SPRINT*- Responsibilities: Design and Development of Security Policies in Process-aware Information Systems†

Maria Leitner[1]‡ Juergen Mangler[2] and Stefanie Rinderle-Ma[1]

[1] *University of Vienna*
Faculty of Computer Science
Research Group Workflow Systems and Technology
Vienna, Austria
{maria.leitner,stefanie.rinderle-ma}@univie.ac.at

[2] *SBA Research*
Vienna, Austria
jmangler@sba-research.org

Abstract

Process-Aware Information Systems (PAIS) enable the definition, execution, and management of business processes. Typically, processes are specified by control flow, data flow, and users or services, authorized to execute process tasks. During process execution, it is often necessary to access sensitive data such as patient or customer information. To secure this confidential data, the use of security policies becomes an essential factor for the application of PAIS in practice. In general, PAIS security policies are specified based on access rules and authorization constraints. On top of these rules, context policies referring to data, location, or time might pose restrictions. Over the years, several approaches for modeling and enforcing security policies in PAIS have appeared. Many of them restrict security policy specification to access rules and authorization constraints, but neglect additional properties such as context information. As a further limitation, security policies are often defined in a heterogeneous way: whereas access rules are mostly defined at process task level leading to a merge of process logic and security aspects, additional policies such as authorization constraints are defined separately from the process logic. Consequently, security policies are not stored and managed centrally, but are rather distributed over different PAIS components, for example, the process model repository or the organizational model manager. In this paper, we introduce the formal concepts behind our SPRINT approach that aims at the consequent separation of security policies and process logic. Specifically, the SPRINT security policy data model and design methodology based on the concepts of responsibilities, permissions, and constraints will be provided. The concepts are evaluated based on a comparison with existing PAIS and a demonstration of the SPRINT prototype. The goal is to unify diverse security policies in different PAIS subsystems, to make security policies independent of these subsystems in order to restrain complexity from process modeling and evolution, and to allow for comprehensive security policy development and maintenance.

Keywords: Security policy design, workflow security, responsibilities, access control, security constraints, process-aware information systems

1 Introduction

Process-Aware Information Systems (PAIS) enable the automated execution of business processes carried out by various actors and the management of private and public data, for example, electronic patient records or bank accounts. With the use of sensitive data in these systems, security becomes an important factor in practice. Currently, commercial systems, such as Staffware, Websphere MQ Workflow

*Security in PRocess-aware INformation SysTems: `http://cs.univie.ac.at/project/SPRINT`

†This paper is an extended version of the work originally presented at the 6th International Conference on Availability, Reliability and Security (ARES'11), Vienna, Austria, August 2011 [1].

‡Corresponding author: University of Vienna, Faculty of Computer Science, Workflow Systems and Technology, Rathausstrasse 19/9, A-1010 Vienna, Austria, Tel: +4314277-39513, Email: maria.leitner@univie.ac.at

or AristaFlow, and research prototypes (e.g., YAWL) offer role-based access control mechanisms that link process activities to organizational structures via so called access rules [2]. In addition, authorization constraints such as separation of duties are defined on top of the access control mechanisms. At last, these rules and constraints are supplemented with contextual information e.g., location or time [3]. Imagine for example, a security policy in a hospital specifying that patient records are only accessed by physicians on duty or by mobile devices in a restricted area.

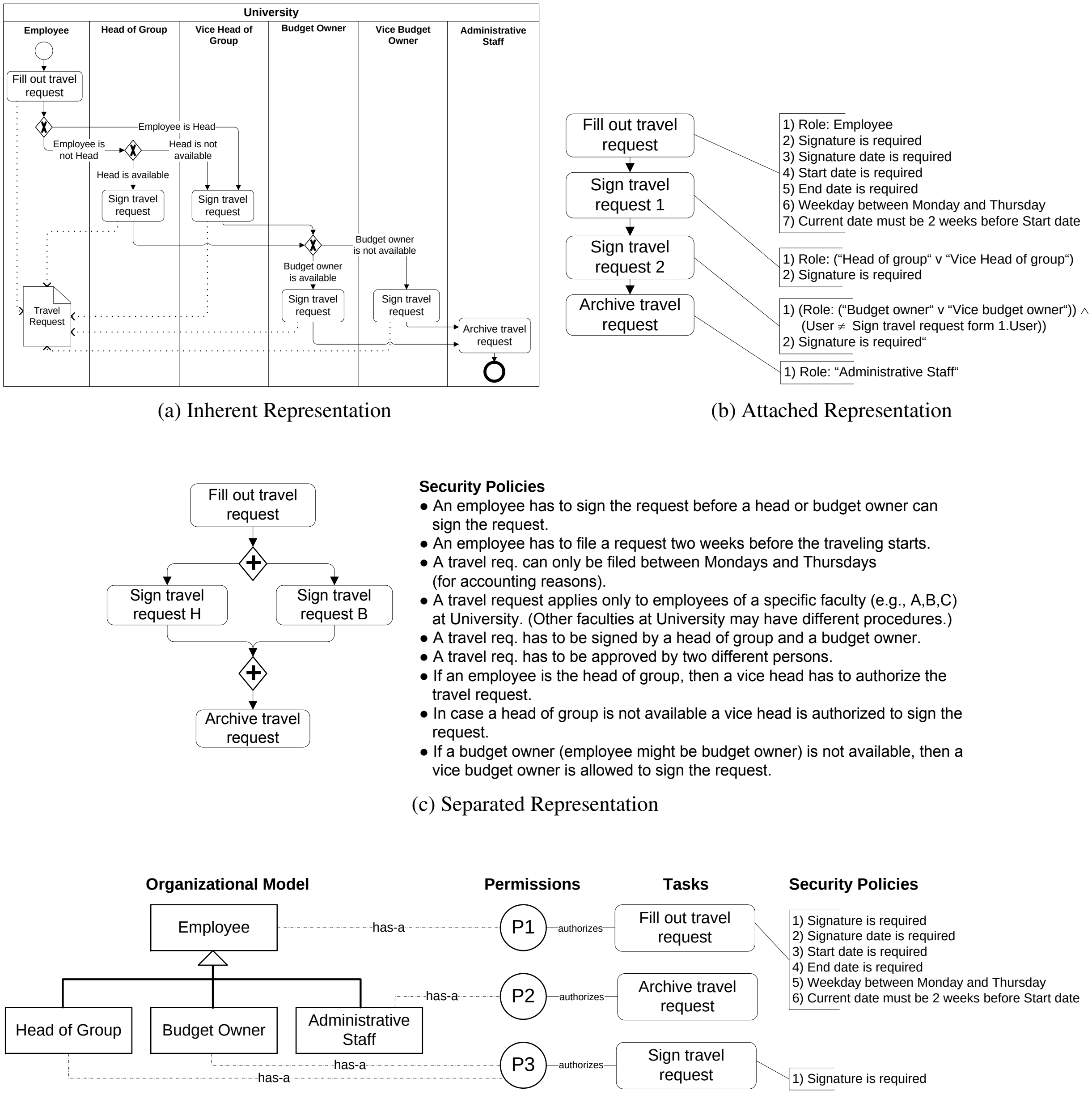
In the following, we refer to the access rules and authorization constraints imposed over a process, potentially enriched by context, by *process-relevant security policies*. As defined in literature [4], business processes are secure if all process-relevant security policies are not violated or, respectively, fulfilled. Different approaches for modeling and enforcing process-relevant security policies exist that assume a set of explicitly defined security policies [5]. These security policies are typically assumed to be stored in a security policy repository and verified over the processes during design or run time. Policies that can be verified at process design time are, for example, static separation of duty constraints. On the other hand, dynamic separation of duty constraints can only be enforced at run time or verified at run time or ex post. One approach for the ex post verification of security policies is based on LTL checking of process logs [6]. However, the assumption that all security policies are explicitly defined and stored within a repository does not hold true for PAIS in practice. In turn, 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 [7]. Further, they might even be integrated within control flow structure such as process tasks or activities of a process model. Another 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, maintenance, and evolution of the PAIS.

