Responsibility-driven Design and Development of Process-aware Security Policies

Maria Leitner,Stefanie Rinderle-Ma
University of Vienna
Vienna, Austria
{*maria.leitner,stefanie.rinderle-ma*}*@univie.ac.at*

Juergen Mangler
SBA Research
Vienna, Austria
jmangler@sba-research.org

***Abstract*—Process-Aware Information Systems (PAIS) enable the automated support of business processes that are executed by a combination of human actors and systems. As processes typically require access to sensitive data, security policies are of high importance. Typically security policies in PAIS range from access rules and authorization constraints to context policies (location, time) and are scattered over the multitude of heterogeneous PAIS components, i.e. process models, repositories, organizational structures, etc. Currently, different approaches for modeling and enforcing security policies exist that assume a set of explicitly defined security policies. Because of aforementioned heterogeneity, these approaches are suboptimal for PAIS. In order to improve upon existing approaches we present a security policy data model and design methodology, based on the concept of responsibilities, permissions and constraints. The goal is to not only unify diverse security policies in different PAIS subsystems, but also to make security policies independent of these subsystems to restrain complexity from process modeling and evolution, and to allow for comprehensive security policy development and maintenance.**

***Keywords*-Security Policy Design; Security Policy Development; Process-Aware Information Systems**

I. Introduction

Process-Aware Information Systems (PAIS) support the automated execution of business processes carried out by various actors and manage private and publich data (e.g., a patient record in a hospital). Therefore, security in PAIS is an essential factor for their successful application in practice. Currently, commercial systems such as Staffware, Websphere MQ Workflow or AristaFlow as well as research prototypes (e.g., YAWL) offer role-based access control mechanisms that connect process activities and organizational structures via so called access rules [1]. Further, the definition of authorization constraints such as separation of duties on top of the access control mechanisms is essential in many practical scenarios. Finally, access rules and authorization constraints might be enriched with various kinds of context information [2]. Of particular importance here are time and location. Think, for example, of a security policy stating that the patient records should be accessed by a physician only during the day.

Access rules and authorization constraints imposed over a process, potentially enriched by context, are process-relevant security policies. As defined in literature [3] business processes are secure, if all security policies imposed over the process are fulfilled. Different approaches for modeling and enforcing different security policies exist [4] that assume a set of explicitly defined security policies. Typically this assumption cannot be made for PAIS, since policies in general, and security policies in particular might be scattered over all different kinds of components of the PAIS, i.e., process models, repositories, or organizational structures. Further, they might even be integrated within the control flow structure of a process model. One example is the decision who has to sign a certain document modeled as an alternative branching within the process. This existing mix of representations and implementations for security policies in PAIS hampers their enforcement, consistency checks between the policies, as well as maintenance and evolution of the PAIS.

In this paper, we claim that security policies and processes must be separately designed from each other. This independence offers many advantages: first of all, security policies can be completely stored and maintained within one policy base. This enables consistency checks as well as maintenance and evolution of the policy repository. To achieve independence, we present a new security policy data model based on responsibilities and permissions to cover structural as well as operational aspects of the process. By doing so the separation of both aspects can be achieved and the relations between process and policies can be expressed by an explicit mapping. If now changes of either the process or policies occur, the side effects can be easily handled within the mapping. Additionally, we show how necessary information can be acquired and evaluate our approach based on a requirements analysis. The approach presented in this paper establishes a new model for developing security policies for PAIS.

Sect. II gives an overview of security policies in PAIS. In Section III, security policy design requirements are presented. Sect. IV gives an overview on policy acquisition. Then, the design (cf. Sect. V) and mapping of security policies (see Sect. VI) are presented. We discuss related work and evaluate our approach in Sect. VII and conclude in Sect. VIII.

II. Security Policies in PAIS

Security policies are a set of principles that control which subject is allowed to access which object within an information systems [5]. Such access restrictions are often defined based on roles (e.g., head of group) or job functions. In PAIS, however, security policies require a more detailed definition due to the multi-faceted characteristics of such systems. Specifically, security policies in PAIS might relate to access control, control flow, information flow, data integrity, and availability. The three parts of Fig. 1 each show the same **Travel Request** example, with the following basic structure. `Fill out travel request` requires that an *employee* has to fill in the required information for the business trip consisting of personal information (e.g., name), travel information (e.g., start and end date), budget information, signature, and date of signature. Subsequently, two signatures on the travel request are required, which can be done in parallel. `Sign travel request` implies that two superiors have to approve the travel request: the *head of group* has to approve the necessity of the trip (and absence of work) by signing the request. The *budget owner* has to confirm the travel request by verifying the financial coverage as well as approving the trip and advances of travel expenses. Both superiors have to authorize the request with their signature and date. Finally, the travel request is archived (activity `Archive travel request`) by *administrative staff*.

