Karlsruhe Reports in Informatics 2011,9
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

A Security Language for BPMN Process Models

Jutta Mülle, Silvia von Stackelberg and Klemens Böhm

2011

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

A Security Language for BPMN Process Models

Jutta Mülle, Silvia von Stackelberg and Klemens Böhm

Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
{muelle|silvia.stackelberg|klemens.boehm}@kit.edu
http://dbis.ipd.kit.edu

Abstract. Security mechanisms are essential for business processes. Currently, business-process-management systems (BPMSs) provide relatively little security support, and programmers must be familiar with interfaces of security mechanisms that are not part of BPMSs. Enforcing security constraints for a business process 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 business-process lifecycle. As current security-modelling approaches lack features important for business processes, we propose a sophisticated language to formulate respective constraints. In addition, we considerably ease the specification of how users can control their security preferences at runtime. Our security language is embedded in BPMN. We have extended an open-source BPMS on the modelling, configuration and execution layer. The BPMS extensions transform BPMN schemas with security constraints into executable secure processes in a versatile manner. Our language is sufficiently broad, because we have systematically realized all requirements identified in a complex, real-world scenario.

Keywords: process modelling, security, user centricity

1 Motivation

Security facets like authorization, authentication, integrity, and confidentiality of data are essential for business processes in service-oriented architectures (SOA). Currently, business-process-management systems (BPMSs) do not support these aspects in an adequate way. From the functional perspective, BPMSs offer some security mechanisms, but not the full range. Next, programmers tend to embed security functionality (e.g., encryption) into BPMSs by hand. However, BPMSs should offer integrated security mechanisms which are easy to use and maintain.

[14], among others, proposes to attach security constraints to business-process models. The aim is to provide a shared communication base for managers, system designers, security experts, and programmers. But existing security vocabularies are not sufficiently comprehensive. For instance, one cannot represent delegation mechanisms for data-access credentials in SOA. Further, the vocabularies enable the specification of high-level security goals, but rarely allow for detailed constraints within the process model. To illustrate, some existing 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, with respect to security and trust in business processes, user involvement is important. With conventional approaches, users specify their security-specific 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 runtime, a feature we refer to as *user involvement*. This is because preferences tend to vary for different instances of a process, because their contexts are different. For example, think of a user selecting a service provider for processing his personal data from the services available. It is tedious to model such user involvements explicitly, and it requires specific knowledge about security, trust, and privacy. For example, to specify the selection of a service deemed trustworthy by the user, we had to model around 10 activities, including user interactions consisting of several steps [1].

Another shortcoming of most existing approaches is that an embedding into an infrastructure for execution of processes is not described.

Thus, one research issue is to specify a security language representing a comprehensive set of security features for business processes in SOA, and this language should be 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 business-process-internal and external data in SOA. The language envisioned has to be all-embracing to enable the transformation into executable processes without any human intervention. A further challenge is to identify a canonical set of security-specific user involvements and to provide support at the modelling layer. Yet another issue is how to extend a BPMS with security components in a generic way, so that an automatic transformation becomes feasible.

Contributions. (1) We propose a language that is sufficiently broad and deep to represent security constraints. It also supports security-specific user involvements, which we have identified systematically. An evaluation with two real-world scenarios has demonstrated that it is sufficiently general [1]. To continue the above example, process designers can specify the security aspects of a service selection by one annotation term with few parameters.

We embed our language in BPMN. The BPMN 2.0 standard itself does not support security aspects. However, it provides so-called artifacts to facilitate comments within business processes [11]. By using BPMN artifacts as containers for constraints, our approach is BPMN 2.0 standard conform at a syntax level. This has reduced the implementation effort necessary from our side.

(2) We have observed that there are three different kinds of targets when transforming our security constraints: security policies at an abstraction level comparable to the one of XACML, adaptations of the process schema, and parameter settings for invocations of security components. By describing the transformation, we demonstrate that different constraint types have different kinds of targets. Regarding realization, dedicated components of our extended BPMS

transform the security-enhanced business-process models into executable BPEL processes. These components assign users to tasks, enforce process-specific security policies, and manage security-relevant variables of the business process.

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

Paper outline: Section 2 describes security requirements for business processes. Section 3 introduces the security language, Section 4 describes the transformation and enforcement components. Section 5 discusses related work, and Section 6 concludes.

2 Requirements

The following requirements, collected in an earlier study for the employability domain [9], form the basis of our security language.

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. Further, authorization to control adaptations of running process instances in an ad-hoc manner should be possible. Such rights to adapt a process are useful, for example, when a data owner wants to see who has accessed his personal data, though the process does not support this activity. Authorization also implies role management, i.e., the specification which role holders may perform which activities, and the direct assignment of individuals to tasks. Since business processes typically connect activities, we see further challenges in the coordination of access control with the process flow. This implies the possibility 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 our context the transfer of credentials for data. This is for the following reasons. First, it must be possible to delegate credentials from the business process to web services, which might require particular rights to access external data stores. Further, if data objects are protected by particular access policies, there might be a need to delegate credentials for data to other process participants.

Authentication ensures that a system identifies actors correctly. An actor can be a human, but also another process. It is challenging for system architects to provide authentication in open, distributed, inter-organizational environments, in the presence of single-sign-on mechanisms (SSO) in particular. In such settings, architects typically 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 business process in order to provide traceability. Typically, compliance regulations frame auditing constraints. Auditing can affect any aspect of business processes. For example, execution of individual activities as well as the message flow may be monitored. Further, access to data or to external services as well as role assignments can be logged.

Confidentiality is the protection of data in business processes against access attempts that are not allowed. This 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 [5]. It means consistency, accuracy and correctness of data. To illustrate, a system has to establish data integrity when business processes terminate exceptionally, due to the occurrence of some events, such as errors.

Up to now, we have summarized the functional requirements from [9]. We now motivate that our language cover user involvements dealing with security issues. For instance, data owners might interactively agree to access of their data and to terms and conditions referring to the process. Modelling security-specific user involvements as part of the application logic would require detailed security knowledge and would result in complex models. Thus, our language has to support the specification of security-specific user involvements, and the BPMS security extensions have to manage them.

Further, business processes couple web services that users may deem more or less trustworthy. As trust is commonly perceived as a facet of security, we also consider trust-specific aspects in business processes. In our context, trust is, e.g., the reputation a service provider has gained in the past. BPMSs have to facilitate the selection of web services dynamically, depending on the trust level of service providers. To illustrate, a data holder might have specified a level of trust service providers accessing his data must have, but now has to modify it, because his specifications cannot be fulfilled otherwise.