This paper is an extension of the paper presented at the ARES 2011 conference in Vienna [1]: In the SPRINT approach, 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 changes of either the process or policies occur, the side effects can be easily handled within the mapping. This extended version of [1] provides an extensive evaluation of existing process notations, Business Process Modeling as well as Process Management Tools. Further on, the prototypical implementation of the SPRINT approach is presented. To sum up, the SPRINT approach establishes a new model for developing security policies for PAIS.

Section 2 gives an overview of security policies in PAIS. In Section 3, security policy design requirements are presented. Section 4 describes security policy acquisition. In the following, design (cf. Section 5) and mapping of security policies (see Section 6) in SPRINT are presented. Section 7 provides an evaluation of existing process notations and tools. The prototypical implementation is presented in Section 8. We discuss related work in Section 9 and conclude in Section 10.

2 Security Policies in PAIS

Security policies are a set of statements of a systems protection strategy, for example, access control rules [8]. 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,



(a) Inherent Representation

(b) Attached Representation

Security Policies

- An employee has to sign the request before a head or budget owner can sign the request.
- An employee has to file a request two weeks before the traveling starts.
- A travel req. can only be filed between Mondays and Thursdays (for accounting reasons).
- A travel request applies only to employees of a specific faculty (e.g., A,B,C) at University. (Other faculties at University may have different procedures.)
- A travel req. has to be signed by a head of group and a budget owner.
- A travel req. has to be approved by two different persons.
- If an employee is the head of group, then a vice head has to authorize the travel request.
- In case a head of group is not available a vice head is authorized to sign the request.
- If a budget owner (employee might be budget owner) is not available, then a vice budget owner is allowed to sign the request.

(c) Separated Representation

(d) Task-based Representation

Figure 1: Travel Request: Process Modeling and Security Policies

information flow, data integrity, and availability. The four 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 four different ways of representing security policies in PAIS:

(a) As part of the process logic as shown in Fig. 1a.

(b) Attached to tasks as shown in Fig. 1b (is the most common approach [9]).

(c) As separate group of annotations, loosely connected to a process as shown in Fig. 1c. This implies that the security model includes knowledge of the process structure (sequence of tasks) that is independent of the process model.

(d) As task-based authorizations (cf. Fig. 1d) integrated in access control models such as in [10].

From a technical point of view, most approaches dealing with authorization constraints such as [5], define security policies as explicit policies that are stored in a repository. This approach is valid for representations b, c, and d, while the granularity of policies is different. For example in representations b and d, 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).

3 Security Policy Design Requirements

In this section, we will present the systems requirements to achieve the most important goal of our approach: operating security policies independently from specific process models or process modeling notations. Therefore, we define the design requirements with natural language. All requirements are obtained from real world case studies and projects. In this paper, we want to focus on the core system requirements for enabling independence. Hence, we chose to only include requirements most relevant for the SPRINT approach. This list of requirements does not claim to be exhaustive and can be enhanced with other requirements (e.g., usability). The requirements can be further formalized by analyzing goal semantics (e.g., [11]) or by using goal models such as [12]. Later in the paper, we will use evaluation-based research (cf. [13]) to verify the requirements (see Section 8).

In order to achieve operating security policies independently from specific process models or process modeling notations, we define the following requirements:

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

Requirement 2 (Maintainability of Security Policies)**.** *Security policies should be easy to maintain (e.g., add, change, and 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 to redesign processes models.*

Requirement 3 (Extendability of Security Policies)**.** *Due to constantly changing business environments, changing process models and security policies is very important. We think that this extendability should be decoupled. Changing processes models, such as adding or deleting activities, 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 of PAIS Components)**.** *Scalability is about the ability to handle growth. In PAIS, scalability means managing an increase of components such as activities or security policies. Certainly, 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, and conflict resolution).*

4 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., [14]) and role engineering [15, 16] and role mining [17] 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, and 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. It centers 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 information about the sequence (or structure) of activities, temporal relations between the activities, and correlation of data elements to activities, is present in the processes *and* 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.

5 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 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 and monitoring of individual tasks in the process, and 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, nonetheless, it is intended to be completely independent of processes. The structural aspect

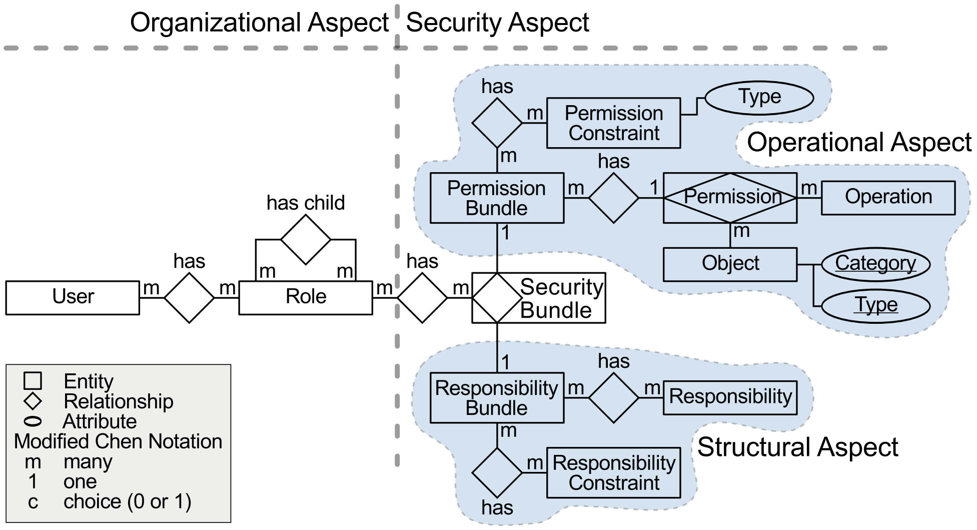


Figure 2: Security Policy Data Model

just deals with a set of names, further called responsibilities, classified either as data object or task. This set can be directly derived from information collected through a process acquisition (cf. Section 4). The structural aspect serves as a basis for the operational aspect and is mapped to the actual data elements and activities of available processes (as described in Section 6).

Security Policy Data Model

In this section, we present a Role-Based Access Control (RBAC) model (e.g., [18]) that integrates structural and operational security aspects (see Figure 2). RBAC is a de facto standard for access control in information systems. Other access control models, such as Attribute-Based Access Control (ABAC), can be also related to our model. For example, extracted information of an ABAC model (e.g., [19]) can be imported into our model. The Security Policy Data Model also provides flexibility by supporting processes throughout the whole life cycle (c.f. [20]). In the following, we will describe the Security Policy Data Model by defining (1) structural aspects using the concept of responsibilities, (2) specifying operational aspects by restricting responsibilities in a certain way, and (3) combining security aspects with organizational aspects to establish security policies.