As depicted in Fig. 1, we differentiate between three different ways of representing security policies in PAIS:

(a) As part of the process logic as shown in Fig. 1(a).
(b) Attached to tasks as shown in Fig. 1(b) (is the most common approach [6]).
(c) As separate group of annotations, loosely connected to a process as shown in Fig. 1(c). This implies that the security model includes knowledge of the process structure (sequence of tasks) that is independent of the process model.

From a technical point of view, most approaches dealing with authorization constraints such as [4], define security policies as explicit policies that are stored in a repository. This approach is valid for representations b and c, while the granularity of policies is different. For example for representation b a security policy has to include a reference to a specific task which requires knowledge about a specific process modeling notation. On the other hand, it is necessary to establish this connection between the tasks and security policies during enforcement for representation c (which requires some additional reasoning and or mapping information).

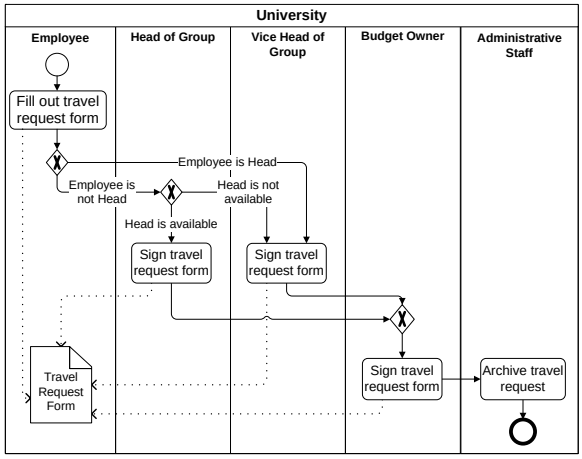
III. Security Policy Design Requirements

As described above, there are different approaches for security policies in PAIS. The most important goal of our approach is to operate independent from specific process models order process modeling notations. In order to achieve this goal we define the following requirements:

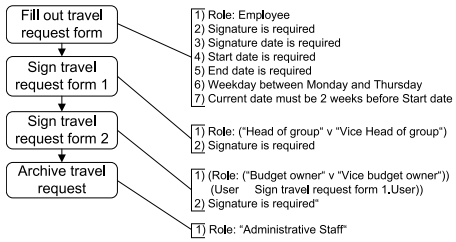
Requirement 1 (Independence): Security polices should not be be intertwined with process logic. By choosing the representation shown in Fig. 1(c) the process model stays simple and security and process design can be kept separately.

Requirement 2 (Maintainability): Security policies should be easy to maintain (e.g., add, change, delete). This includes their reuse in multiple process models and activities. When changing a security policy it should not be necessary to change multiple connections to process models, or even worse, to redesign processes models.

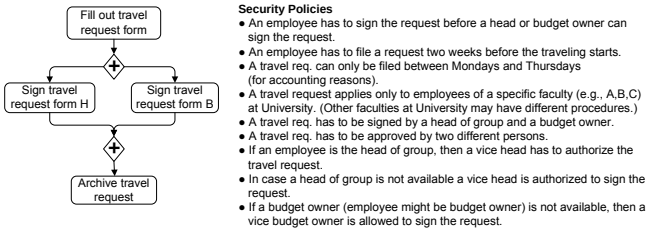
Requirement 3 (Extendability): Due to constantly changing business environments, changing process models and security policies is very important. We think that this ex-



(a) Inherent Representation



(b) Attached Representation



(c) Separated Representation

Figure 1. Travel Request: Process Modeling and Security Policies

tendability should be decoupled. Changing processes models should require no security policy knowledge and vice versa. Instead the extendability should be fostered by providing tool support to warn of violations.

Requirement 4 (Scalability): Scalability is about the ability to handle growth. In PAIS, scalability means managing an increase of components such as activities or security policies. Of course this also affects *Maintainability*. Scalability demands for both, a well structured process repository as well as authorities (staff roles) responsible for diverse aspects (e.g. process modeling, security policy design, conflict resolution).

IV. Security Policy Acquisition

Acquisition is an important part in the process of designing security policies. Many techniques exist for the accruing process such as process mining to derive process models (e.g., [7], [8]), role engineering [9], [10] and role mining [11] to establish an organizational model. Other methods for the acquisition are for example, interviews with employees about the correlations between their job and their work tied to other employees, or evaluating the organizational model, internal guidelines or national law. The acquisition includes but is not limited to the following topics: organizational structure, job functionalities (roles), permissions, authorization rules, authorization constraints (e.g., time, location), control flow of tasks, and information flow. The results of the acquisition are:

Process Acquisition: The resulting view counters on structural aspects such as activities, and the control flow and data flow between them.

