

# Task-Based Entailment Constraints for Basic Workflow Patterns

Christian Wolter  
SAP Research  
Vincenz-Priessnitz-Str. 1  
76131, Karlsruhe, Germany  
christian.wolter@sap.com

Andreas Schaad  
SAP Research  
Vincenz-Priessnitz-Str. 1  
76131, Karlsruhe, Germany  
andreas.schaad@sap.com

Christoph Meinel  
Hasso-Plattner-Institute  
University of Potsdam,  
Germany  
meinel@hpi-uni-  
potsdam.de

## ABSTRACT

Access Control decisions are based on the authorisation policies defined for a system as well as observed context and behaviour when evaluating these constraints at runtime. Workflow management systems have been recognised as a primary source for defining authorisation policies at workflow design-time, as well as generating context at runtime.

This paper analyses recent work in the workflow community regarding established control-flow patterns. We claim that there is an intrinsic relationship between these patterns and a set of task-based entailment constraints - such as Separation of Duty - that have been recently identified by the access control community. These constraints are based on a pre-determined partial order on sequence and parallel execution patterns. When, however, such an order does not exist, because of more complex control-flow patterns, ambiguous constraint evaluation situations will arise at workflow runtime.

Accordingly, this paper reviews basic workflow patterns and identifies relationships between these and task-based entailment constraints. In addition, an analysis of possible runtime ambiguities that may arise from these relationships is presented. Our approach is based on recently developed techniques for visual constraint representation at a workflow design-time.

## Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*access control*

## General Terms

Security

## Keywords

Access Control, Separation of Duty, Task-Based Entailment Constraints, Workflow Management, Workflow Modelling

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SACMAT'08, June 11–13, 2008, Estes Park, Colorado, USA.  
Copyright 2008 ACM 978-1-60558-129-3/08/06 ...\$5.00.

## 1. INTRODUCTION

A workflow separates various activities of organisational processes into a set of well-defined tasks [1]. Tasks in such a workflow are usually carried out by several users in accordance with their organisational roles relevant to the process represented by the workflow.

Controlling work allocation and execution in a workflow is a recognised fundamental principle of user interaction in computer security. Organisational goals must be complemented by control goals such as realised through authorisation constraints [2]. One of the earliest authorisation constraints for user work allocation control is the *four-eyes principle* first to appear in Saltzer and Schroeder [3]. Later the term *separation of duty* was introduced in [4] as a principle for preserving integrity and a mechanism for error control and fraud prevention.

These concepts are applied to a workflow by limiting a user's work allocation statically at the point of the workflow definition and dynamically at workflow runtime [5]. In the former, a user's work allocation is limited by assigning a role with a fixed set of tasks. In the latter, a user's work allocation is constrained depending on the tasks the user recently performed. In [6], a taxonomy for different kinds of static and dynamic *separation of duty* is given based on the conflicting tasks paradigm, that implies that the risk of fraud increases if the associations within a set of conflicting tasks  $T_c$  are not carefully controlled and monitored.

Such specification of authorisation constraints is formally defined in [7] for workflow management systems. The specification is based on the assumption that: "The workflow tasks are sequentially executed in order of appearance in the workflow role specification" [7]. In [8] these concepts are further refined, by entailment and cardinality constraints that restrict task authorisation depending on the execution history of a given workflow instance in a more fine-grained manner for adjacent tasks.

We investigated if it is possible to enrich the semantics and syntax of a workflow modelling notation based on these formal definitions. We defined annotation elements that could be used to visually represent several types of role-based, task entailment, and task cardinality constraints in a workflow model at design-time [9].

Workflow modelling covers several perspectives with different concerns. In this context we focus on the control-flow perspective that deals with aspects of modelling workflow execution sequences consisting of routing, synchronisation, and merging. In [10], over 40 control-flow modelling patterns

are presented and their execution semantics are formally described. In this paper we focus on the basic control-flow patterns. These patterns are supported by almost all state-of-the-art workflow modelling notations, and can be executed at runtime by most of recent workflow execution engines [10, 11]. The basic patterns address the modelling of parallel execution branches and the merging of these branches.

Therefore, consider the following workflow description given in Figure 1 consisting of four human tasks, where task  $t_2$  and task  $t_3$  are executed in parallel. We define a workflow  $W$  as a 4-tuple  $\langle T, F, c_s, c_e \rangle$ . Let  $c_s$  be the start condition of the workflow. The end condition of the workflow is denoted as  $c_e$ .  $T$  is a set of tasks. In this example  $T$  describes the set of tasks  $\{t_1, t_2, t_3, t_4\}$ . The control-flow relation  $F$  (i.e., task sequence, splits, and joins) of all tasks in  $T$  is defined by  $F \subseteq (c_s \times T) \cup (T \times c_e) \cup (T \times T)$ . That means every task in the workflow  $(T, F)$  is on a partially ordered path  $p = (T, <)$  from  $c_s$  to  $c_e$ . If task  $t_n < t_{n+1}$ , then  $t_{n+1}$  depends on  $t_n$  and must be executed after  $t_n$  in  $p$ .