The requirements discussed so far are the basis for our security language that we introduce next.

3 Modelling Security Constraints for Business Processes

Our security language is embedded in BPMN 2.0 as structured text annotations. The aim is to represent security constraints in business processes declaratively. Having identified all security requirements of business processes of two complex real-world scenarios in the employability domain in SOA environments, our language covers all these issues, i.e., the requirements from the previous section. This chapter completely specifies our language. We group our language primitives into different categories, namely authorization, authentication, auditing, data and message flow security, and security- and trust-specific user involvements. The first categories are well-known in the security literature, but not fully specified in existing approaches for business processes ([4], [12], [15], [14], [8]). Additionally, we extend authorization constraints by controlling the rights for process adaptations. Annotations for security- and trust-specific user involvements are a new notion, aiming to support user centricity.

Annotations have the following generic structure:

$$\ll \textit{Annotation term: list(parameter-name=“value”)} \gg$$

For example, ≪BoD≫ is a simple annotation term, specifying a binding of duty (BoD) constraint. The security annotations require no, one, or many parameter-name/value-pairs, dependent on the term. Some are optional in certain cases. If nothing else is said in the following, the parameter values are "string" data types.

Each annotation term is defined for a particular set of BPMN 2.0 elements, e.g., 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.

We now describe the annotation terms for authorization, authentication, auditing, data and message flow security, and security-specific user involvements in detail.

3.1 Authorization Constraints

The authorization constraints (Table 1) specify who possesses which rights under which restrictions. The dots in Table 1 stand for missing parameters. To manage authorizations, we apply roles, i.e., we specify access rights for role holders.

Auth. Constraint	Syntax
Role Assignment	≪Assignment: type="Role" ... ≫
Assignment Mechanism	≪Assignment: type="Mechanisms" ... ≫
User Assignment	≪Assignment: type="User" ... ≫
Separation of Duty	≪SoD: ... ≫
Binding of Duty	≪BoD: ... ≫
Delegation	≪Delegation: ... ≫
Adaptation	≪Adaptation: ... ≫

Table 1. Overview of Authorization Constraints

Role assignment. The term

$$\ll \textit{Assignment: type} = \text{“}\textit{Role}\text{''}\ \textit{name} = \text{“\$}\textit{rolename}\text{''} \gg$$

assigns a role to an activity, to a group of activities, to pools, or to lanes. It means that users with the given role are allowed to execute activities.

If no role name is specified, the annotation implies that the pool or lane name determine the role name. Clearly, this option is only possible for the annotation of pools or lanes and means that users holding role *poolname* or *lanename* are allowed to perform tasks of the annotated pool or lane. In particular cases, if the role name differs from the name of pools and lanes, pools and lanes are annotated with a role name. In these cases, role is an obligatory parameter. A role assignment with the parameter/value-pair *name* = "$*rolename*" is typically used for the annotation of activities or group of activities, indicating that role holders of *rolename* have to perform annotated activities. For simplification, each BPMN element can be annotated only by one role name. In case that many roles are required, the process modeller adapts the role name accordingly (e.g., by introducing an additional role name representing all required roles).

Example: Figure 1 shows two role assignments for lane "placement coordinator" and lane "learner" for a student-placement scenario. This means that lane "placement coordinator" and lane "learner" represent roles.

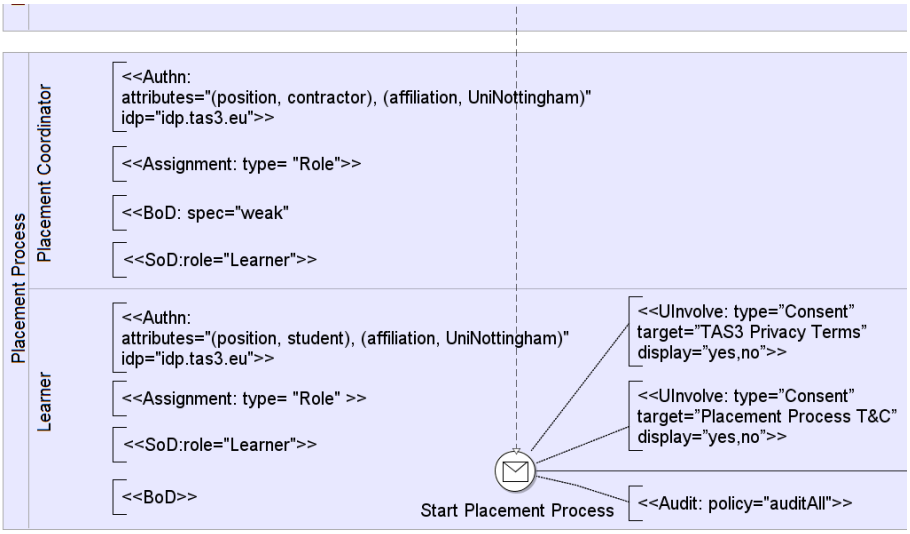

Fig. 1. Example Security Annotations for Process Start Event

An **assignment mechanism** specifies the assignment of work items to role holders. We specify a mechanism to be used as follows:

$$\ll \textit{Assignment: type="Mechanism" name} = \text{"}\$\textit{mechanismname}\text{"} \gg$$

The *mechanismname* indicates the mechanisms to be used. A mechanism can be specified for roles, activities, group of activities, and for pools and lanes.

For example, an employability company preferably assigns clerks to a customer case who have already gained experience with that applicant/customer. By supporting the specification of different assignment schemes (e.g., "assignment to role holders with the most experience", or "shortest execution time"), allocation of role holders becomes more flexible, as opposed to role holders picking up items from a work list. We facilitate the specification of assignment schemes in the process model, because they are application-specific. A mechanism can be specified for any BPMN elements with a corresponding role assignment. As roles are not part of BPMN, we allow the indirect assignment of mechanisms to roles in BPMN diagrams as follows: (1) pools or lanes represent roles (expressed by a role assignment) and are annotated with a mechanisms assignment accordingly. (2) activities or group of activities that have to be performed by a role (specified by an annotated role assignment) can be annotated additionally with a mechanism assignment. Such twofold annotations will be interpreted accordingly.

User Assignment. Our language also allows for direct **user assignment** (Table 1). A user "*username*" (process participant) is assigned explicitely to an activity or a group of activites by the annotation

$$\ll \textit{Assignment: type="User" name} = \text{"}\$\textit{username}'' \gg$$

This means that the user "*username*" has the rights to perform the annotated activities. Typically, individual users are not assigned to particular tasks because a users' absence blocks the process execution. Consequently, this assignment has to be used only for exceptional cases.