Responsibilities

The first aspect in the Security Policy Data Model is the structural security aspect. 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 $\mathscr{R}$ that serves 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 constraints related to data and order of tasks, and security constraints (operations allowed regarding data and structure on certain circumstances). For more details please see Section 6.

Assignment: A group of responsibilities including a set of constraints may be assigned to several roles. The creation of bundles fosters structure and reusability. 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 exists a set of responsibility bundles $\mathscr{R} = \{b_1, \ldots, b_n\}$. A responsibility bundle $\mathscr{R}$ contains a set of responsibilities $r = \{r_1, \ldots, r_n\}$ and a set of constraints $rc = \{rc_1, \ldots, rc_n\}$ referring to a subset of r.

In order to exemplify the concepts involved in a responsibility bundle, we introduce a simple travel request example $b_{\text{travel request}}$ (cf. Section 2, Fig. 1c). In this case, a responsibility bundle contains $b_{\text{travel request}} = \{\{r_1, \ldots, r_9\}, rc_1^{tp}, rc_2^{r}\}$ a set of responsibilities r and responsibility constraints rc. The following responsibilities are included:

$$\begin{aligned}
&r_0^{data} : \text{start date}\\
&r_1^{data} : \text{end date}\\
&r_2^{data} : \text{signature employee}\\
&r_3^{data} : \text{signature employee date}\\
&r_4^{data} : \text{signature approval a}\\
&r_5^{data} : \text{signature approval b}\\
&r_6^{data} : \text{additional unspecified data}\\
&r_7^{task} : \text{filling form and signing it}\\
&r_8^{task} : \text{approval A}\\
&r_9^{task} : \text{approval B}
\end{aligned} \qquad (b_{\text{travel request}}, \text{Responsibilities})$$

Example ($b_{\text{travel request}}$, Responsibilities) displays a set of responsibilities $r = \{r_1, \ldots, r_9\}$ 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** rc: *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}. In this paper, we describe these patterns as Linear Temporal Logic (LTL) expressions [21], although arbitrary languages for describing structural dependencies may be used. For the $b_{\text{travel request}}$ example the pattern is as follows:

$$rc_1^{tp} : \Box(r_7^{task} \rightarrow ((\Diamond r_8^{task} \rightarrow \Diamond r_9^{task}) \vee \Diamond r_9^{task})) \qquad (b_{\text{travel request}}, rc_1^{tp})$$

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.

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:

$$rc_2^r : r_0^{data} \wedge r_1^{data} \wedge r_2^{data} \wedge r_3^{data} \wedge r_4^{data} \wedge r_5^{data} \wedge r_6^{data} \wedge rc_1^{tp} \qquad (b_{\text{travel request}}, rc_2^r)$$

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 Section 6. 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.

Permissions

In contrast to the structural aspect, the operational aspect deals with permissions and constraints on these permissions. Single permissions in conjunction with a set of permission constraints form permission bundles $\mathscr{P}$ with the following purpose: Permissions constraints refer to responsibilities defined in a responsibility bundle $\mathscr{R}$, and restrict them in a certain process related security context (i.e., a specific permission).

We define a Permission Bundle $\mathscr{P} = \{p, pc\}$ to consist of a single permission p and a set of permissions constraints $pc = \{pc_1, \ldots, pc_n\}$. **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. One permission is further constrained by set of **Permission Constraints**. As the permission describes “the situation”, not all of the following constraints make sense for each possible permission. For example, 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 connected 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.

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

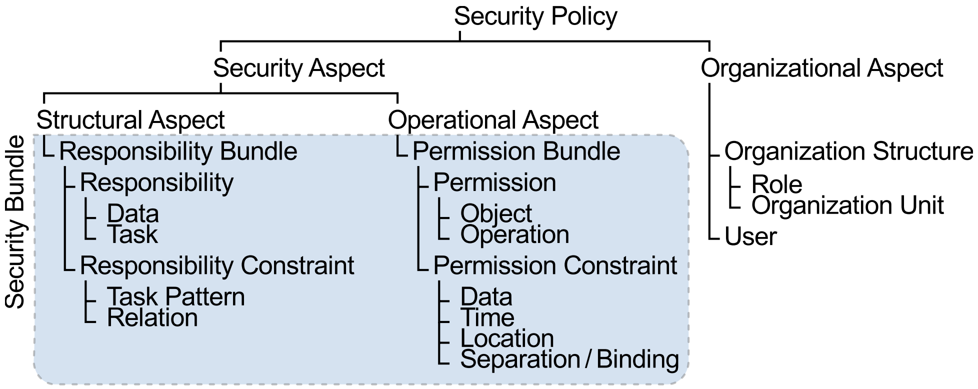


Figure 3: Security Policy - Overview and Definition

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 to *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:

$$\begin{aligned} pc_3^p &: \text{control flow}, r_8, \text{execute} \\ pc_{3,0}^{sb} &: (r_7 \neq r_8) \wedge (r_8 \neq r_9) \end{aligned} \qquad (b_{\text{travel request}}, \text{permission } 2)$$

pc_3^p describes the execution of r_8 which is intended for the role of group leaders. $pc_{3,0}^{sb}$ denotes that a user who 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 who filed the travel request signs the second approval (e.g., head of group).

Assigning Security Aspects to Roles

The verdict so far is that responsibility bundles and permission bundles together describe the *security aspect*, while roles and users together define the *organizational aspect*. Thus, we 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* $\mathscr{S} = \{s_1, \ldots, s_n\}$, where $s = \{\mathscr{R}, \mathscr{P}\}$ is a single combination of responsibility bundles and permission bundles. As depicted in Fig. 3, a security policy is the connection between responsibility bundles, permission bundles, roles and eventually users.

While the relation between roles and users is clear [18] (a role has several users), the relation between roles and security aspects is more complicated:

- Every role can be related to 0...* security bundles.
- Every security bundle can be assigned to 1...* roles.

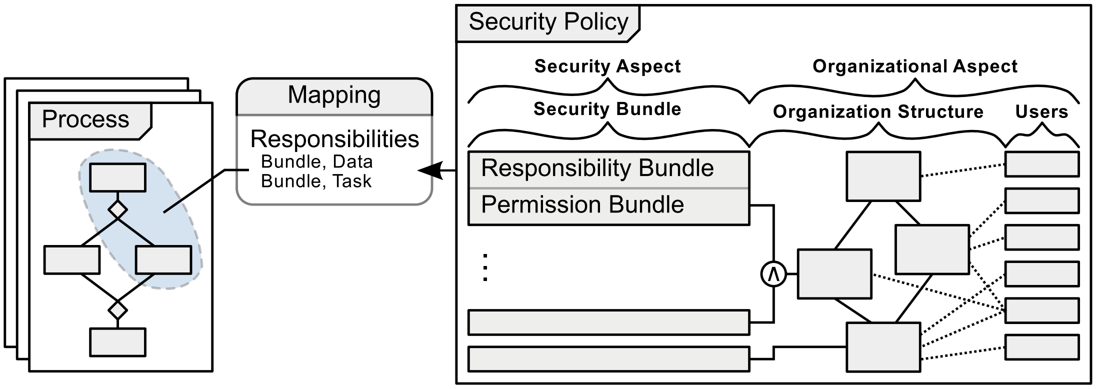


Figure 4: Security Policy Mapping

- 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 introduce significant management and runtime complexity.