Role Acquisition: The outcome focus on the organizational aspects such as structure and job functionalities.

Security Acquisition: The result is user centric. If focuses on the properties and restrictions imposed on the work relations with other users. This may incorporate structural and operational aspects.

It is important to note, that the information about the sequence (or structure) of activities, temporal relations between the activities, and correlation of data elements to activities, are present in the processes as well as in security policies. This duplication is well desired, as the security policies are not only used to enforce secure process models at *design time* (e.g., if certain data elements are only to be accessed by certain users and an activity uses data elements that are only allowed to be accessed by different users, then there is a static security violation in the model) but also at *runtime* such as separation or binding of duty constraints.

V. Security Policy Design

In order to ensure the primary requirements of independence and maintainability it is vital to separate process related information and security related information. From a security standpoint each process can be viewed from two

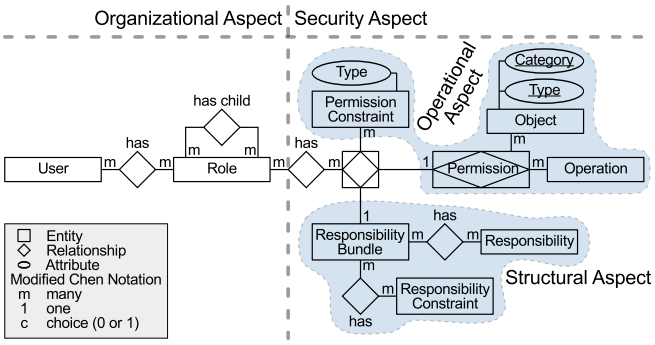


Figure 2. Security Policy Data Model

perspectives: (1) the process has a structure (process model), and (2) the process is subject to operations. Point (2) not only includes the instantiation and execution of a process, but may also include changes on the process structure, execution or monitoring of individual tasks in the process, or dynamic service selection. Operational security always depends on structural information, in that the operations always refer to certain process models and/or a set of tasks. In order to cater to this duality, we further distinguish between structural and operational security aspects:

Structural Aspect denotes a set of data objects and tasks, and how they occur in a process model.

Operational Aspect denotes constraints on this data objects and tasks, for example during process execution. I.e., under which circumstances something is allowed.

Please note, that while the structural aspect deals with names of data objects and tasks also covered by processes, it is nonetheless intended to be completely independent of processes. The structural aspect just deals with a set of names, further called responsibilities, classified as either data object or task. This set can be directly derived from information collected through a process acquisition (cf. Sect. IV). The structural aspect serves as a basis for the operational aspect, and is then mapped to the actual actual data elements and activities of available processes (as described in Sect. VI).

A. Responsibilities

As depicted in Fig. 2, the structural aspect includes responsibilities. We define a responsibility r to be a piece of data (e.g. a document, an information) or *interrelated tasks* from the point of a certain role. The concept of responsibility does not include the notion of operation such as *execute*, *monitor*, or *access*. The purpose of responsibilities is solely to comprehend data and interrelated work tasks as objects, that can be constrained and assigned to roles in an organizational structure (as depicted in Fig. 2).

In a large organization with many roles, a deep hierarchy and many responsibilities, responsibilities may occur for several roles on several hierarchy levels. Also responsibilities do not exist separately, but are related to each other: e.g. several data objects and interrelated tasks belong together.

Therefore, we introduce the concept of responsibility bundles b that serve two purposes:

Constriction: Grouping allows to define constraints regarding the order of tasks, the existence of data regarding certain order of tasks, separation / binding of duty related to data and order of tasks, as well as security constraints (operations allowed regarding data and structure on certain circumstances). For more details please see Sect. VI.

Assignment: A group of responsibilities including a set of constraints may be assigned to several roles. The creation of bundles fosters structure and reuseability. Assigning each responsibilities to multiple levels of an organizational hierarchy and constraining them separately would arguably increase the workload for administrators and foster errors. Whereas an inheritance (with the necessity of multiple inheritance between roles and users/roles) would introduce all the complexity known from object oriented programming.

For a given organization there exist a set of responsibility bundles $\mathcal{B} = \{b_1, \ldots, b_n\}$. A responsibility bundle b contains a set of responsibilities $\mathcal{R} = \{r_1, \ldots, r_n\}$ and a set of constraints $\mathcal{RC} = \{rc_1, \ldots, rc_n\}$ referring to a subset of $\mathcal{R}$.

