service deemed trustworthy by the user, we had to model 11 activities, including user interactions consisting of several steps (see Figure 4).

Yet another shortcoming of existing approaches is that there is no integration with an infrastructure for BP execution.

Challenges: One research issue is to specify a security language covering a comprehensive set of security features for BPs in SOA, embedded into a process-modelling language. This is challenging: It calls for a systematic analysis of functional requirements. An example for an issue that remains to be specified is the delegation of access rights to data used in a BP. A further challenge is to identify a canonical set of user involvements and to provide support at the modelling layer. Another issue is the transformation of the various language constructs into executable processes in an extended BPMS environment. "Extended" means that the environment provides the security functionality required.

Contributions: (1) We propose a language sufficiently rich to represent security constraints. It also supports user involvements, which we have identified in a requirements analysis of the employability domain. We have successfully used
this annotation language to model two real-world scenarios. To illustrate, process designers can now specify the privacy aspects of a service selection, concerning a service that has to process personal data, by one annotation term with few parameters. We embed our language in BPMN annotations. With this, the language is BPMN-conform at the syntax level. This reduces our implementation effort necessary. (2) We have found out that there are three kinds of targets when transforming our privacy and security constraints: security policies at an abstraction level of XACML, adaptations of the process schema, and parameter settings for invocations of security components. By implementing the transformation, we demonstrate that different constraint types have different kinds of targets. Regarding realization, dedicated components, which we have added to the extended BPMS, enforce process-specific security policies, let users control their privacy constraints, and manage security-relevant variables of the BP.

In summary, our approach supports process designers and programmers by providing a broad range of integrated security and privacy functionality.

Paper outline: Section II describes security requirements for BP designs. Section III gives an overview of our language. Section IV describes the transformation and enforcement components. We then discuss related work and conclude.

II. REQUIREMENTS

In an earlier study for the employability domain, we have systematically collected security requirements for BP designs. They form the basis of our annotation language. The first requirements concern security aspects of BP designs. The latter ones say how to let users specify security, trust and privacy aspects during process execution.

Authorization regulates which users may execute activities or access the process data. Respective mechanisms have to ensure that access rights hold exactly for the execution time of the activities in question. Since BP designs typically connect activities, we see further challenges in the coordination of access control with the process flow, namely to separate and to bind duties. The latter is desirable in order to obtain clear responsibilities for related activities. We need delegation mechanisms to transfer access rights to data. In a BP context, we need to support two kinds of delegation. First, it must be possible to delegate access rights from the BP to web services. This might require particular rights to access external data stores. Second, if access policies, such as sticky policies, protect data objects there is a need to delegate access rights for data to other process participants. The delegation of access rights in SOA is challenging and existing approaches, e.g., delegating credentials, fulfill partly the requirements. One needs mechanisms to delegate access rights on object granularity that are restricted for BP-specific situations.

Authentication functionality ensures that a system can rely on the identity of actors and on the attributes of actors. An actor can be a human or another process. It is challenging to provide authentication in distributed, inter-organizational environments with mechanisms such as single-sign-on (SSO). There, process designers have to rely on federated identity-management systems where identity providers (IdPs) authenticate individuals and provide certified attributes about them. Auditing means that the system monitors the BP, to provide traceability. Typically, compliance regulations frame auditing. Auditing can affect any aspect of BP designs. For example, execution of individual activities, the message flow, access to data or to external services as well as role assignments can be logged, and thus requires flexible specifications.

Confidentiality is the protection of data in BP designs against access attempts that are not allowed. This protection includes calls of external web services. Authorization constraints already are a first step towards confidentiality, but additional mechanisms are needed, such as encryption.

Data Integrity is commonly perceived as another security goal. It means consistency, accuracy and correctness of data. To illustrate, a system has to establish data integrity when BP designs terminate exceptionally, due to the occurrence of events such as errors. Integrity ensures that a system must act in an expected way at each point in time regarding the transfer, processing or storage of data.