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 a group to execute certain activities of a 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.

6 Security Policy Mapping

While Section 5 describes a data model to allow for a comprehensive representation of arbitrary process related security policies, this section is dedicated to the mapping of security policies to actual processes and process instances (see Fig. 4). 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).

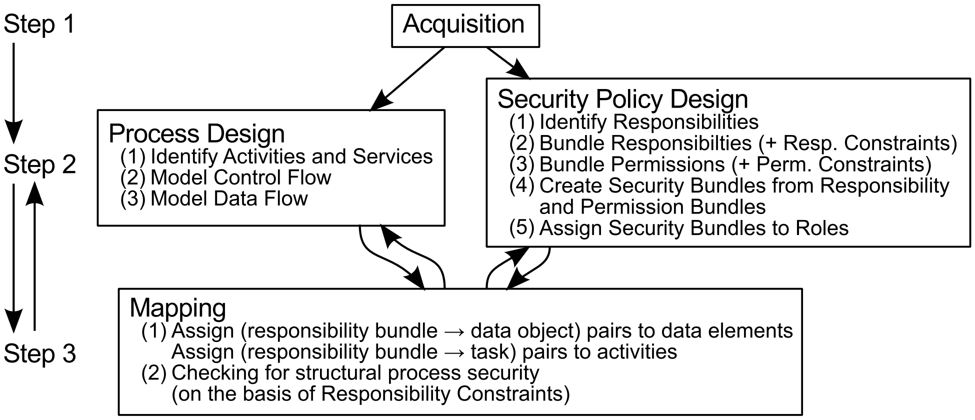

Step 1
Step 2
Step 3
Acquisition
Process Design
(1) Identify Activities and Services
(2) Model Control Flow
(3) Model Data Flow
Security Policy Design
(1) Identify Responsibilities
(2) Bundle Responsibilties (+ Resp. Constraints)
(3) Bundle Permissions (+ Perm. Constraints)
(4) Create Security Bundles from Responsibility and Permission Bundles
(5) Assign Security Bundles to Roles
Mapping
(1) Assign (responsibility bundle → data object) pairs to data elements
Assign (responsibility bundle → task) pairs to activities
(2) Checking for structural process security
(on the basis of Responsibility Constraints)

Figure 5: Responsibility Mapping

6.1 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* (Section 4), *Security Policy Design* (cf. Section 5) 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. 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 Section 5). The extraction, as displayed in Listing 1, is fairly straight forward: it generates a list of task and data responsibilities per bundle:

Listing 1: Extract Mapping Information

```
1 tasks = Array.new
2 data = Array.new
3 $\mathscr{R}$.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 in Listing 1, two lists are prepared which hold entries that are identified by the combination of bundle id and responsibility id. We further refer to these pairs as $m^{data}_{x,y} = (\mathscr{R}_x, r^{data}_y)$ and $m^{task}_{x,y} = (\mathscr{R}_x, r^{task}_y)$. 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 Section 5) 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 ($b_{\text{travel request}}$) may not only occur in a process travel request but also in multiple other accounting and reporting processes.

6.2 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 Section 5, 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* and can correct the error. **(2)** In case 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: (3a) trigger a process designer for possible correction if the process designer confirms the process structure, or (3b) trigger a security policy designer for possible adaption of the security policy. (3b) 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.

6.3 Selecting Roles

As explained in Section 5, 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 Section 5.

Listing 2: Role Selection

```
# input variable process:   pr
# input variable tasks:     ta
# input variable object:    ob
# input variable operation: op
rb = Array.new # list of responsibility bundles
$\mathscr{R}$.each do |b|
  temp = Array.new
  b[r].each do |r|
    if $r^{task}$
      temp << r
    end
  end
  if ta ⊆ temp
    b[rc].each do |rc|
      if $rc^{tp}$ and m = rc.matches(pr) and ta ⊆ m
        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. However, the most important application is an independent *security monitor* in order to check if certain users are allowed to execute certain tasks (i.e., enforcement of Security Policies).

7 Evaluation

The goal of this section is to provide a comparison of existing business process notations, business process modeling, and process management tools with respect to the question whether and how they support the definition, enforcement, verification, and management of security policies. Specifically for the definition of security policies, an important comparison criterion is the expressiveness of the security policies. As discussed in Section 2, basically, there are two different ways of specifying security policies: (a) a set of security policies, for example, managed within a repository, or (b) implementation of security policies divided into access rules defined at task level and additional (authorization) constraints on top of the access rules such as separation of duties. Variant (a) is mostly proposed in literature, whereas (b) is often used in research and practice. Hence, in the comparison of notations and tools, we employ variant (a) in SPRINT by considering the specification of access rules and additional (authorization) constraints.

On top of access rules and (authorization) constraints, it is necessary to define additional constraints, for example, imposing time, data, or location constraints. In the comparison, we evaluate the basic

Table 1: *Evaluation of Process Notations and Modeling Tools*

	BPMN	EPC	Workflow Nets	ARIS
Policy Modeling				
Definition Access Rules	Swimlanes	Organigram	□[1]	Organigram
(Authorization) Constraints	Annotation	Annotation	□[1]	Annotation
Basic Role Assignment	☒	☒	□	☒
Advanced Role Assignment	□	□	□	Function Relationship Diagram
Policy Constraint Properties				
Basic Data	☒	☒	□	☒
Basic Time	☒	Annotation	☒	☒
Basic Location	Annotation	Annotation	Annotation	Annotation
Use in Constraints	□	□	□	□

[1] Only in extended Workflow Nets.

possibilities for imposing such constraints, i.e., whether and how do notations / tools basically support data or time aspects. Further, we discuss whether the tools and notations directly support the definition of additional constraint properties such as time, data, and location on top of security policies. The differentiation is important since even if the definition of constraint properties is not directly supported yet, the basic support might imply the possibility for respective extensions.

These criteria are evaluated along a set of process notations and modeling tools in Table 1. The criteria catalog will be reused for process management systems and extended by criteria regarding the process execution aspect. Overall, the selected notations and tools provide a broad scope of functionality and a mixture of academic prototypes, open source, and commercial systems.