Separation of Duty. The annotations **separation of duty** (SoD) and **binding of duty** (BoD) specify dependencies of process activities. They constrain authorizations for a group of activities (or for multiple instances of an activity), to be performed by different individuals (SoD) or by the same one (BoD).

The annotation

$$\ll \textit{SoD: role} =\text{"}\ \$\textit{rolename}''\ \textit{number} = \text{"}\$\textit{value}''\ \textit{threshold} =\text{"}\ \$\textit{value}'' \gg$$

enforces separation of duty. We offer various possibilities for the specification: Without the optional parameter/value pair $role$ = "$\$rolename''$, it must be simply ensured that the annotated objects (typically activities) must be performed by different role holders. The specification $role$ = "$\$rolename''$ excludes role holders of *"rolename"* to perform the activities. With the optional parameters $number$ = "$\$value''$ and $threshold$ = "$\$value''$ (both of type integer) we specify the minimum number of different users (number) that have to perform activity/activities and a threshold value of the sum of activity instances a user is allowed to perform (these cardinalities are adapted from [15]).

We allow the "separation of duty" annotation (Table 1) for activities, groups of activities, pools, and lanes. A SoD constraint for several lanes forbids the allocation of a user to several roles. It holds for all activities of the pool or lane.

Example: A setting in the employability context where a SoD constraint is meaningful is as follows: A placement coordinator cannot carry out his own placement as learner (see Figure 1).

Binding of Duty. The concept of binding of duty constraints by the term

$$\ll \textit{BoD: spec} = \text{"}\textit{weak}'' \gg$$

that (typically) a group of activities will be performed by the same user. Such a restriction is desirable in order to obtain clear responsibility for a related set of activities. The annotation ≪ BoD ≫ is defined for activities, group of activities, and pools and lanes. Usually, a set of activities or a group of activities is specified by a BoD constraint. In the particular case, if only one activity is annotated, SoD constraints for loops or activity instances of a process are described. The default case is without the parameter *spec*. To avoid blocked process instances due to absence of process participants, we allow to ease BoD constraints by enabling the explicit transfer of responsibility *i.e., re-assignment* for activities. Note that we do not use the term delegation here, since delegation has a different meaning

later. If the optional parameter $spec =$ "$weak''$ is specified, a re-assignment of activities is possible, otherwise not.

To avoid process blocking because users are absent, we allow to ease BoD constraints by explicit transfer of responsibility, *i.e., re-assignment* of activities. The optional parameter $spec =$ "$weak''$ specifies this.

Example: The BoD annotations in Figure 1 mean that an individual learner cannot pass responsibility to carry out activities to other role holders of learner, while a re-assignment of activities of placement coordinators is possible.

Delegation. Process participants can delegate credentials to someone who currently does not have the authorization. In our context, delegation handles access rights to data which is external or is controlled by the same process; see the data handling in BPMN 2.0 [11]. Delegation varies depending on the category of the data concerned and therefore requires varying parameters. We see the following two categories: A *data object* is local to the process, and its lifetime is bound to the process where it is defined and used; *data stores* are external data sources which persist beyond the lifetime of a process. For instance, this could be a personal data store (PDS) or a database. For internal and external data, there might exist access restrictions. Rules to access *data stores* are often specified using external mechanisms, Access rights to *data objects* have to be defined explicitly within the BPMS. Using the data during the process can have the effect that changing role holders execute the tasks accessing the data, or that external services called need to access external data for using it a role holder is authorized but not the service itself. These are situations requiring delegation of access rights or credentials.

An example of how to use a data store is the following: The business process needs access to data in a PDS of a certain role holder. To this end, the role holder can specify a respective view on his PDS, together with an access policy, represented as a protected reference to that view and the policy, e.g., in form of a token. The process can then use this token to delegate the access rights to a web service invoked.

The annotation

$$\ll \textit{Delegation: } rights = \text{“}\$rightsname'' \ object = \text{“}\$objectname''$$
$$interval = \text{“}(\$activityname1, \$activityname2)|\$group'' \ role = \text{“}\$rolename''$$
$$poolname = \text{“}\$poolname''|\text{“}\$lanename'' \ ws = \text{“}\$webservicetypename''$$
$$spec = \text{“}\$specification'' \gg$$

specifies the delegation of access rights to data, with pre-defined vocabulary: *read*, *write*, *delete*, or *policy*. If there is a delegation of access rights which are attached to the data object in form of a policy, the "rights" parameter is optional, but in this case the "object" parameter is required. The following parameters are optional: resource, role, poolname, ws, spec.

The specified rights or the use of the object with a related policy are delegated. The rights describe which use of $objectname is allowed. The delegation is valid for the execution time of activities in the interval [$\$activityname1$, $\$activityname2$], i.e., the part of the process starting with $\$activityname1$ and ending with $\$activityname2$, or contained in the $\$group$ of activities. The delegate is either a *rolename*, a *poolname*, a *lanename*, or a *webservicename*.

Optionally, the parameter *spec* denotes a delegation with *grant* rights for the delegate.

Example: Figure 2 shows a process activity dealing with a personal data store. The annotation ≪ Delegation ... ≫ gives credentials for the "$View-PDS-of-Learner" from role "Placement coordinator" to the web service "PDS".

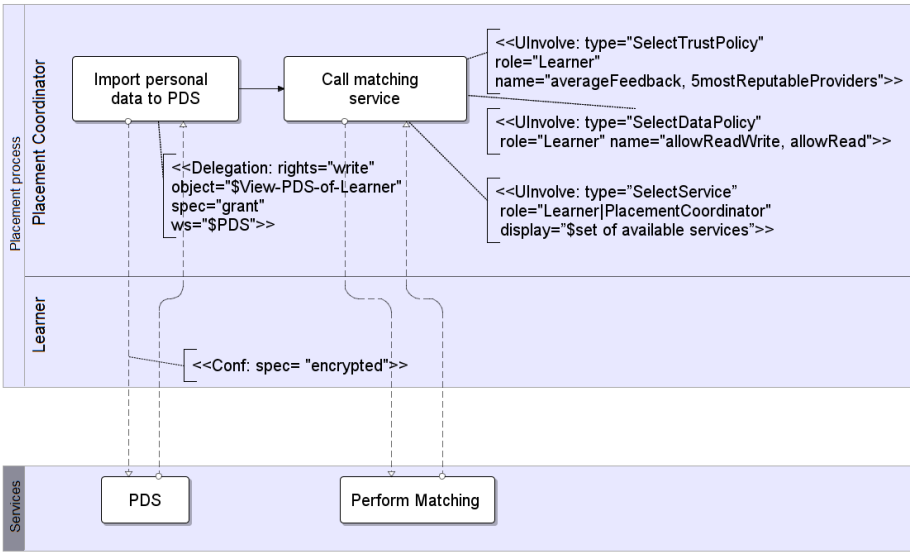

Fig. 2. Example Security Annotations for Process Activity

For the annotation of data, the rights *read*, *write*, and *delete* are possible. The right to write data can also be used to create data objects.