Security- and privacy-aware BP designs call for additional properties beyond these functional requirements. We now motivate that our language includes user involvements to specify security, privacy and trust aspects.

Involving Data Owners: Privacy laws state that providers have to ask for agreement of data owners to use their personal data in certain situations. Further, data owners might want to control who should be allowed to have access to their data. This requires user interactions as part of the BP design, as motivated earlier. Further, BP designs couple web services that users may deem more or less trustworthy. We understand trust as the reputation a service provider has gained in the past by the users. As trust is commonly perceived as a facet of security, we postulate to consider trust-specific aspects. In particular, we call for involving data owners into the specification of trust and the selection of available services at process runtime. BPMSs have to facilitate the selection of web services dynamically, depending on the trust level of available service providers. Since service availability changes over time, users should be able to modify their preferences according to the BP context. To illustrate, a data holder might have specified a level of trust that service providers accessing his data must have, but now he has to modify it at runtime, because his specifications cannot be fulfilled otherwise. Modelling security-specific user involvements as part of the application logic would require detailed security and privacy knowledge and would result in complex models. Thus, re-usable mechanisms to represent complex, recurring user involvements are needed.

III. MODELLING SECURITY CONSTRAINTS FOR BUSINESS PROCESSES

Our language dealing with these requirements is embedded in BPMN 2.0 as structured text annotations. The aim is to represent security constraints in BP designs declaratively. In this paper, we give an overview of the language and describe
some selected constraints in detail. The constraints chosen are typical in the BP context. They also serve as examples to describe the transformation process in Section V. Due to restricted space, we refer to [7] for further details.

We group the annotation types of our language by requirements, namely authorization, authentication, auditing, confidentiality, integrity, and privacy- and trust-related user involvements. The first groups are well-known in literature, but not sufficiently specified in existing approaches for BPs [8], [9], [10], [3], [11]. For details, we refer to Section V. Annotations for privacy- and trust-related user involvements are a new notion.

Annotations have the following generic structure:

$$\langle \text{Annotation term: list(parameter-name}=\text{"value"}) \rangle$$

Parameters can be optional. Each annotation term is defined for a particular set of BPMN 2.0 elements, i.e., for activities, lanes, data objects, data stores, or message flows. According to the BPMN standard, an activity can be a simple task or a sub-process.

A. Authorization Constraints

The authorization constraints (Table I) specify who possesses which rights under which restrictions. Dots indicate parameters that have been omitted. To manage authorizations, one typically specifies the roles allowed to perform or to access annotated elements.

<table>
<thead>
<tr>
<th>Authz Constraint</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role Assignment</td>
<td>$$\langle \text{Assignment: type}=\text{&quot;Role&quot; name}=\text{&quot;Srolename&quot;} \rangle$$</td>
</tr>
<tr>
<td>Mechanism Assignment</td>
<td>$$\langle \text{Assignment: type}=\text{&quot;Mechanisms&quot;} \dots \rangle$$</td>
</tr>
<tr>
<td>User Assignment</td>
<td>$$\langle \text{Assignment: type}=\text{&quot;User&quot; name}=\text{&quot;Susername&quot;} \rangle$$</td>
</tr>
<tr>
<td>Separation of Duty</td>
<td>$$\langle \text{SoD: role}=\text{&quot;Sm&quot; number}=\text{&quot;Sm&quot;} \dots \rangle$$</td>
</tr>
<tr>
<td>Binding of Duty</td>
<td>$$\langle \text{BoD: spec}=\text{&quot;weak&quot;} \dots \rangle$$</td>
</tr>
<tr>
<td>Adaptation</td>
<td>$$\langle \text{Adaptation: rights}=\text{&quot;Srightname&quot;} \dots \rangle$$</td>
</tr>
<tr>
<td>Task-Delegation</td>
<td>$$\langle \text{T-Delegation: target}=\text{&quot;Srolename&quot;} \dots \rangle$$</td>
</tr>
<tr>
<td>Data-Delegation</td>
<td>$$\langle \text{D-Delegation: rights}=\text{&quot;Srightname&quot;} \dots \rangle$$</td>
</tr>
</tbody>
</table>