In order to exemplify the concepts involved in a responsibility bundle we introduce a simple travel request example $b_{\text{travel request}}$ (cf. Sect. II, Fig. 1(c)). In this case, a responsibility bundle contains $b_{\text{travel request}} = \left\{\{r_1, \ldots, r_9\}, rc_1^{tp}, rc_2^{r}\right\}$ a set of responsibilities $\mathcal{R}$ and responsibility constraints $\mathcal{RC}$. The following responsibilities are included:

r_0^{data} : start date
r_1^{data} : end date
r_2^{data} : signature employee
r_3^{data} : signature employee date
r_4^{data} : signature approval a
r_5^{data} : signature approval b
r_6^{data} : additional unspecified data
r_7^{task} : filling form and signing it
r_8^{task} : approval A
r_9^{task} : approval B

($b_{\text{travel request}}$, Responsibilities)

Example ($b_{\text{travel request}}$, Responsibilities) displays a set of responsibilities $\mathcal{R} = \{r_1, \ldots, r_9\}$ where r_n^{data} refers to a data object and r_n^{task} to a task. Furthermore, each responsibility can be restricted with responsibility constraints.

There are two categories of **Responsibility Constraints** $\mathcal{C}$: *Responsibility Task Pattern Constraints* rc^{tp}, and *Responsibility Relation Constraints* rc^{r}. First, the responsibilities have to be constrained regarding the order (pattern) in which the tasks may occur. We further call this class of constraints **Responsibility Task Pattern Constraint** rc^{tp}. We describe these patterns as Linear Temporal Logic (LTL) expressions [12]. For the $b_{\text{travel request}}$ example the pattern is as follows:

$$rc_1^{tp} : \Box(r_7^{task} \rightarrow ((\Diamond r_8^{task} \rightarrow \Diamond r_9^{task}) \vee \Diamond r_9^{task}))$$

($b_{\text{travel request}}$, rc_1^{tp})

This can be read as: r_7^{task} is eventually followed by r_8^{task} and r_9^{task} or task r_9^{task}.

In the second step, we define the relation between the data responsibilities r^{data} and the responsibility task pattern constraints rc^{tp}. It is important to note that unlike for processes there is no assignment of data to tasks necessary. Because the purpose is not process execution, but rather providing a basis for process consistency checking and secure resource (user) allocation. We call this type of constraints **Responsibility Relation Constraint** rc^{r}. For $b_{\text{travel request}}$ the constraint is:

$$rc_2^{r} : r_0^{data} \wedge r_1^{data} \wedge r_2^{data} \wedge r_3^{data} \wedge r_4^{data} \wedge r_5^{data} \wedge r_6^{data} \wedge rc_1^{tp}$$

($b_{\text{travel request}}$, rc_2^{r})

More complex rc^{r} relations may define that certain data responsibilities occur only for certain task patterns. Please note that the inclusion of r_6^{data} is important for the mapping as described in Sect. VI. Because it allows to define that additional (but unspecified) data elements may be present for the given pattern rc_1^{tp}. This way, redesigning the travel request process (e.g., including additional data elements) is independent of the responsibilities and constraints.

Regarding the checking of **Responsibility Bundles** against processes, it is important to note, that if a bundle holds multiple rc_r relations, only one of the rc_r relations has to match.

B. Permissions

In contrast to the structural aspect, the operational aspect deals with permissions and constraints on these permissions. Permissions constraints refer to the responsibility constraints (as defined above) and are closely interrelated (refer to each other). In other words: they use the responsibilities defined in a responsibility bundle b, and restrict them in a certain process related security context (permission).

We define an operational context $\mathcal{O} = \{p, \mathcal{PC}\}$ to consist of a single permission p and a set of permissions constraints $\mathcal{PC} = \{pc_1, \ldots, pc_n\}$.

Permissions p define which *operations* (execute, monitor) are allowed for which security *objects* (process execution, process model change, service selection). They apply *only* to an entire responsibility bundle. Thus, a permission describes the *situation* in which the permission constraints are checked. It is important to note that one permission is further constrained by set of *Permission Constraints* as explained below.

As the permission describes "the situation", not all of the following constraints make sense for each possible permission. E.g., data permissions constraints pc^{d} (see below) are not used when checking process model change (which affects the order of tasks).

We identified the following four classes of permission constraints:

Data constraints pc^d to restrict certain data responsibilities r^{data} according to their value.

Time constraints pc^t to restrict certain task responsibilities r^{task} according to time they may occur.

Location constraints pc^l to restrict certain task responsibilities r^{task} according to location of an assigned resource (user).