Adaptations. To give way to ad-hoc process changes, but in a controlled way, we allow the specification of rights to adapt process instances. (Annotation **adaptation** in Table 1). With

$$\ll \textit{Adaptation: rights} = \text{“}\$\textit{rightsname}\text{”}\ \textit{role} = \text{“}\$\textit{rolename}\text{”}$$
$$\textit{pre} = \text{“}\$\textit{precondition}\text{”}\ \textit{post} = \text{“}\$\textit{postcondition}\text{”} \gg$$

we give rights to adapt (i.e., to migrate) process instances. By means of the parameters $role$ = "$\$rolename$" and $rights$ = "$\$rightsname$" (the predefined vocabulary for rights is *insert*, *update*, *delete*) we specify which role holders have particular rights to adapt the process at the annotated position. Further, we specify pre- and postconditions with the parameters *pre* and *post* that must hold for process modifications. Consequently, role holders with the appropriate rights can trigger and migrate running process instances. For example, a student in an e-learning scenario wants to see who accessed his personal data, although the process does not support this activity. Or a coach might decide that a student

has to pass further exams, which are not foreseen yet. With the rights to adapt (migrate) the process, both users can adapt the process instance according to their needs. Adaptation rights are defined for any BPMN elements (activities, group of activities, pools and lanes, data, message flows, and events).

From the realization point of view, these activities are managed as abstract web services and are maintained typically as process fragments in particular repositories. When users want to plug-in non-foreseen process parts (single activities or sub-processes) into the process, they can select the appropriate process fragments accordingly.

In our approach, we additionally support a second way for process adaptations. The annotation of user involvements dealing with security and trust issues lead to modifications of the process model. We describe this in Section 3.5 and Section 4.3).

Table 2 summarizes the allowed authorization constraints.

Annotation Term	Roles	Activ.	Gr. of Activ.	Pools&Lanes	Data	Messagef.	Events
Role Assignment	-	X	X	X	-	-	-
Assignment Mechanism	X	X	X	X	-	-	-
User Assignment	-	X	X	X	-	-	-
Binding of Duty	-	X	X	X	-	-	-
Separation of Duty	-	X	X	X	-	-	-
Delegation	-	X	X	X	-	X	-
Adaptation	-	X	X	X	X	X	X

Table 2. Allowed Authorization Annotations

3.2 Authentication

The annotation $\ll$ *Authn: attribute= list*(“*attributename*″, “*value*″)
idp = “$*identityprovider*″ $\gg$
means that process participants must be authenticated and identified. The task of an identity provider (IDP) is to certify attributes of process participants. This certification leads to authentications. For this purpose, we specify a list of attributes which a particular identity provider (IDP) can certify. For instance, any holder of role “learner” in Figure 1 must be authenticated as a student at the University of Nottingham and identified by an identity provider given or a related one. The term ≪Authn≫ can be annotated for roles and pools and lanes (which will be interpreted as roles). In the first case, a twofold annotation (see discussions mechanisms assignment) is needed.

3.3 Auditing

The annotation $\ll$ *Audit: policy* = "$*policyname*'' $\gg$
enforces a monitoring of the execution of an activity, a group of activities, data access, event, or message flow.

Various legal requirements call for different logging activities. For example, if a process handles personal data, it must be logged, among others, who performed which action upon which data, and at which time [2]. Thus, we specify the objects to be logged in an auditing policy (parameter *policy* = "$*policyname*''). To illustrate, we have annotated the process "start event" with the annotation term *Audit* in Figure 1. This specifies a monitoring of consenting to terms and conditions at the process start according to the policy "auditAll".

3.4 Data and Message Flow Security

Confidentiality. Authorization mechanisms support a basic level of confidentiality, because only authorized humans or web services may perform activities and access data. To improve confidentiality further, the system can also protect message flows. This can be expressed by:

$$\ll \textit{Conf: spec="\$value"} \gg$$

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 2 must be encrypted.

Integrity. The annotation $\ll$ *Integrity* $\gg$
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.

Table 3 summarizes the usage of these annotation terms.

Annotation Term	Roles	Activ.	Gr. of Activ.	Pools&Lanes	Data	Messagef.	Events
Authentication	X	-	-	X	-	-	-
Auditing	-	X	X	X	X	X	X
Confidentiality	-	-	-	-	-	X	-
Integrity	-	-	-	-	X	X	-

Table 3. Allowed Annotations for other Security Goals

3.5 User Involvements

By studying real-world applications that handle personal data in particular, we have identified user involvements that are specific in the security and trust context. These user involvements are important, because users have to specify and agree at runtime to security preferences that are context-specific. Having examined various application scenarios, we have identified the following security-

and trust-specific user involvements: giving consent, selecting services according to required trust levels, specifying data access policies or trust levels, specifying users' interaction preferences (i.e., their preferred degree of direct involvement in security issues), and trust-feedback mechanisms. To support the representation of such involvements, we introduce annotation terms to specify them declaratively. Thus, the process designer simply annotates user involvements instead of modelling them explicitly. We now describe the annotation terms that represent security-specific user involvements.

The representation is as follows:

$$\ll \textit{UInvolve:list(parameter-name=value)} \gg$$

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. Table 4 lists our language for security- and trust-specific user involvements. Dots indicate further parameters, such as specification of users' interaction preferences, giving user consent or trust feedback, service selection, specification of a data policy, trust policy setting or selection. We can annotate activities and some events of a process model with user involvements. Unless specified otherwise, these involvements are invoked before the annotated activity takes place. To deal with other cases, there is a parameter *insertplace*, indicating the position of the involvement in the flow of the process.