### TABLE I

**OVERVIEW OF AUTHORIZATION CONSTRAINTS**

Authorization constraints, except for delegation for data objects, can annotate all kinds of BPMN activities, pools, and lanes and hold for all activities contained therein. Assignment annotations allow to bind roles or users to BPMN elements, so that only role holders or specified users are allowed to execute the activities of these elements. Further, Mechanism Assignment allows to specify the resource-allocation mechanism used, to assign individuals to perform the annotated BPMN activities. The annotations for separation of duty (SOD) and binding of duty (BoD) constrain the execution of a group of activities or of multiple instances of an activity, to be performed by different individuals (SoD) or by the same one (BoD). To avoid process blocking because, say, process participants are absent, we allow to ease BoD constraints by explicit transfer of responsibility, i.e., a weakening of the BoD constraint by means of a re-assignment. The optional parameter spec = "weak" specifies this. To give way to ad-hoc process changes in a controlled way, we allow the specification of rights to adapt process instances.

**Example:** Figure [1] shows two role assignments for lane “placement coordinator” and lane “learner”, next to other annotations. This means that these lanes represent roles. The BoD annotations in Figure [1] mean that a learner cannot transfer activities to other role holders of learner, while a re-assignment of activities by placement coordinators is possible.

**Delegation:** Delegating access rights may apply to BPMN activities, so-called task-based delegation, but also to data objects. To differentiate between these two kinds of delegations, we introduce $$\langle \text{T-Delegation}\rangle$$ and $$\langle \text{D-Delegation}\rangle$$. We now describe data-based delegation. Process participants can delegate rights to access data objects to someone who currently does not have the authorization. In our context, delegation deals with access rights to data which is external or is controlled by the same process; see the data handling in BPMN 2.0 (Section 10.3 in [12]). Delegation varies depending on the category of the data concerned and therefore requires different parameters. We see two categories: A data object is local to the process, and its lifetime is bound to the process where it is defined; data stores are external data sources which persist beyond the lifetime of a process. For instance, this could be a database, say, with personal data, like e-health or e-portfolio data. For internal and external data, there can exist access restrictions. Rules to access data stores are often specified using external mechanisms. Access rights to data objects in turn have to be defined explicitly within the BP. During process execution, role holders who have the right to access the data in question might lose it. Next, external services called need access to external data which a role holder but not the service itself is authorized to. These are situations requiring delegation of access rights.

The annotation

$$\langle \text{T-Delegation: rights}=\text{"Srightname"} \text{ target}="\text{Sobjectname}" \text{ interval}="\langle \text{Sactivityname1}, \text{Sactivityname2}\rangle\text{\$group}" \text{ role}=\text{"Srolename"} \text{ poolname}="\text{Spoolname}" \text{ \$lanename} \text{ \$spec}="\text{Sobjecttype}\text{\$specification}\rangle$$

specifies the delegation of access rights to data, with read, write, delete, or policy as possible values for rights. If there is a delegation of access rights which are attached to the data object in form of a sticky policy, the rights parameter is optional, but in this case the target parameter is required. The parameters target, role, poolname, ws, spec are optional.

The specified rights or the rights of the object specified in the policy are delegated. The rights describe which use of the parameter Sobjectname is allowed. The validity of a delegation can be specified for the execution time of activities in the interval \$activityname1, \$activityname2\text{, i.e., the part of the process starting with Sactivityname1 and ending with Sactivityname2, or in the Sgroup of activities. The delegate is specified using either an rolename explicitly given or implicitly defined by a poolname or a lanename, which are used as rolenames, or a webservicetypename. spec denotes a delegation with grant rights for the delegate.}

**Example:** Figure [1] shows a process activity dealing with a
personal data store (PDS). The annotation \texttt{≪D-Delegation ...≫} gives access rights to the “$View-PDS-of-Learner” from the holder of role “Placement coordinator” to the web service of type “PDS”.