Separation/binding constraints pc^{sb} to define that different/same resources (users) have to be assigned. They can only occur in relation to responsibility task pattern constraints rc^{tp}.

To exemplify the relation between permissions, permission constraints and responsibilities, we created the following two examples, connect to our travel request use case. The first example ($b_{\text{travel request}}$, permission 1) defines a permission that restricts how a user may file a travel request.

$$\begin{aligned} pc_3^p &: \text{control flow}, r_7, \text{execute} \\ pc_{3,0}^d &: r_0 - r_3 > 2 \text{ weeks} \\ pc_{3,1}^t &: \text{Monday till Thursday} \\ pc_{3,2}^l &: \text{Faculty of C} \end{aligned}$$

($b_{\text{travel request}}$, permission 1)

As mentioned above, pc_3^p describes a situation. In this case, the right to execute activity r_7 is granted during process execution. This is related to pattern given in rc_1^{tp} and includes the right to access and change all data elements as defined in rc_2^r (as long as they are available during process execution for this activity, which is defined in the process). In this case, this bundle is intended to be linked with *every* role, as every employee can make a travel request. The constraints are described as follows: $pc_{3,0}^d$ describes that the travel request has to be filed two weeks in advance. $pc_{3,1}^t$ describes that the travel request can only be made between Monday and Thursday (because of internal resource planning reasons). $pc_{3,2}^l$ describes that the right to file a travel request applies only to employees from a certain faculty (as e.g. employees from other faculties use different procedures).

The second example ($b_{\text{travel request}}$, permission 2) deals with the approval of the travel request:

$$\begin{aligned} pc_3^p &: \text{control flow}, r_8, \text{execute} \\ pc_{3,0}^{sb} &: (r_7 \neq r_8) \wedge (r_8 \neq r_9) \end{aligned}$$

($b_{\text{travel request}}$, permission 2)

pc_3^p describes the execution of r_8 which is intended for the role of group leaders. $pc_{3,0}^{sb}$ denotes that if the user that filed the travel request r_7, or signed the approval r_9, is not allowed to sign the approval r_8. This does not exclude that the user that filed the travel request, signs the second approval, e.g. the head of group.

C. Assigning Security Aspects to Roles

The verdict so far is that responsibility bundles, permissions and permission constraints are connected. While responsibility bundles, permissions and permission constraints together describe the *security aspect*, roles and users together define the *organizational aspect*. We thus define a security policy to be the combination of *security aspect* and *organizational aspect*.

In order to simplify our argumentation we introduce the notion of *security bundles* $\mathcal{S} = \{s_1, \ldots, s_n\}$, where $s = \{b, p, x\}$ with $x \subseteq \mathcal{PC}$, is a single combination of responsibility bundles, permissions and permission constraints. As depicted in Fig. 3, a security policy is the connection between responsibility bundles, permissions, permission constraints, roles and eventually users.

While the relation between role and user is clear [13] (a role has several users), the relation between roles and the security aspect is more complicated:

- Every role can be related to $0 \ldots *$ security bundles.
- Every security bundle can be assigned to $1 \ldots *$ roles.
- Different security bundles can combine the same responsibility bundle with different permissions / permission constraints.
- Security bundles can override / complement each other.

In an organization hierarchy, it would be very tedious to assign all single responsibilities, permissions, and constraints to roles over and over again. A solution would be, to implement inheritance for the hierarchy, however this would introduce all the limitations, problems and solutions connected to inheritance (e.g. "diamond problem" as a role or user can inherit from multiple parent roles in order to implement flexible organizational structures), and thus introduce significant management and runtime complexity.

So in order to avoid inheritance and the overhead of single assignments we introduced the concept of responsibility and security bundles. Bundles follow the idea of mixins in object oriented languages: they are a means of collecting constraints and aspects and foster reuse. For example a responsibility bundle can be used to allow the employees of group to execute certain activities of process instance. The same responsibility bundle with a different permission (and optional permission constraints) can be used to allow management the monitoring of the execution of said activities.

When multiple security bundles with the same permission are assigned to a role, they can complement each other. They can be evaluated in the assigned order, with the first matching set of responsibilities and responsibility constraints denoting the relevant security bundles.