User Involvement	Syntax
Set Interaction Preferences	≪UInvolve: type="SetIAPref" ... ≫
Give Consent	≪UInvolve: type="Consent" ... ≫
Service Selection	≪UInvolve: type="Select Service" ... ≫
Set Data Access Policy	≪UInvolve: type="SetDataPolicy" ... ≫
Select Data Access Policy	≪UInvolve: type="SelectDataPolicy" ... ≫
Set Trust Policy	≪UInvolve: type="SetTrustPolicy" ... ≫
Select Trust Policy	≪UInvolve: type="SelectTrustPolicy" ... ≫
Give Trust Feedback	≪UInvolve: type="TrustFeedback" ... ≫

Table 4. Overview of User Involvements

Set Interaction Preferences. The annotation of the first line in Table 4 specifies the interaction preferences of a user for a business process. These preferences may concern, for example, data access, or preferred interaction behavior concerning web-service selection.

With the annotation

$$\ll \textit{UInvolve: type} = \text{“}\textit{SetIAPref}\text{”}\ \textit{display} = \text{“}\textit{list}(\$\textit{option})\text{”}$$
$$\textit{role} = \text{“}\$\textit{rolename}\text{”}\ \textit{insertplace} = \text{“}\$\textit{activityname}\text{”} \gg$$

a user can specify its preferences with respect to interactions for a business process. These preferences may concern, for example, data access, or its preferred interaction behavior concerning web service selection. The parameters *role* and *insertplace* are optional.

The annotation is illustrated by the following example:

$\ll$ *UInvolve: type* = "$SetIAPref''$ *display* =“data: allowAll,allow individuals, allow none”; “web service selection: ask anytime, do not ask”. $\gg$

Example: A user selects his interaction preferences from the options “ask me anytime” or “do not interrupt me” in case that security issues occur during process execution.

User Consent. Due to legal regulations, user consent for privacy terms is needed when personal data is managed (Table 4). Process designers have this domain knowledge and can represent it accordingly. Process modeller have this application knowledge and can represent this requirement accordingly.

The annotation

$\ll$ *UInvolve: type* = "$Consent''$ *target* = “$targetname of privacy terms” *role* = "$\$rolename''$ *insertplace* = "$\$activityname''$ *display* = "$\$option_1, ..., option_n''$ $\gg$

specifies that a user must give consent to some privacy terms, otherwise the process ends. The *target* identifies the particular privacy terms that must be accepted by a user (type string), *display* specifies the displayed text of a user interface a user can choose for agreement.

For illustration, we give a brief example where a user has to choose between “yes” and “no” of the TAS^3 Privacy Terms:

$\ll$ *UInvolve: type* = "$Consent''$ *target* = "$TAS3PrivacyTerms''$ *display* = "yes, no''. $\gg$

Example: Figure 1 shows two annotated user involvements aiming to consent to the placement process as well as to privacy terms when the process starts.

Service Selection. The term

$\ll$ *UInvolve: type* = "$SelectService''$ *display* = "$list(\$option)''$ *insertplace* = "$\$activityname''$ *role* = "$\$rolename''$ $\gg$

means that the user who performs the annotated activity should first be asked to select a service with a specific trust level from the set of available services. A service-discovery mechanism using trust characteristics of services obtains the available services in advance. The result is shown to the user, in line with the variable *display* = "$list(\$option)''$. Optionally, the process modeller can specify a role name, stating that a holder of the role has to select the service. A prerequisite for service discovery is that a user has specified his trust policy (see below).

Example: Figure 2 contains three user involvements, one representing a service selection. A learner or a placement coordinator has to select one of the services available.

Data-Access Policy. When a process manages sensitive data, the data owner can select security policies for this data. This can be specified either by user involvements of type “Set Data Policy” or of type ”Select Data Policy”,

Set Data Policy. The annotation

$$\ll \textit{UInvolve: type} = \text{“}\textit{SetDataPolicy}\text{”}\ \textit{role} = \text{“}\$\textit{rolename}\text{”}\ \textit{name} = \text{“}\$\textit{policyname}\text{”}\ \textit{target} = \text{“}\$\textit{namesensitivedata}\text{”}\ \textit{insertplace} = \text{”}\ \$\textit{activityname}\text{”} \gg$$

sets a data policy. The parameters *role* and *insertplace* are optional.

$$\ll \textit{UInvolve: type} = \text{“}\textit{SetDataPolicy}\text{”}\ \textit{target} = \text{“}\textit{e}-\textit{portfoliodata}\text{”}\ \textit{name} = \text{“}\textit{allowAll}\text{”} \gg$$

gives the rights for e-portfolio data of a user.

Select Data Policy. The annotation term

$$\ll \textit{UInvolve: type} = \text{“}\textit{SelectDataPolicy}\text{”}\ \textit{target} = \text{“}\$\textit{nameofsensitivedata}\text{”}\ \textit{display} = \text{“}\textit{option}1, \textit{option}2, ..\text{”} \gg$$

enables that a user specifies a security policy for access to his sensitive data by selecting from a policy list. As parameters, we have target=“name of sensitive data” (type string) and display=“option1, option2,..” (of type list of strings). The following example illustrates the annotation:

$$\ll \textit{UInvolve: type} = \text{“}\textit{SelectDataPolicy}\text{”}\ \textit{target} = \text{“}\textit{e}-\textit{portfoliodata}\text{”}\ \textit{role} = \text{“}\$\textit{rolename}\text{”}\ \textit{insertplace} = \text{“}\$\textit{activityname}\text{”}\ \textit{display} = \text{“}\textit{denyAll}, \textit{allowRead}, \textit{allowAll}\text{”}. \gg$$

The user can decide whether all process participants are denied or allowed to access its e-Portfolio data. Process modeller can also consider the case that data policies might be too restrictive. In this case, potential process adaptation by means of a modification of data policies can be modelled.

Trust Policies. We characterize service providers by trust levels, representing their reputation gained in the past. Process participants can specify the minimum trust level of service providers by means of a trust policy. Users can either set (i.e., specify) or select trust policies interactively from a list (see lines 6 and 7 in Table 4).

Set Trust Policy. A trust policy will be set (i.e., initialized or modified) in a process instance by the annotation

$$\ll \textit{UInvolve: type} = \text{“}\textit{SetTrustPolicy}\text{”}\ \textit{name} = \text{“}\$\textit{policyname}\text{”}\ \textit{role} = \text{“}\$\textit{rolename}\text{”}\ \textit{insertplace} = \text{“}\$\textit{activityname}\text{”}. \gg$$

The parameter *name* identifies the trust policy to be used.