### B. Authentication

The annotation \texttt{≪Authn: attributes= list ("attributename", "value") idp="Identity provider"≫} means that process participants must be authenticated. The task of an IDP is to certify attributes of process participants. This certification leads to authentications. For this purpose, we allow for specifying a list of attributes which an IDP has to certify. For instance, any holder of role “learner” in Figure 1 must be authenticated as a student at KIT and authenticated by an identity provider defined in the annotation.

### C. Auditing

The annotation \texttt{≪Audit: policy="policyname"≫} enforces a monitoring of the execution of an activity, a group of activities, a data access, an event, or a message flow. Legal requirements call for different kinds of logging. For example, if a process handles personal data, it must be logged, among others, who has performed which action upon which data, and at which time [13]. Thus, we assume that an auditing policy (parameter \texttt{policy=“policyname”}), stored in the BPMS, specifies the objects to be logged and the kind of logging. To illustrate, we have annotated the pool “placement process” with the annotation term \texttt{Audit} in Figure 1. This specifies a monitoring of all task executions and all security tasks, e.g., the delegation or the encrypted call of the matching service according to the policy “auditAll”.

### D. Confidentiality and Integrity

Authorization mechanisms support a basic level of confidentiality, because only authorized humans or services may perform activities and access data. To improve confidentiality the system can also protect message flows. This can be expressed by \texttt{≪Conf≫}.

We allow the parameter “spec”. Its possible values are “encrypted” or “signed”. For example, the data flow from activity “Import personal data to PDS” to activity “PDS” in Figure 1 must be encrypted. This addresses the shortcoming of existing security vocabularies exemplified in the introduction.

The annotation \texttt{≪Integrity≫} means that data integrity must be enforced. We allow the annotation of data (i.e., data objects, data stores, data inputs, data outputs) and message flows.

### E. User Involvements

We have identified so-called user involvements to support user concerns, e.g., privacy concerns of data owners. These user involvements are important, because users have to specify and agree to privacy and trust preferences that are context-specific at runtime. Having examined various application scenarios, we have identified the following user involvements: giving consent, selecting services in line with the trust levels required, specifying data-access policies or trust levels. There is also the need to allow to give feedback about web-service calls, in order to be able to compute the trust level of these services later on. To support the representation of user involvements, we introduce annotation terms to specify them declaratively. One can annotate activities and certain events of a process model with user involvements. Thus, the process designer simply specifies them instead of modelling them explicitly. The representation is as follows:

\texttt{≪UInvolve: list(parameter-name=value)≫}
Each user involvement has a parameter of name type, and its value specifies the intention of a user, e.g., type="consent" means giving user consent. Unless specified otherwise, these involvements are invoked right before the annotated activity takes place. Otherwise, there is a parameter insertplace, indicating the position of the involvement in the sequence of activities with the activity the fragment has to be inserted before. Our constraints (Table IV) allow to specify when during process execution users can set their privacy preferences.

**Select Trust Policy.** It is intuitive to characterize service providers by trust levels, e.g., their reputation gained in the past. Process participants can specify the minimum trust level of providers by means of a trust policy. The following annotation specifies the selection of a trust policy:

> ≪ UInvolve: type=“SelectTrustPolicy” display=“list(option)” ≫

*Example:* The user involvement of type “SelectTrustPolicy” in Figure 1 means that a learner has to select a trust policy from the options given.

### IV. Transformation of Security Annotations

In this section, we describe how we transform our security annotations of process models to the execution level. A BPMS enhanced with BP-specific security components, see Section IV-A, executes the BPs. Depending on the nature of the security constraints, we have identified three kinds of transformation targets. We explain the specifics in Section IV-B.