VI. Security Policy Mapping

While Sect. V describes a data model to allow for a comprehensive representation of arbitrary process related

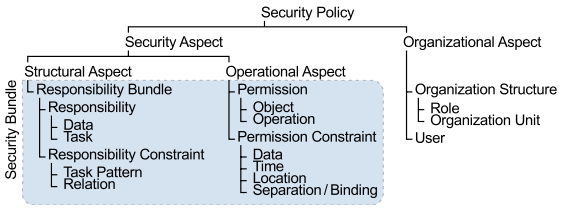


Figure 3. Security Policy - Overview and Definition

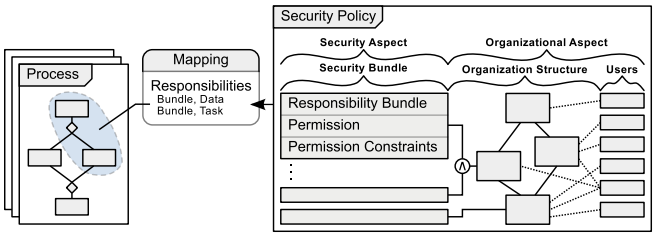


Figure 4. Security Policy Mapping

security policies, this section is dedicated to the mapping of security policies to actual processes and process instances. This includes:

A. Assigning responsibilities to actual processes models.
B. Checking for structural process model security by utilizing responsibility constraints.
C. Selecting roles based on the responsibility mapping and enforcing security policies based on the responsibility mapping (for running instances).

A. Assigning Responsibilities

Mapping responsibilities to process activities is intended to be carried out by a person (the policy guardian). As shown in Fig. 5, *Mapping* is carried out after *Acquisition* (Sect. IV), *Security Policy Design* (cf. Sect. V) and *Process Design*. In this section, we focus on point (1) of Mapping, depicted in Fig. 5.

As stated before this step is essential for providing the separation between a process (control flow, data flow) view and a security view, that focuses on users, roles and their responsibilities. Responsibilities consist of tasks and data objects, that are potentially shared and used in many processes, yet always fall under the same security restrictions.

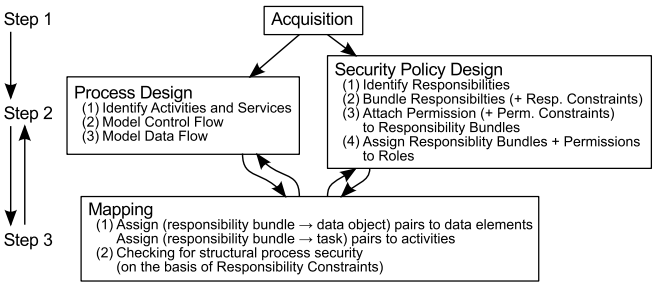


Figure 5. Responsibility Mapping

One advantage of the mapping is, that the policy guardian does not have to know about responsibility constraints or permission constraints. The policy guardian starts out with a list of responsibilities, that can be extracted from the data model described in Sect. V:

Listing 1. Extract Mapping Information

```
1 tasks = Array.new
2 data = Array.new
3 B.each |b|
4   b[R].each |r|
5     tasks << MappingPair.new(b.id,r.id) if r^data
6     data << MappingPair.new(b.id,r.id) if r^task
7   end
8 end
```

As can be seen in lines 5 and 6, two lists are prepared which hold entries that each consist of the bundle id and a responsibility id. Because responsibilities may occur in multiple bundles, the bundle id adds a category. We further refer to these pairs as $m^{data}_{x,y} = (b_x, r^{data}_y)$ and $m^{task}_{x,y} = (b_x, r^{task}_y)$.

Based on these two lists, the policy guardian has to iterate all processes in order to assign each m^{data} to data elements and m^{task} to activities. Please note that responsibility bundles are intended to include catchall responsibilities (see for example r^{data}_6 in Sect. V) in order to categorize additional data objects and tasks that are not important for security considers but are present in tasks.

It is also important to note that data objects and task of the travel request example (*b*travel request) may not only occur in a process travel request but also in multiple other accounting and reporting processes.

B. Checking for Structural Process Security

As stated in point (2) of *Mapping* in Fig. 5, after mapping the responsibilities, it is possible to check if the process structure is consistent with the responsibility constraints. As described in Sect. V, responsibility constraints neither describe the input or output of tasks, nor do they describe the exact task order. They just loosely describe a set of tasks (which may occur mixed with other tasks in an actual process), and some connected data objects (how they are connected is not specified).

These responsibility constraints can be utilized for running an automated check on processes. This check has to occur: (1) whenever the mapping for a processes is finished, and (2) whenever a mapped process is changed. The result of such a check consists of three different classes of errors:

- Tasks occur in an unspecified order (no rc^{tp} for this particular order exists).
- Known data objects are used or written in combination with unknown tasks (as specified in a rc^{r}).
- Unknown data objects are used in combination with known tasks (as specified in a rc^{r}).