The following example illustrates the setting of the trust policy “trustAll”:

$$\ll \textit{UInvolve: type} = \text{“}\textit{SetTrustPolicy}\text{”}\ \textit{name} = \text{“}\textit{trustAll}\text{”} \gg$$

Select Trust Policy. The following annotation specifies the selection of a trust policy:

$$\ll \textit{UInvolve: type} = \text{“}\textit{SelectTrustPolicy}\text{”}\ \textit{display} = \text{“}\textit{list}(\textit{option})\text{”} \gg$$

For example, the following options can be selected by a user:

$$\ll \textit{UInvolve: type} = \text{“}\textit{SelectTrustPolicy}\text{”}\ \textit{display} = \text{“}\textit{trustAll}, \textit{averageFeedback}, \textit{mostreputableUsers}, 5\textit{mostreputableUsers}\text{”} \gg$$

Example: The user involvement of type "SelectTrustPolicy" in Figure 2 means that a learner has to select a trust policy from the available options.

Trust Feedback. The annotation

$$\ll \textit{UInvolve: type} = \text{“}TrustFeedback\text{”}\ \textit{display} = \text{“}list(option)\text{”}$$
$$\textit{role} = \text{“\$}rolename\text{”}\ \textit{insertplace} = \text{“\$}activityname\text{”} \gg$$

means that users who hold the role calling the web service are asked for feedback about its trustworthiness. Here, we have parameter *insertplace*. It specifies where exactly in the process model users give feedback. The parameter role is optional.

Example: The annotation

$$\ll \textit{UInvolve: type} = \text{“}TrustFeedback\text{”}$$
$$\textit{display} = \text{“}positive, neutral, negative, nofeedback\text{”}$$
$$\textit{insertplace} = \text{“\$}activityname\text{”} \gg$$

means that users can pick a feedback value from the options listed. This user involvement will be triggered before activity *activityname* executes.

Table 5 summarizes how user involvements can be annotated in BPMN diagrams.

User Involvement	Roles	Activ.	Gr. of Activ.	Pools&Lanes	Data	Messagef.	Events
Set Interact. Preferences	-	X	X	-	-	-	X
Give Consent	-	X	X	-	-	-	X
Service Selection	-	X	X	-	-	-	-
Set Data Access Policy	-	X	X	-	-	-	X
Select Data Access Policy	-	X	X	-	-	-	X
Set Trust Policy	-	X	X	-	-	-	X
Select Trust Policy	-	X	X	-	-	-	X
Give Trust Feedback	-	X	X	-	-	-	-

Table 5. Allowed Annotations for User Involvements

In summary, our language represents security aspects and security- and trust-specific user involvements. Having taken into account all requirements from two complex, realistic application scenarios, we argue that our language is the result of a more comprehensive design effort than previous ones.

4 Transformation of Security Annotations

In this section, we describe how we transform our security annotations of the BPMN model to the execution level. A BPMS enhanced with BP-specific security

components, see Section 4.1, executes the resulting BPMN model. Depending on different constraint types, we have identified three different kinds of targets and explain the specifics of the respective transformations.

4.1 Secure Business-Process-Management Systems

A common security framework, not confined to business processes, is the XACML framework [10]. It is a declarative access-control policy language implemented in XML. The processing model of XACML contains the following components: a policy-decision point (PDP), to check against a (PDP-)specific security policy 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. A security framework of a heterogeneous environment additionally requires a federated identity framework with Identity Providers (IdP), to identify and authenticate users. Finally, further components, e.g., to manage trust with a trust PDP, or to support auditing by means of an auditing bus, are needed.

The BP-specific security requirements of Section 2 necessitate the refinement of the components of the XACML processing model. We refer to the new components as BP-PEP, BP-PDP and BP-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, assignment of users to activities must take the history into account; this also holds for authorization constraints, such as BoD and SoD. The use of credentials is essential for service calls which the process performs for a specific user. In this case, the business process must delegate the necessary rights to the service called, e.g., to access the external data store.

The right-hand side of Figure 3 shows the main components of a BPMS at the execution level. according to the OASIS standard of the workflow management coalition [17]: the *business-process engine*, the *tool agent* invoking external applications, e.g., web services and other processes. and the *BP user interface* with a worklist handler and binding of client applications, i.e., typically web interfaces to users. The *administration and monitoring tool component*, which is typically a BP-external component, and the design component, see Figure 4, are not part of this figure. To give way to security-specific functionality, we have modified the BPMS architecture as follows: A dedicated security component, the BP-PEP (business-process-specific policy-enforcement point), handles all interactions with users and with external web services, as a proxy of the BP engine. With the help of components of the security framework, the BP-PEP identifies and authenticates each BP user, and it calls web services in a secure way. The BP-PEP first checks if access rights are sufficient and if the transport has to be encrypted.

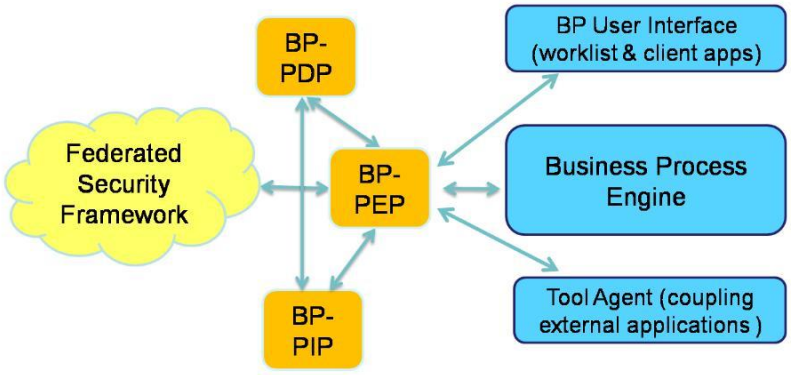

Fig. 3. Architecture of a secure BPMS

4.2 Business-Process-Security Modelling and Configuration Component

Figure 4 shows the *three phases* of the life-cycle of secure processes. The *design phase* results in a BPMN model with security annotations. The annotations containing security constraints are BPMN artifacts. To deal with security annotations, we have developed a dedicated tool, the security modelling and configuration component for BP (BP-SMC). It carries out the *transformation*, as follows: It takes the annotated BPMN model of the design phase, extracts the security annotations, and transforms it into an adapted BPMN model and additional security information. This is used to configure the BP-specific security components. In the *execution phase* the secure BPMS runs instances of the resulting process, enforcing the security constraints. In the following, we describe