#### A. Secure Business-Process-Management System

A common security framework, but without explicit BP support, is XACML [14]. It is a declarative access-control policy language. The processing model of XACML contains the following components: a policy-decision point (PDP), to check against security policies whether certain operations are allowed, a policy-enforcement point (PEP) to intercept access requests and to enforce the decision of a PDP, and a policy-information point (PIP) to provide further security information for PDP and PEP. In addition to the XACML framework, a security framework in a heterogeneous environment requires further functionality. There, authentication typically relies on federated identity management with IDPs. Finally, a trust PDP to manage trust and an auditing bus to support auditing are further components of the security framework we rely on.

The security requirements of Section III necessitate the refinement of the components of the XACML processing model by BP-security-specific aspects. We refer to the new BP-specific components as PEP-WS, PEP-HT, BP-PDP and PIP. In particular, the components have to take into account the history and context of the process instance, to, say, bind duties to a specific role holder, or to ensure that access rights hold exactly for the execution time of the activities in question.

Further, delegating access rights is essential for service calls handling private data which the process performs for a user. In this case, the BP has to transfer the necessary rights to the service called, e.g., to access an external data store.

The architecture in Figure 2 (see [15] for details) contains the main components and interfaces of a BPMS according to the WfMC reference model [16]: the BP Engine in the middle connected with interfaces to the other components, the Web Services component dealing with calls to web services and to other processes (Interfaces 3 and 4), and the Human Tasks component (Interface 2) with a worklist handler and binding of client applications, i.e., typically web interfaces to users and a user interface with a user-specific worklist. The Administration and Monitoring Tool Component is not part of this figure. Interface 1 connects the modelling component to the engine.

![Fig. 2. Architecture of a BPMS with security extensions (from [15])](image)

To give way to security-specific functionality, in our team we have extended a WfMC-compliant BPMS architecture as follows [15]: A dedicated security component, the BP-specific PEP, handle all interactions with users (PEP-HT) and with external web services (PEP-WS), as a proxy of the engine. Before any web-service call and any execution of human
tasks, the PEP-WS checks if access rights are sufficient, if the transport has to be encrypted etc. The same holds for incoming service calls.

Security modelling complements the BP modelling. The resulting security-enhanced BP model is input for our transformation component which we present next. Figure 2 gives an overview of the data involved in this life-cycle of secure BPs.

The design phase results in a BPMN model with security annotations. To deal with security annotations, we have developed a dedicated tool, the Security Modelling and Configuration Component for BP (BP-SMC). In the execution phase, the secure BPMS runs instances of the resulting process enforcing the security constraints. In the following, we describe the transformation phase.

B. BP-Security-Model Transformation

The main concern of this phase is the transformation of an annotated BP model to representations which support the enforcement of the annotated constraints during process execution. To this end, we have to deal with several system layers described by distinct models. Model transformation is essential for any model-driven software development (MDD) [17]. MDD approaches for security in BPs, see [18–21] in particular, have, e.g., generic process models and security models as source, and XACML policies or UML use case models as targets of the transformation. Our transformation approach starts from a BPMN process model and a security model, i.e., the annotations. The BPMN meta model, i.e. the source model of our transformation, is expressed as a Meta-Object Facility (MOF) model [22]. We have discovered that different security constraints on BPs need to be transformed in different ways, i.e., have different outcomes depending on the security category of the annotation. A transformation has at least one of the following effects: generating or modifying a BP-access-control security policy, setting parameters to configure security components, or adapting the BP by canned process fragments. This leads to the three target models: XACML as policy language enhanced with BP-specifics, a data model of configuration parameters for the security components of our secure BPMS, and a BPMN process model adapting the model of the application process.

a) BP-security policy: A security policy consists of rules, such as authorization rules or rules protecting the data flow, e.g., with delegations. BPs control the sequence of tasks. Thus, general security-policy languages such as XACML are not sufficient. The enhancements include security rules for tasks and flows of tasks, and constraints related to the process status. We refer to such an enhanced policy as BP-access-control security policy or BP-security policy, in short. We argue that one should map these security annotations to policies the extended PDP can handle.