7.1 Process Notations and Modeling Tools

Table 1 comprises the evaluation of three process notations, i.e., BPMN (`http://www.omg.org/spec/BPMN/2.0/`), EPCs (e.g., [22]), and Workflow Nets (e.g., [23]), as well as process modeling tool ARIS (`www.aris.com`). A first important question is how the notation supports the definition of security policies from a language point of view. Inherently, different methods are used, such as swim lanes (BPMN) or linkage of functions and organizational units (EPCs). Using swim lanes, access rules are modeled based on lanes that can be further bundled into pools describing, for example, participants in a collaboration. BPMN does not explicitly specify the usage of lanes, but the concept is often used to express internal roles. The fundamental difference is that based on swim lanes organizational aspects are "merged" into the process logic, whereas the use of separated organizational elements that are assigned to control flow elements e.g., role-task assignments, basically enables the separation of both aspects. This idea of separation is realized in ARIS for Event-driven Process Chains (EPCs) where organizational elements and tasks can be assigned to each other in a separated diagram, called function-assignment diagram. Further on, swim lanes as used in BPMN 2.0 enable the assignment of maximum two roles to one task. Neither assignment of three or more roles to one task nor the specification of certain security policies (e.g., separation of duties) are supported by the standard (individual extensions exist e.g., [24]). Generally, BPMN supports modeling of data flow and timer events. The latter can be used for expressing time constraints that, for example, trigger an exceptional threat of control flow (e.g., send a reminder message, complete a task) after a certain time limit. (Basic) Workflow Nets support security policies on a rudimentary level. Organizational aspects are supported in extended Workflow Nets, i.e., by resource-constrained Workflow Nets [25]. Basic Workflow Nets also abstract from data aspects [23], however, support time triggers that specify a time period until the next (enabled) activity starts. Workflow Nets coated with "syntactic sugaring" and extended by data and resources are used, for example, in YAWL that is discussed in the next section. Overall, all process notations support data and time aspects that might be used for constraint property specification. None of these notations supports the direct definition of such constraint properties though, except for textual annotations. Such annotations can be exploited for later implementation of the processes within process management tools.

7.2 Process Management Tools

In addition to the notations and process modeling tools presented in Table 1, process management tools are evaluated regarding the modeling, specification, verification, and enforcement of security policies. Table 2 displays the results of the evaluation of the process management tools AristaFlow (e.g., [26]) together with the compliance plugin SeaFlows [27], Apache ODE (`http://ode.apache.org`), DECLARE (e.g., [28]), YAWL (e.g., [29]) and the SPRINT prototype.

First of all, the notation used by the process management tool is important because it indirectly affects the specification and range of security policies. All process management tools use different process notations and formalisms, i.e., Well-Structured Marketing-Nets (WSM-Nets) in AristaFlow, Compliance Rule Graphs (CRG) and First Order Logic (FOL) in SeaFlows [30], WS-BPEL in Apache ODE, Linear Temporal Logic (LTL) in DECLARE, extended Workflow Nets in YAWL, and Process Pattern Matching Expression (PPMEX) in SPRINT [7]. Contrary to declarative languages such as LTL and PPMEX that are based on defining process constraints, imperative languages such as WSM-Nets, WS-BPEL, and Workflow Nets require an additional "layer" for specifying constraints on top of the imperatively defined process logic (including data flow and organizational aspects). For the implementation of WSM-Nets in AristaFlow, for example, the SeaFlows plugin enables the specification of control and data-aware constraints on top of the process logic.

Aside these general considerations on the process notations, we evaluated whether the tools implement security policies as a set of policies in a separate repository (variant (a)) or at task level (variant (b)). Most applications support the specification of security policies attached to workflow tasks except for SPRINT that stores and manages them within a separate repository [7].

For example in AristaFlow, basic role assignments (e.g., assigning tasks with organizational units, roles or users) are directly specified at task-level based on an organizational model. Advanced assignments are also supported because it provides functionality to impose additional constraints (e.g., separation of duties) on the basic role assignments. Time constraints are enabled with escalations that define time limits until tasks have to be completed. In connection with the plugin SeaFlows, AristaFlow supports the enforcement of control and data flow constraints as well. Basically, SeaFlows offers the possibility to include advanced security policies in connection with data, time, and location for verification over the processes at design, run, and change time [31].

Apache ODE focuses on process control flow, specifically on the interactions between business partners, and enables the definition of some policies as well. For example, time-based alarms and time expiration periods are defined in the activities `pick` and `wait`. Other security policies (e.g., separation of duties) are not supported by default, but WS-BPEL extensions integrate further policies such as resource constraints (e.g., [32]).

As declarative language, DECLARE specifies control flow constraints such as relation (e.g., if task A is executed then task B has also to be executed), existence (e.g., perform task A at least once), or choice (e.g., choose one task out of three). It also supports the specification of mutual exclusion constraints for tasks. Using the LTL extension of process mining [6], security policies can be verified based on process logs ex-post, for example, whether or not two tasks have been always executed by the same person for all process instances (binding of duties).

In YAWL, several security policies are defined attached to workflow tasks. For example, task timers specify an expiry time or duration period until a task is completed (despite the current task status).

Table 2: *Evaluation of Workflow Execution Tools*

	AristaFlow / SeaFlows	Apache ODE	Declare	YAWL	SPRINT
Policy Modeling					
Notation	WSM-Net / CRG, FOL	WS-BPEL	LTL	Extended Workflow Nets	PPMEX
Implementation	Task / Repository	Task	Task	Task	Repository
Basic Role Assignment	☒ / ⊡	⊡	☒	☒	☒
Advanced Role Assignment	☒ / ⊡	⊡	□	☒	☒
Policy Constraint Properties					
Basic Data	☒ / ☒	☒	☒	☒	☒
Basic Time	☒ / ⊡	☒	☒	☒	☒
Basic Location	⊡ / ☒	⊡	⊡	⊡	☒
Policy Enforcement and Monitoring (Examples)					
Data	☒ / ☒	☒	☒	☒	☒
Time	☒ / ⊡	☒	⊡	☒	☒
Location	⊡ / ☒	⊡	⊡	⊡	☒
Separation/Binding of Duty	☒ / ⊡	⊡	☒	☒	☒
Policy Verification					
Conflicting Policies	☒ / ⊡	□	☒	☒	⊡
Empty Valid Actor Set	⊡ / ⊡	□	⊡	⊡	☒

Table 2 Legend:

☒	. . . supported
⊡	. . . support with extensions possible
□	. . . not supported

Resource management provides the assignment of users or roles to tasks and separation/binding of duty constraints are defined by selecting an option to (not) choose a user who completed a previous task.

Only in SPRINT, security policies are stored in a separate policy repository. SPRINT supports *all* security policies for PAIS. For example, data constraints restrict responsibilities in value. Furthermore, SPRINT provides task pattern constraints to enforce security on the control flow of a process and relation constraints to provide process consistency. At run time, all security policies such as separation/binding of duty or time constraints are enforced and monitored.

Another important feature in PAIS is the verification of security policies. Currently, only few applications detect conflicting policies (e.g., mutual exclusion conflicts in [33]). AristaFlow uses the *correctness by construction* design principle and displays conflicts while process modeling. DECLARE supports the detection of dead activities and conflicting constraints at design time and the verification of modified processes for history-based errors at run time [28]. Also the verification of (empty) valid actor sets, i.e., the set of users who qualify for the particular access rule (cf. [34]), is a convenient feature at design time. The detection of empty valid actor sets is only supported in SPRINT. Imagine, a role `R` is only associated with one user `U`, and a process `P` contains the tasks `T1` and `T2` (which role `R` is assigned to) having a separation of duty constraint. During process execution, this definition results automatically in an empty valid actor set for task `T2`. Most tools either automatically delegate the task to other users (e.g., supervisors, administrators) or, in worst case, do not re-offer the task to other users which might lead to workflow blockage.