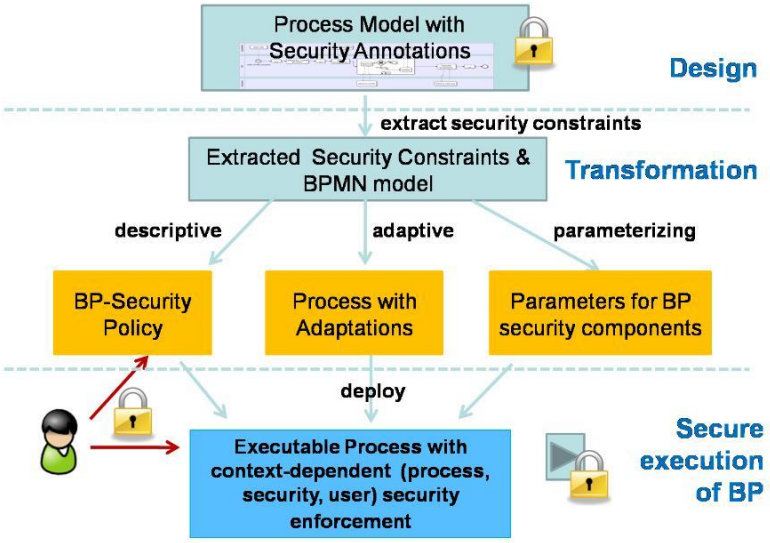

Fig. 4. Three Phases of Life-Cycle of Secure Business Processes

the transformation phase in more detail.

4.3 Transformation Specifics

The transformation of an annotated BP model aims at generating the security configurations. In a preprocessing step, the BP-SMC tool extracts the security constraints of the annotations and the specific process context. This generates Java objects of the process-model elements which are queryable and updatable. Further, the BP-SMC tool handles conflicts of security annotations. Such conflicts may occur due to contradictory annotations, e.g., role specifications of a pool and of activities of the pool that are not compatible.

The main concern of this phase is the transformation of the security annotations to representations which support the enforcement of the constraints during process execution. We have observed that different security constraints on business processes need to be transformed in different ways, i.e., have different outcomes. 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,
- adapting the business process with process fragments modelled a priori.

BP-security policy: A security policy consists of security rules, such as access rules or rules protecting the data flow. Business processes control the flow of tasks. Thus, general security-policy languages, for which XACML is a standard, 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 in short as BP-security policy.

A PDP (see Section 4.1) is a standard component in security frameworks to ensure access restrictions. Therefore, we argue that one should map these security annotations to such policies.

The transformation in this category results in an access-control security policy in XACML, enhanced with business-process specific constructs: In particular, security constraints may refer to tasks and relationships between tasks, as well as to references to the execution history and the status of the business process. During process execution, the BP-PDP checks if any access to tasks and data conforms to those rules. Taking the execution history into account requires a PDP more flexible than the ones currently available.

Example: Think of a BoD constraint that is part of the BP-security policy. It 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 results in additional rules (in this case an extension of XACML) that become part of the BP-access-control security policy:

$$\begin{array}{c} < TaskGroupSettings > \\ < TaskGroupSpecTargetID = \text{“}PlacementCoordinatorLane\text{”} \\ BoD = \text{“}true\text{”} weakBoD = \text{“}true\text{”} / > \\ < /TaskGroupSettings > \end{array}$$

When assigning a holder of role "placement coordinator" to a human task, the BP-PEP calls the BP-PDP to check if the assignment fulfills the access rules of the BP-security policy, e.g., whether a BoD constraint has to be fulfilled.

Configuring security components: The second kind of annotation concerns the parameterization of calls of security-specific components other than the BP-PDP. The BP-specific security components, in particular, BP-PEP or BP-PIP, take in account process context like validity periods of security rules, history of the process flow, or role holders involved in the process. E.g., the context of role holders consists of properties such as security preferences, policies or identifying attributes from the IdP.

Specific activities of the process model invoke the security components via the BP-PEP. These activities are calls going to or coming from external components or human activities, as already described in Section 4.1. The context of those security calls makes it necessary to provide information about the status and the history of the process instance. The BP-PIP provides this information. The BP-SMC inserts information taken from those annotations into the BP-PIP, to facilitate parameterization of security calls issued by the BP-PEP. This procedure follows the proposal of security handling in the XACML standardization.

If a component of the security framework enforces constraints of an annotation, its transformation determines parameter values for the component.

Example: ≪Authn: ...≫ will result in an authentication call to an IdP. The annotation and its context information yield the identifying attributes of the user in question. As another example, think of a ≪confidentiality≫ annotation of a web-service task. It specifies parameters to invoke the web service in an encrypted manner or with a digital signature, see Figure 2. In consequence, the BP-PEP can set WS-Security-compliant parameters for the WS call, to induce an execution that is encrypted or digitally signed.

Process adaptation: The third kind of annotation needs an explicit implementation to enforce it. The implementation is embedded into the business-process activities, i.e., on the application level, and may involve the actors. In particular, this is the case for ≪UInvolve≫ annotations which require also interactions with users. Another example is the ≪Audit≫ annotation. It results in a BP task calling the audit bus to log access to the annotated BPMN element, e.g., the execution of a task or a user interaction.

Our approach regarding this type of annotation is to adapt the process model by plugging process fragments into it. Such a process fragment includes security-specific activities. For instance, for each ≪UInvolve≫ annotation, there usually is more than one fragment. The variants depend on the required behaviour of the fragment. Adapting the process model occurs before any process instance starts. But we also provide a certain degree of runtime flexibility allowing to plug in several branches of security handling, if there are alternatives specified in the annotation. The choice of the branch depends on user preferences, set during process execution or according to an existing user preferences policy. For example, the service-selection annotation in the process part of Figure 2 shows two alternative actors responsible for service selection, the Learner or the

Placement Coordinator, which results in different process fragments. Using predefined process fragments brings several benefits: The process designers do not need to be familiar with the specifics of security components. Also, providing process fragments supports the correct implementation of the annotations.

Example: In the process of Figure 2, a Learner might allow trusted web services to match her personal e-portfolio data with job offers only to trusted web services. But there, this activity (i.e., the service call) is only specified without any trust limitations. To take trust into account for the service selection, the process designer now can annotate the activity with a ≪UInvolve≫ constraint of type "SelectTrustPolicy". The annotation specifies that the Learner can select one of the options listed in the display parameter:

$$\ll \textit{UInvolve: type} = \text{“}SelectTrustPolicy'' \; display = \text{“}trustAll, averageFeedback, mostreputableUsers, 5mostreputableUsers'' \gg$$

The process fragment for this user involvement contains the following activities and user interactions:

- A user interaction, i.e., new tasks to prompt the user and to receive his input, e.g., the trust policy selected. The web form must contain (links to) the trust policies available. They have to be parameters of the annotation.
- The BP-PEP calls the web-service discovery with the trust policy selected as parameter. It returns only web services that are trusted according to the trust policy.
- In case there is no web service with that trust level, there is a further iteration of trust-level selection and service discovery.