The transformation in this category results in an XACML policy, enhanced with BP-specific constructs. In particular, security constraints may refer to tasks and relationships between tasks, as well as to the execution history, like BOD, and the execution status of the BP. During process execution, the BP-PDP checks any access to tasks for conformance to this policy.

Example: Think of a BoD constraint that defines the part of the process BoD is valid for and the role it holds for. The transformation of annotation ≪BoD: spec=“weak”≫ of the Placement Coordinator lane in Figure 1 generates additional rules that become part of the BP-access-control security policy:

```xml
<TaskGroupSettings>
  <TaskGroupSpec TargetID=“PlacementCoordinatorLane”
      BoD=“true” weakBoD=“true” />
</TaskGroupSettings>
```

When assigning a holder of role “placement coordinator” to a human task, the PEP-HT in interaction with the worklist handler calls the BP-PDP to check if the assignment envisioned observes the access rules of the BP-security policy, e.g., in case a BoD constraint is affected.

b) Configuring security components: The second kind of transformation concerns the parameterization of calls to security-specific components other than the BP-PDP. These components, PEP and PIP in particular, take process context like validity periods of security rules, history of the process flow or role holders involved in the process into account. The PEP comes into play, because calls going to or coming from external components or human tasks are tasks the secure BPMS first calls the PEP for. The PEP then executes the security checks required, e.g., a PDP call to check access rights to parameters of the call. Further, it configures the call with security parameters, e.g., a WS-Security header with encryption parameters. The BP-SMC passes information taken from the annotations to the PIP, to facilitate parameterization of security calls issued by the PEP. If a component is responsible for compliance with an annotation, transforming the annotation yields the parameters of the invocation of that component.

Example: ≪Authn: ...≫ will result in an authentication call to an IDP. The annotation and its context information yield the identifying attributes of the role holder in question. This information is stored in the PIP so that it can be used in other PEP calls during process execution. As another example, think of a ≪Conf≫ annotation of a message flow to a web-service task. It specifies parameters to invoke the web service in an encrypted manner or with a digital signature, see Figure 4.

We have identified a set of security-configuration parameters required to call the security components of the secure BPMS. The design of our security-annotation language from Section 11 has taken these configuration opportunities into account. Examples of the security components are an IDP, service discovery using security and trust constraints, an audit bus for logging, a trust PDP, or a security PDP of an external data store. In consequence, the PEP can set WS-Security-compliant parameters for the WS call as well as parameters for calls to other components.

c) Process adaptation: The third kind of annotation needs an explicit realization with several activities to enforce it. It must be tailored to the sequence of the BP, and it can involve process participants. In addition, calls to security com-
components, e.g., PDP checks, calls to service discovery etc., can be necessary with this kind of annotations. In particular, this is the case for \(\text{≪UInvolve≫}\) annotations. Another example is the \(\text{≪Audit≫}\) annotation. It results in a BP task that calls the PEP to initiate a call to the audit bus to log the execution of the annotated BPMN element. Note that the logging call is parameterized, and the parameter values are an outcome of the transformation of the respective annotation. The parameters have been stored in the PIP.

We deal with this type of annotation by adapting the process model, i.e., by plugging process fragments into it, with security-specific activities. For each \(\text{≪UInvolve≫}\) annotation, there usually exists more than one fragment, depending on the behaviour required. I.e., there is the need for semantic specification in order to select an appropriate one.