In conclusion, process management tools support the specification, enforcement, and management of security policies. Most applications implement security policies rather at task level than in a separate repository. They also support additional constraints imposed on top of access rules and (authorization) constraints, such as data or time constraints. However, not all of them provide the direct specification on top of security polices. Furthermore, process management tools do not provide enough verification mechanisms for security policies to detect conflicts at design time which can lead to unexpected situations during process execution.

8 The SPRINT Prototype

Process-Aware Information Systems (PAIS) typically consist of (1) a Process Engine that instantiates and executes Process Models, and (2) a Worklist Manager that handles the allocation of tasks to users. (3) Information about organizational units, users, and roles typically resides in a back-end database. Generally, all three parts are tightly coupled components utilizing product specific interfaces and data formats.

Security policies are typically specified as part of the Process Model (see Section 7), and their enactment and validation is built into the Process Engine and the Worklist Manager. The Process Engine typically invokes machine tasks and delegates human tasks to a Worklist Manager. In terms of security policies, the Worklist Manger has to deal with allocation of users, access control, and separation of duties, while the Process Engine has to deal with all security policies regarding service invocation. Furthermore, external functionality (e.g., web services) invoked by the Process Engine may also implement the checking of own security policies as they directly control access to data. Finally, there may exist a multitude of custom or standard secondary systems (e.g., databases). The result is a zoo of systems and components with support for security at multiple levels.

In order to highlight how the presented concepts allow for an improved, loosely coupled system architecture, we will examine again the requirements:

Requirement 1 (Independence of Security Policies)**.** *Security polices should not be be intertwined with process logic. The advantage here is that the enforcement of security policies has not to be built in or to be tightly coupled with the Process Engine itself (e.g., Apache ODE uses an add-on system). Instead, it is possible to have a separate SPRINT Enactment Engine that can be reused by different Process Engines. The independent Security Policy Design and Mapping allows to keep two important pieces (Process Enactment, Policy Enactment) separate: from an architectural and a users point of view.*

Requirement 2 (Maintainability of Security Policies)**.** *Maintainability is more focused on the policies and less on the implementation. Thus, the maintainability improvements are visible in the user interface and not in the architecture of the implementation.*

Requirement 3 (Extendability of Security Policies)**.** *By keeping key components of the PAIS independent, they can evolve separately. While Responsibilities are specifically geared towards processes, Permissions can cover a lot of other systems invoked by (but independent from) the Process Engine. Such systems should be able to utilize security policies and security policy enactment, instead of duplicating them.*

Requirement 4 (Scalability of PAIS Components)**.** *In contrast to traditional approaches, where every task has its own security related annotations, with the mapping approach, a conceptual security policy change does not lead to a plethora of changes in tasks, but only affects one security policy. This reduces data duplication and potentially solves all kinds of errors that can occur when massive changes to multiple entities (in this case tasks) have to be made. At runtime, on the other hand, the lookup of security policies through responsibilities and their mapping approach is considerably more demanding than the simple lookup necessary when a policy is directly annotated to a process task. Simple lookup caching, however, can bring down the effort to the same level as direct annotations.*

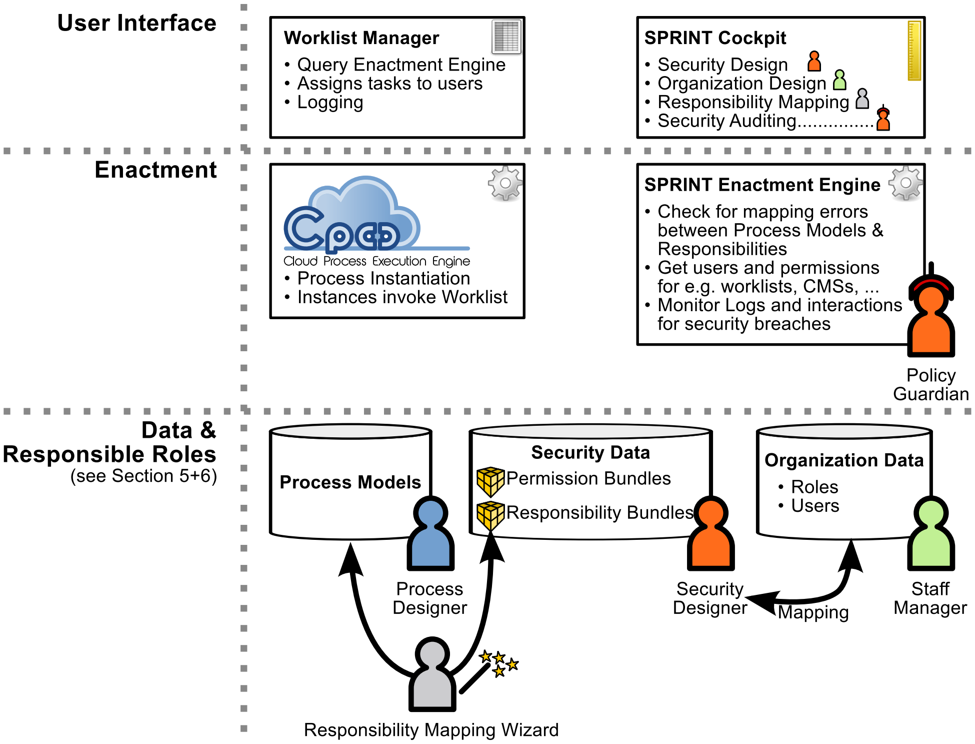
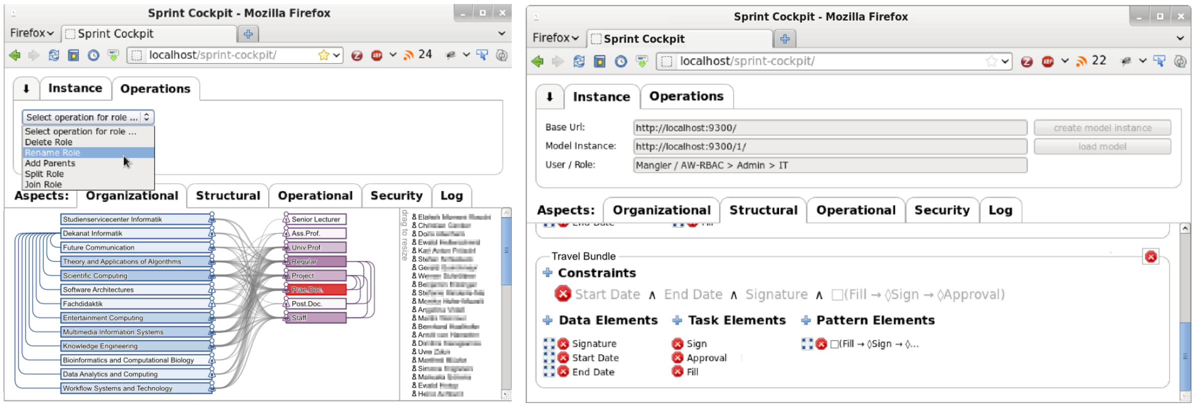


Figure 6: Prototype Architecture



(a) Organizational Aspect

Base Url: http://localhost:9300/

Model Instance: http://localhost:9300/1/

User / Role: Mangler / AW-RBAC > Admin > IT

Travel Bundle

Constraints

Start Date ∧ End Date ∧ Signature ∧ □(Fill → ◊Sign → ◊Approval)

Data Elements	Task Elements	Pattern Elements
Signature	Sign	□(Fill → ◊Sign → ◊...
Start Date	Approval	
End Date	Fill	

(b) Structural Aspect

Figure 7: SPRINT Cockpit: Security Bundle Design and Mapping

Figure 6 depicts the User Interfaces, Enactment Components, Data Stores, and the roles utilizing them. The figure is an aggregated and grouped overview of the concepts described in the requirements above and in preceding chapters. In order to demonstrate the flexibility of our approach, we decided to integrate with the lightweight Cloud Process Execution Engine (CPEE) [35]. The purpose of the SPRINT Enactment Engine can be summarized as:

- Select users on behalf of the Worklist Manager.
- Select resources on behalf of the Process Engine.
- Grant data access on behalf of external functionality (e.g., web services).