5 Related Work

Wolter et. al. [15], [16], [14] and Menzel et. al. [8], [7] have presented work on modelling security for business processes in SOA. [15] focuses on authorization constraints in BPMN artifacts, but does not cover delegation of data-access rights, weak BoD, and direct user assignments. Authorization is limited to human tasks. [16] describes a model-driven transformation of security annotations into XACML. It allows to relate security annotations to security components. This is an approach similar to our transformation of annotations to configuration parameters. The creation of an XACML policy in [16] does not include user involvements. There also is no counterpart to our transformation resulting in an adapted process model. [14] extends [15] by a broader security vocabulary. It also comprises aspects such as integrity and confidentiality. [8] annotates inter-organizational processes with trust. We have another understanding of trust: Users specify their desired trust level of web services interactively. The model-driven approach in [8] translates security constraints into security configurations in SOA. However, there does not seem to be an implementation for the execution of processes. [7] defines a process-independent policy language, and provides a mapping to a design model for security constraints.

[6] 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 business processes and an RBAC (role-based access control) WS-BPEL specification with support of human activitites, 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; they also do not consider BPMN. [12] proposes a broad security vocabulary for BPMN artifacts. However, it lacks authorization constraints (such as SoD), and it is less detailed than our language. A model-driven approach transforms security constraints into implementation specifications [13].

Our language is the first to deal with security-specific user involvements. [3] represents the control of user dialogs in BPMN. Our work is unique in that we bundle user interactions, the security-specific tasks and the flow between them. Finally, our transformation approach differs from existing approaches ([13], [16]) because we have identified three different ways to transform security constraints.

6 Conclusions

We have presented a rich language to represent security constraints for business processes and provide security support from the modelling to the runtime phase of a business-process lifecycle. On the modelling, configuration and execution layer, we have extended an open-source BPMS. The BPMS extensions transform BPMN schemas with security constraints into executable secure processes in a versatile manner.

The language supports "classical" security aspects, but also trust- and security-specific user involvements. Thus, it represents a comprehensive set of security features for business processes in SOA, and is embedded into a process-modelling language, i.e., BPMN. We are using so-called artifacts as containers for constraints. Thus, our approach is BPMN 2.0 standard conform at a syntactic level.

By describing how to transform our security constraints, we have demonstrated that there are three different kinds of targets: security policies at an abstraction level comparable to the one of XACML, adaptations of the process schema, and parameter settings for invocations of security components. A PDP allows to check the execution of process tasks and the use of process data against the resulting access control policy of the business process. To enable security constraints beyond access control rules, we enhance the process model with calls to security components. Particularly, user involvements are challenging in a more and more user-centric application environment, e.g., to adapt user preferences related to security. For these complex activities with user interactions, we have introduced process adaptations based on pre-defined process fragments.

Acknowledgements

This research has received partial funding from the Seventh Framework Programme of the European Union (FP7/2007-2013) under grant agreement n° 216287 (TAS^3 - Trusted Architecture for Securely Shared Services)[1].

References

1. Integration with TAS^3 trust applications of employment and eHealth processes. TAS^3 Deliverable 3.3, 1st Iteration (December 2009)
2. Alhadeff, J., Alsenoy, B.V.: Requirements: Privacy, governance and contractual options. TAS^3 Deliverable 6.1 (December 2009)
3. Auer, D., Geist, V., Draheim, D.: Extending BPMN with submit/response-style user interaction modeling. IEEE Int. Conf. on E-Commerce Technology pp. 368–374 (2009)
4. Backes, M., Pfitzmann, B., Waidner, M.: Security in business process engineering. In: Proc. of the Int. Conf. on Business Process Management. pp. 168–183. BPM'03, Springer-Verlag, Berlin, Heidelberg (2003)
5. Bertino, E., Byun, J.W., Kamra, A.: In: Petkovic, M., Jonker, W. (eds.) Security, Privacy, and Trust in Modern Data Management, chap. Database Security, pp. 87–101. Springer Berlin/Heidelberg (2007)
6. Bertino, E., Martino, L., Paci, F., Squicciarini, A.: Security, Privacy, and Trust in Modern Data Management. Springer Berlin/Heidelberg (2010)
7. Menzel, M., Meinel, C.: SecureSOA - modelling security requirements for service-oriented architectures. In: IEEE Int. Conf. on Services Computing,. pp. 146–153. IEEE Computer Society, Los Alamitos, CA, USA (2010)
8. Menzel, M., Thomas, I., Meinel, C.: Security requirements specification in service-oriented business process management. In: ARES. pp. 41–48. IEEE Comp. Society (2009)
9. Müller, J., Mülle, J., von Stackelberg, S., Böhm, K.: Secure business processes in service-oriented architectures – a requirements analysis. In: Europ. Conf. on Web Services. pp. 35–42. IEEE Computer Society, Los Alamitos, CA, USA (2010)
10. OASIS: eXtensible Access Control Markup Language tc v2.0 (XACML) (February 2005)
11. Object Management Group: Business Process Model and Notation, V2.0. OMG Available Specification (January 2010), `http://www.omg.org/spec/BPMN/2.0/PDF`
12. Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN extension for the modeling of security requirements in business processes. Trans. Inf. Syst. – IEICE E90-D, 745–752 (March 2007)
13. Rodríguez, A., de Guzmán, I.G.R., Fernández-Medina, E., Piattini, M.: Semi-formal transformation of secure business processes into analysis class and use case models: An MDA approach. Information and Software Technology 52(9), 945 – 971 (2010)

[1] The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

14. Wolter, C., Menzel, M., Meinel, C.: Modelling security goals in business processes. In: Modellierung'08. pp. 197–212 (2008)
15. Wolter, C., Schaad, A.: Modeling of task-based authorization constraints in BPMN. In: Business Process Management. pp. 64–79. Springer Berlin / Heidelberg (2007)
16. Wolter, C., Schaad, A., Meinel, C.: Deriving XACML policies from business process models. In: Weske, M., Hacid, M.S., Godart, C. (eds.) Web Information Systems Engineering (WISE) Workshops, LNCS, vol. 4832, pp. 142–153. Springer Berlin/Heidelberg (2007)
17. Workflow Management Coalition: The Workflow Reference Model (1995)