As stated before, for each bundle one or many rc^{tp} or rc^{r} constraints may match. If multiple rc^{tp} or rc^{r} constraints exist, they are independent.

An error may have one of the following reasons/solutions: **(1)** The policy guardian has made a *mapping error*. He/she can correct the error. **(2)** Because of process change, *additional mapping* is necessary. The policy guardian can correct the error. **(3)** The process is not consistent with the responsibilities and responsibility constraints for reasons *unknown* to the policy guardian: (a) trigger a process designer for possible correction, if the process designer confirms the process structure, or (b) trigger a security policy designer for possible adaption of the security policy. (b) may lead to a confirmation of the security policy in which case the process designer is overruled and has to redesign the process anyway.

The methodology described in this section depends on humans. The automatic checking is convenient, but solving the resulting errors is to be coordinated by the policy guardian.

C. Selecting Roles

As explained in Sect. V, security policies not only consist of (1) responsibilities that describe structural aspects, but also (2) permissions that describe operational aspects. In this section, we discuss how to derive a set of roles and eventually users that are associated with a certain activity or set of activities in a process. This selection is based on the mapping of responsibilities to data objects and tasks as described above, and the data model introduced in Sect. V.

Listing 2. Role Selection

```
1 # input variable process:    pr
2 # input variable tasks:      ta
3 # input variable object:     ob
4 # input variable operation: op
5 rb = Array.new # list of responsibility bundles
6 B.each do |b|
7   temp = Array.new
8   b[R].each do |r|
9     if r^task
10      temp << r
11    end
12  end
13  if ta ⊆ temp
14    b[RC].each do |rc|
15      if rc^tp and m = rc.matches(pr) and ta ⊆ m
16        rb << b
17        break
18      end
19    end
20  end
21 end
22 roles = Array.new # list of roles
23 S.each do |s|
24   success = true
25   if s[b] ∈ rb and s[p].object==ob and s[p].operation==op
26     s[x].each do |pc|
27       unless pc
28         success = false # pc not successful evaluated
29       end
30     end
31     if success
32       roles << s.connected_roles
33     end
34   end
35 end
36 roles = roles.uniq # remove duplicate roles
```

The algorithm in Listing 2 as a perquisite assumes that four input variables are set in the first lines: (1) *process* holds the process id in which certain (2) *tasks* occur, for which a certain security (3+4) *object and operation* is requested. For example when running a process instance, it is necessary to derive a list of roles that is allowed to execute a certain activity. In this case object is "control flow" and operation is "execute". Many other combinations for fine-grained administration, change, and monitoring of processes are imaginable.

For each bundle (line 6) every responsibility inside the bundle is iterated and all tasks are concatenated to a list `temp` (8 to 12). Lines 13 to 20 describe that if the set of input tasks is a part of the responsibility task in the bundle, then check all responsibility constraints (15): if the process containing the tasks, matches a certain task pattern and the tasks are part of the match. If this condition holds true this responsibility bundle matches and is appended to the list `rb`.

In the next step (line 23) all security bundles are iterated, in order to check (25) if they hold a responsibility bundle identified in the last step, and if the object and operation for this security bundle matches the request. Lines 26 to 30 ensure that all permission constraints for the given permission are fulfilled, which eventually leads to appending the all roles connected to this security bundle to a list of `roles` (see line 32).

The resulting list of roles can be used in different process related contexts. For example *worklists* need a set of roles in order to show tasks for certain users. But the most important application is an independent *security monitor* in order to check if a certain user is allowed to execute a certain task (*Enforcing of Security Policies*).

VII. Related Work

In this section, we discuss and evaluate a selection of policy approaches and our method along the design requirements as set out in Sect. III. The results of the evaluation are shown in Tab. I.

Table I
Evaluation of existing Approaches

Policy	R1	R2	R3	R4
NIST RBAC [13], W-RBAC [14]	+			
ARBAC [15]	+	+	+	
Bertino et al [4], Casati et al [16]	+			
Riberio et al [17]	+	+	+	
Neumann et al [2]	+	∼	∼	
Responsibilty-driven	+	+	+	+

The Role-Based Access Control (RBAC) model (e.g., NIST RBAC [13]) expresses security policies based on the role-permission assignments. Further models such as Administrative RBAC models (e.g., ARBAC [15]) or PAIS related RBAC models (e.g., Workflow RBAC [14]) have been developed. Whereas these models depend on abilities

of authorized users to perform tasks (e.g., set in job description), our approach uses responsibilities such as data objects or tasks related to permissions and roles to develop security policies. Even though RBAC models enable independence (R1), only administrative models enable maintenance features (R2, R3) in PAIS.