- Monitor the behavior of the Process Engine and the Worklist and the communication with web services in order to detect irregularities.

In order realize the administration of Responsibilities, Permissions, and their Mapping to Organization Structures and Processes, we created a simple web-based user interface (SPRINT Cockpit) that directly reflects the concepts described in this paper (see Fig. 7). Fig. 7a shows the organization editor, including organizational units on the left, roles in the middle, and a list of users on the right. Fig. 7b displays the user interface for designing bundles. Data Elements and Tasks can be collected. Then, Tasks can be grouped via drag & drop as Pattern Elements (LTL expressions). Finally, Data Elements and Pattern Elements can be organized in constraints.

As can be seen in Fig. 7b, creating Data Elements and Task Elements is a matter of entering free text statements. Only through mapping the system becomes functional. Extending this procedure to cover, for example, web services is only a matter of additional mapping, thus, a holistic security solution is conceivable.

9 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 Section 3. The results of the evaluation are shown in Table 3.

Table 3: *Evaluation of existing Approaches*

Policy	Independence	Maintainability	Extendability	Scalability
NIST RBAC [18], W-RBAC [36]	+			
ARBAC [37]	+	+	+	
Bertino et al [5], Casati et al [38]	+			
Riberio et al [39]	+	+	+	
Neumann et al [3]	+	$\sim$	$\sim$	
SPRINT	+	+	+	+

The Role-Based Access Control (RBAC) model (e.g., NIST RBAC [18]) expresses security policies based on the role-permission assignments. Further models such as Administrative RBAC models (e.g., ARBAC [37]) or PAIS related RBAC models (e.g., Workflow RBAC [36]) have been developed. Whereas these models depend on abilities of authorized users to perform tasks (e.g., set in job description), the SPRINT approach uses responsibilities, such as data objects or tasks related to permissions and roles, to develop security policies. Even though RBAC models enable Independence of Security Policies (R1), only administrative models enable maintenance features (R2, R3).

The specification and enforcement of constraints in PAIS are proposed in [3, 5, 38]. They support Independence (R1) but most of them ignore Maintainability of Security Policies (R2), Extendibility of Security Policies (R3), and Scalability of PAIS components (R4). SPRINT enforces static and dynamic authorization *and* assignment constraints. In [39], workflow processes are verified against organization security policies by transforming each in a common constraint language. This way, Scalability of PAIS Components (R4) can become an issue when transforming and processing a large amount of data. In the SPRINT approach, inconsistencies only have to be checked when policies are created or changed.

The related work discussion showed that SPRINT provides 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.

10 Conclusion

This paper presented design and development techniques for security policies in PAIS, mainly aiming at a clear separation of aspects between business processes / workflows and security policies. The current "mix" of both aspects, hampers consistency checks and enforcement of the security policies on one side, and maintenance and evolution of processes and associated policies on the other side. The SPRINT approach separates security policies from business processes by enriching RBAC models with structural aspects (responsibilities) as used in PAIS. Using SPRINT, all different kinds of security policies can be expressed, such as access rules and authorization constraints, and they can be easily connected to the business processes based on a simple mapping. The independent definition of security policies supports their management and maintenance in an easy way. Further on, explicitly defined security policies enable their extension by any kind of further constraint with respect to, for example, location, time, or data. By separating security policies and business processes, the SPRINT approach can be deployed on *any* PAIS regardless of the process modeling and system used. This paper formally defines SPRINT fundamentals. They are evaluated based on a literature review and an extensive comparison of existing process notations, modeling tools, and process management tools. Additionally, the SPRINT approach has been implemented within a prototype presented in this paper. In future work, we will elaborate on our SPRINT prototype and further demonstrate the feasibility of the approach based on integration of SPRINT with existing process management systems. We expect a completely new set of questions when extending our considerations to cross-organizational process settings.

Acknowledgements

This work was partially supported by the Commission of the European Union within the ADVENTURE FP7-ICT project (Grant agreement no. 285220).

References

[1] M. Leitner, S. Rinderle-Ma, and J. Mangler, "Responsibility-driven design and development of process-aware security policies," in *Proc. of the 6th International Conference on Availability, Reliability and Security (ARES'11), Vienna, Austria*. IEEE, August 2011, pp. 334–341.

[2] N. Russell, W. van der Aalst, A. H. ter Hofstede, and D. Edmond, "Workflow resource patterns: Identification, representation and tool support," in *Proc. of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05), Porto, Portugal, LNCS*. Springer-Verlag, June 2005, vol. 3520, pp. 216–232.

[3] G. Neumann and M. Strembeck, "An approach to engineer and enforce context constraints in an RBAC environment," in *Proc. of the 8th ACM symposium on Access control models and technologies (SACMAT'03), Como, Italy*. ACM, June 2003, pp. 65–79.

[4] P. C. K. Hung and K. Karlapalem, "A secure workflow model," in *Proc. of the Australasian information security workshop (AISW'03), Adelaide, Australia*, vol. 21. Australian Computer Society, February 2003, pp. 33–41.

[5] E. Bertino, E. Ferrari, and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 65–104, February 1999.

[6] W. M. P. van der Aalst, H. de Beer, and B. van Dongen, “Process mining and verification of properties: An approach based on temporal logic,” in *Proc. of the 13th International Conference on Cooperative Information Systems (CoopIS’05), Agia Napa, Cyprus, LNCS*, vol. 3761. Springer-Verlag, October-November 2005, pp. 130–147.

[7] S. Rinderle-Ma and J. Mangler, “Integration of process constraints from heterogeneous sources in Process-Aware information systems,” in *Proc. of the 4th International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA’11), Hamburg, Germany, LNI*. Gesellschaft für Informatik, September 2011, pp. 51–64.

[8] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing, 2008.

[9] S. Rinderle-Ma and M. Leitner, “On evolving organizational models without losing control on authorization constraints in web service orchestrations,” in *Proc. of the 12th IEEE International Conference on Commerce and Enterprise Computing (CEC’10), Shanghai, China*. IEEE Computer Society, November 2010, pp. 128–135.

[10] J. Park and R. Sandhu, “The UCON ABC usage control model,” *ACM Transactions on Information and System Security*, vol. 7, no. 1, pp. 128–174, February 2004. [Online]. Available: http://doi.acm.org/10.1145/984334.984339

[11] T. D. Breaux and A. I. Anton, “Analyzing goal semantics for rights, permissions, and obligations,” in *Proc. of the 13th IEEE International Conference on Requirements Engineering (RE’05), Paris, France*. IEEE, September 2005, pp. 177–186.