**Example:** In the process model from Figure 1 a “Learner” might allow web services to match her personal e-portfolio data with job offers only from trusted web services. To take trust into account for the service selection, the process designer can now annotate the activity with a \(\text{≪UInvolve≫}\) constraint of type “SelectTrustPolicy”. It specifies that the “Learner” can select one of the options listed as display parameter:

\[
\text{≪ UInvolve: type=“SelectTrustPolicy” display=“trust All, average Feedback, most reputable Users, 5 most reputable Users” } \]

To illustrate, Figure 4 shows the process fragment which is inserted into the BP before the annotated activity “Call Matching Service”. It contains the following activities and user interactions:

- A user interaction, i.e., tasks to prompt the user. The web form, which is provided together with the process fragment, must contain (links to) the trust policies available to be checked during the transformation. They have to be parameters of the annotation. This requires a consistency check, which is part of the process fragment (but, due to space problems not contained).
- The DiscoveryService task causes a PEP-WS call of a web-service discovery with the trust policy selected as parameter.
- If there is no service with that trust level, there is a further iteration of trust-level selection and service discovery.

Using pre-defined BP fragments brings several benefits: The process designers do not need to be familiar with the security components. Next, providing such fragments supports the correct implementation of the annotations.

V. RELATED WORK

Wolter et al. [10], [18], [3] and Menzel et al. [11], [23] have presented work on modelling security for BPs in SOA. [3] extends [10] by a broader security vocabulary. It also comprises aspects such as integrity and confidentiality. [10] focuses on authorization constraints in BPMN annotations, but does not consider delegation of data-access rights, weak BoD, and direct user assignments. Authorization is limited to human tasks. [9] proposes a broad security vocabulary for BPMN artifacts. However, it lacks authorization constraints (such as SoD), and it is less detailed than our language.

[24] proposes an XACML authorization scheme to handle permissions of a WS-BPEL process, the related role assignment and the associated attributes of a role. Further, they introduce a constraint language to specify authorization constraints for BPs and an RBAC (role-based access control) WS-BPEL specification with support of human activities, as well as algorithms for the enforcement level. In contrast to our approach, they do not focus on security facets other than authorization, role assignment and policy checks at the enforcement level.

[11] annotates inter-organizational processes with trust. We have a different understanding of trust: Users specify their desired trust level of web services interactively. Our approach to support privacy differs from Short et al. [25]: They focus on privacy aspects by restricting purpose, recipient and retention obligation of data in a sticky policy, we enable
a generic protection of data by access-control and delegation mechanisms. Alhaqbani et al. [26] assume that a BP engine accesses data owners’ privacy preferences (e.g., available as a policy). They let it open how and when users specify their policies, and do not address that user preferences change with BP-context. We involve users at BP runtime to specify their preferences.

[18] describes a model-driven transformation of security annotations into XACML. It allows to relate security annotations to security components and has some similarity to our transformation of annotations to configuration parameters. We use a BPMN-specific security model, they, in turn, a generic model, and they do not deal with user involvements. There also is no counterpart to our transformation resulting in an adapted process model. [23] defines a process-independent policy language and provides a mapping to a design model for security constraints. The model-driven approach in [11] translates security constraints into security configurations in SOA. However, there does not seem to be an implementation for process execution. A model-driven approach transforms security constraints into implementation specifications [27]. [28] uses MOF to represent the models for MDD as well. They deal with security using view-based access control, but not related to BPs.

Our language is first to deal with security-specific user involvements. Finally, our transformation approach differs from existing ones (see, e.g., [27], [18]) because it is more differentiated.

VI. CONCLUSIONS

We have presented a rich language to represent classical security aspects as well as trust- and privacy-specific user involvements for BPs in SOA. Our extensions of a BPMS provide security support from the modelling to the runtime phase of a BP lifecycle. Regarding the transformation of our security constraints, we have demonstrated three different kinds of targets for an extended security framework. We have refined the architecture of [15] with the transformation component to configure BPs with security and user-centric privacy-and trust-related constraints. Using this infrastructure, we are able to check the execution of BP tasks and the use of BP data against the resulting access-control policy of the BP.

Acknowledgements: This research has received funding from the Seventh Framework Programme of the European Union (FP7/2007-2013) under grant agreement n° 216287 (TAS – Trusted Architecture for Securely Shared Services)

REFERENCES