The specification and enforcement of constraints in PAIS are proposed in [2], [4], [16]. They support independence (R1) but most of them ignore maintainability (R2), extendibility (R3), and scalability (R4) in PAIS. In our paper, static and dynamic authorization *and* assignment constraints are enforced. In [17], workflow processes are verified against organization security policies by transforming each in a common constraint language. This way, scalability (R4) can become an issue when transforming and processing a large amount of data. In our approach, inconsistencies only have to be checked when policies are created or at change time.

The related work discussion showed that our approach is a new method for designing and developing security policies in PAIS. To our best knowledge, no other approach has met all necessary design requirements in PAIS.

VIII. Conclusion

In this paper we presented design and development techniques for security policies in PAIS. The main motivation behind is the separation of business processes and security policy aspects, since in current PAIS often a mix of both aspects exists. This, however, hampers consistency checks and enforcement of the security policies on the one side, and maintenance and evolution of processes and associated policies on the other side. The latter problem mainly arises due to the potential side effects of changes that cannot be controlled within such a mixed representation. To achieve independence of security policies from business processes they are imposed on, we enriched existing RBAC models with structural aspects (responsibilities) as used in PAIS to further define security policies. Based on these extensions, all different kinds of security policies can be expressed (e.g., access rules, authorization constraints, or context-aware security policies) and they can be easily connected to the business processes based on a simple mapping. This approach can be deployed on *every* PAIS regardless of the process modeling and system used. In future work we aim at extending our considerations to cross-organizational process settings. Further, we will integrate all concepts within a proof-of-concept prototype.

Acknowledgements

The research was partially funded by COMET K1, FFG - Austrian Research Promotion Agency.

References

[1] N. Russell, W. M. van der Aalst, A. H. ter Hofstede, and D. Edmond, "Workflow resource patterns: Identification, representation and tool support," in *Proc. of CAISE*. Springer, 2005, pp. 216–232.

[2] G. Neumann and M. Strembeck, "An approach to engineer and enforce context constraints in an RBAC environment," in *Proc. of ACM SACMAT*. ACM, 2003, pp. 65–79.

[3] P. C. K. Hung and K. Karlapalem, "A secure workflow model," in *Proc. of AISC on ACSW frontiers 2003 - Volume 21*. Australian Computer Society, Inc., 2003, pp. 33–41.

[4] E. Bertino, E. Ferrari, and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 1, pp. 65–104, 1999.

[5] R. J. Anderson, "Security in clinical systems," Tech. Rep. 1.1, 1996.

[6] S. Rinderle-Ma and M. Leitner, "On evolving organizational models without loosing control on authorization constraints in web service orchestrations," in *CEC*, 2010.

[7] J. E. Cook and A. L. Wolf, "Event-based detection of concurrency," in *Proc. of ACM SIGSOFT FSE*. ACM, 1998, p. 35–45.

[8] W. van der Aalst, H. Reijers, A. Weijters, B. van Dongen, A. A. de Medeiros, M. Song, and H. Verbeek, "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5, pp. 713–732, Jul. 2007.

[9] E. J. Coyne, "Role engineering," in *Proceedings of the first ACM Workshop on Role-based access control*. ACM, 1996.

[10] G. Neumann and M. Strembeck, "A scenario-driven role engineering process for functional RBAC roles," in *Proc. of ACM SACMAT*. ACM, 2002, p. 33–42.

[11] M. Kuhlmann, D. Shohat, and G. Schimpf, "Role mining - revealing business roles for security administration using data mining technology," in *Proceedings of the eighth ACM SACMAT*. ACM, 2003, p. 179–186.

[12] M. Pesic and W. van der Aalst, "A declarative approach for flexible business processes management," in *Business Process Management Workshops*. Springer, 2006, vol. 4103, pp. 169–180.

[13] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST model for role-based access control: towards a unified standard," in *Proceedings of the fifth ACM workshop on Role-based access control*, 2000, pp. 47–63.

[14] J. Wainer, P. Barthelmess, and A. Kumar, "W-RBAC - a workflow security model incorporating controlled overriding of constraints," *International Journal of Cooperative Information Systems*, vol. 12, no. 4, pp. 455–485, 2003.

[15] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 1, pp. 105–135, 1999.

[16] F. Casati, S. Castano, and M. Fugini, "Managing workflow authorization constraints through active database technology," *Inf. Syst. Frontiers*, vol. 3, no. 3, pp. 319–338, 2001.

[17] C. Ribeiro and P. Guedes, "Verifying workflow processes against organization security policies," in *Proc. of WETICE*. IEEE Computer Society, 1999, pp. 190–191.