[12] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso, “Specifying and analyzing early requirements in tropos,” *Requirements Engineering*, vol. 9, no. 2, pp. 132–150, May 2004.

[13] B. H. C. Cheng and J. M. Atlee, “Research directions in requirements engineering,” in *Proc. of the Workshop on the Future of Software Engineering (FOSE’07), Minneapolis, Minnesota, USA*. IEEE, May 2007, pp. 285–303.

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

[15] E. J. Coyne, “Role engineering,” in *Proc. of the 1st ACM Workshop on Role-based access control (RBAC’95), Gaithersburg, Maryland, USA*. ACM, November-December 1996.

[16] G. Neumann and M. Strembeck, “A scenario-driven role engineering process for functional RBAC roles,” in *Proc. of the 7th ACM symposium on Access control models and technologies (SACMAT’02), Monterey, California, USA*. ACM, June 2002, pp. 33–42.

[17] M. Kuhlmann, D. Shohat, and G. Schimpf, “Role mining - revealing business roles for security administration using data mining technology,” in *Proc. of the 8th ACM symposium on Access control models and technologies (SACMAT’03), Como, Italy*. ACM, June 2003, pp. 179–186.

[18] R. Sandhu, D. Ferraiolo, and R. Kuhn, “The NIST model for role-based access control: towards a unified standard,” in *Proc. of the 5th ACM workshop on Role-based access control (RBAC’00), Berlin, Germany*. ACM, July 2000, pp. 47–63.

[19] M. A. Al-Kahtani and R. Sandhu, “A model for attribute-based user-role assignment,” in *Proc. of the 18th Annual Computer Security Applications Conference (ACSAC’02), Las Vegas, Nevada, USA*. IEEE, December 2002, pp. 353–362.

[20] B. Weber, M. Reichert, W. Wild, and S. Rinderle-Ma, “Providing integrated life cycle support in Process-Aware information systems.” *International Journal of Cooperative Information Systems*, vol. 18, no. 1, pp. 115–165, March 2009.

[21] M. Pesic and W. van der Aalst, “A declarative approach for flexible business processes management,” in *Proc. of the 1st International Workshop on Dynamic Process Management (DPM’06), Vienna, Austria, LNCS*, vol. 4103. Springer-Verlag, September 2006, pp. 169–180.

[22] W. M. P. van der Aalst, “Formalization and verification of event-driven process chains,” *Information & Software Technology*, vol. 41, no. 10, pp. 639–650, July 1999.

[23] ——, “Verification of workflow nets,” in *Proc. of the 18th International Conference on Application and*

Theory of Petri Nets (ICATPN'97), Toulouse, France, LNCS, vol. 1248. Springer-Verlag, June 1997, pp. 407–426.

[24] C. Wolter and A. Schaad, "Modeling of Task-Based authorization constraints in BPMN," in *Proc. of the 5th International Conference on Business Process Management (BPM'07), Brisbane, Australia, LNCS*, vol. 4714. Springer-Verlag, September 2007, pp. 64–79.

[25] K. v. Hee, N. Sidorova, and M. Voorhoeve, "Resource-Constrained workflow nets," *Fundamenta Informaticae*, vol. 71, no. 2-3, pp. 243–257, June 2006.

[26] P. Dadam, M. Reichert, S. Rinderle-Ma, A. Lanz, R. Pryss, M. Predeschly, J. Kolb, L. T. Ly, M. Jurisch, U. Kreher *et al.*, "From ADEPT to AristaFlow BPM suite: A research vision has become reality," in *Proc. of the 1st International Workshop on Empirical Research in Business Process Management (ER-BPM'09), Ulm, Germany, LNBIP*, vol. 43. Springer-Verlag, September 2009.

[27] L. T. Ly, D. Knuplesch, S. Rinderle-Ma, K. Göser, H. Pfeifer, M. Reichert, and P. Dadam, "SeaFlows toolset – compliance verification made easy for Process-Aware information systems," in *Proc. of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10), Hammamet, Tunisia, LNBIP*, vol. 72. Springer-Verlag, January 2011, pp. 76–91.

[28] M. Pesic, H. Schonenberg, and W. v. d. Aalst, "DECLARE: full support for Loosely-Structured processes." in *Proc. of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC'07), Annapolis, Maryland, USA*. IEEE, October 2007, pp. 287–300.

[29] W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245–275, June 2005.

[30] L. Ly, S. Rinderle-Ma, and P. Dadam, "Design and verification of instantiable compliance rule graphs in Process-Aware information systems," in *Proc. of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10), Hammamet, Tunisia, LNCS*, vol. 6051. Springer-Verlag, June 2010, pp. 9–23.

[31] T. Ly, S. Rinderle-Ma, K. Göser, and P. Dadam, "On enabling integrated process compliance with semantic constraints in process management systems – requirements, challenges, solutions," *Information Systems Frontiers*, pp. 1–25, June 2009.

[32] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, P. Trickovic, A. Yiu, and M. Zeller, "WS-BPEL extension for people (BPEL4People)," Specification Version 1.0, June 2007, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf, last accessed 12/15/2011. [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf

[33] S. Schefer, M. Strembeck, J. Mendling, and A. Baumgrass, "Detecting and resolving conflicts of Mutual-Exclusion and binding constraints in a business process context," in *Proc. of the 19th International Conference on Cooperative Information Systems (CoopIS'11), Crete, Greece, LNCS*, vol. 7044. Springer-Verlag, October 2011, pp. 329–346.

[34] S. Rinderle-Ma and M. Reichert, "Comprehensive life cycle support for access rules in information systems: The CEOSIS project," *Enterprise Information Systems*, vol. 3, no. 3, pp. 219–251, July 2009.

[35] G. Sturmer, J. Mangler, and E. Schikuta, "Building a modular service oriented workflow engine," in *Proc. of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA'09), Taipei, Taiwan*. IEEE, January 2009, pp. 1–4.

[36] 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, December 2003.

[37] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 105–135, February 1999.

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

[39] C. Ribeiro and P. Guedes, "Verifying workflow processes against organization security policies," in *Proc. of the 8th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises (WETICE'99),*

Stanford, California, USA. IEEE, June 1999, pp. 190–191.

Maria Leitner is a PhD Student at the Workflow Systems and Technology Group at the Faculty of Computer Science, University of Vienna. Her PhD thesis is on providing security in adaptive process-aware information systems. The goal is not only to develop specific security mechanisms but also to support security throughout the workflow life cycle. Further research interests are business process modeling, business process compliance, and security related fields in adaptive process-aware information systems.

Juergen Mangler is currently finishing his PhD thesis on monitoring and repair in distributed process management systems at the University of Vienna, Workflow Systems and Technology group. His current research is focused on service oriented architectures for mobile devices, and on a light-weight modular process engine to fully support external monitoring and intervention. He further published in the field of RESTful service description, composition and evolution.

Stefanie Rinderle-Ma is a full professor and head of the research group Workflow Systems and Technology at the Faculty of Computer Science, University of Vienna, Austria. At the same time, she serves as key researcher at Secure Business Austria Research. She obtained her Ph.D. and Habilitation degree in Computer Science from Ulm University, Germany. Her research interests include adaptive processes and services, business process compliance, as well as security and privacy aspects in process-aware information systems.