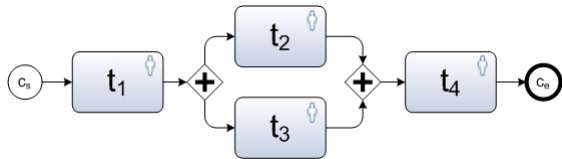


Figure 1: Simple Parallel Execution

## 1.1 Problem Statement

Existing work on workflow-based authorisation specification, especially the definition of task-based entailment constraints [8], is based on ordered conflicting tasks [6], implying that the temporal order of task execution is already known at design-time, i.e. by forced serialised workflow execution [7] or a partial order  $t_n < t_{n+1}$  for conflicting tasks [8]. Referring to our example this is appropriate for  $T_c = \{(t_3, t_4) | t_3 < t_4\}$ . Consider, however, the case where an entailment constraint is defined on  $T_c = \{(t_2, t_3)\}$ . Assuming that two human tasks can not be performed simultaneously, two partial orders are possible at runtime either  $t_2 < t_3$  or  $t_3 < t_2$ .

Existing specifications deal with this by an enforced serialisation of all parallel branches to derive a single execution sequence, such as it is common practice in the domain of database transactions, that does not violate any authorisation constraints. Nevertheless, sequential workflows are “unnecessary constraints on the process logic” [12] and performance is a very important indicator for business analysts. Therefore, workflows that contain a high number of concurrent execution paths are very common in today’s enterprise business scenarios. This is further emphasised by the large number of control-flow patterns used to describe several kinds of runtime branching, task cancellation, and advanced branch synchronisation that result in various execution paths with an arbitrary path length depending on the business logic [10].

The question is, what happens in case of entailment constraints being specified on a set of tasks that may result in execution paths with a varying path length at runtime. Existing consistency analysis proposals predetermine a dedicated path execution that satisfies defined entailment constraints [13]. Complex workflows consist of various alterna-

tive paths and complex branching with different path length observable at runtime. Therefore, we need to question whether is it possible that for constrained tasks, the execution semantics of control-flow patterns may result in an ambiguous constraint evaluation at runtime or may even lead to a deadlock-like situation disrupting the overall workflow execution.

To answer these questions we will present a modified entailment specification during the course of this paper and provide example constraints in a workflow model, evaluate dependencies on the execution semantics of basic workflow patterns, and discuss potential issues that may occur at runtime.

In essence, the contributions of this paper are as follows:

- it defines an entailment specification consolidating related authorisation requirements that are common in the domain of workflow management and presents entailment constraint patterns for workflow modelling.
- it evaluates the execution semantics of basic control-flow patterns and reveals ambiguities regarding the partial order of tasks at runtime.
- it outlines dependencies between entailment constraints and the control-flow of constrained tasks and discusses potential impact on the runtime constraint evaluation.
- it further fosters the idea of business process experts and security experts to communicate, define, and discuss security concerns over a common abstract model.

## 1.2 Organisation of the Paper

The rest of this paper is structured as follows: We provide an entailment constraint specification without implying any temporal order or limitation to exactly two tasks in Section 2. In addition we present modelling examples depicting how these constraints could be expressed as part of the workflow model. This is followed by a brief overview of selected basic control-flow modelling patterns in Section 3 along with an evaluation of their execution semantics in terms of a graph depicting all possible execution paths at runtime. This leads to a follow-up discussion in Section 4 about constrained tasks and their control-flow semantics in the context of workflow models and potential side effects. In Section 5 we discuss related work for security modelling concepts in the domain of workflow models and existing fine-grained authorisation specifications. In the last section we outline potential extensions to our existing concepts with respect to more complex control-flow patterns exclusive data access in terms of Chinese Wall policies or other aspects, such as compliance modelling [14] and risk assessment [15].

## 2. SPECIFICATION OF CONSTRAINTS IN WORKFLOWS

Fine-grained specification of authorisation constraints has been investigated for a long time in the literature [2, 6, 7, 8, 16, 17, 18]. Concepts were presented investigating how authorisation, risk assessment, or compliance artefacts can be specified as part of a workflow model [9, 14, 15, 19]. They emphasise the collaboration between the technical security expert and business process domain expert. Ideally, the task of defining security aspects is done at the workflow model itself. Therefore, both business process and technical security

experts may collaborate on a security augmented workflow. Each expert is still responsible for his domain specific activities, but the communication takes place over a shared workflow model.

These concepts of authorisation and security modelling in workflows - as depicted in Figure 2 - are based on two key observations:

1. Several well defined, standardised, and accessible business process modelling notations are available that can be used by the business process domain expert at an appealing and abstract level that can be easily communicated to various stakeholders.
2. Security experts however, can only specify enforceable authorisation policies and security metrics (e.g. confidentiality) on a very technical level, instead of directly in the context of the workflow and business process expert domain. Valuable domain knowledge of the business stakeholder, for instance knowledge regarding compliance constraints, is lost or its refinement not intuitively traceable.

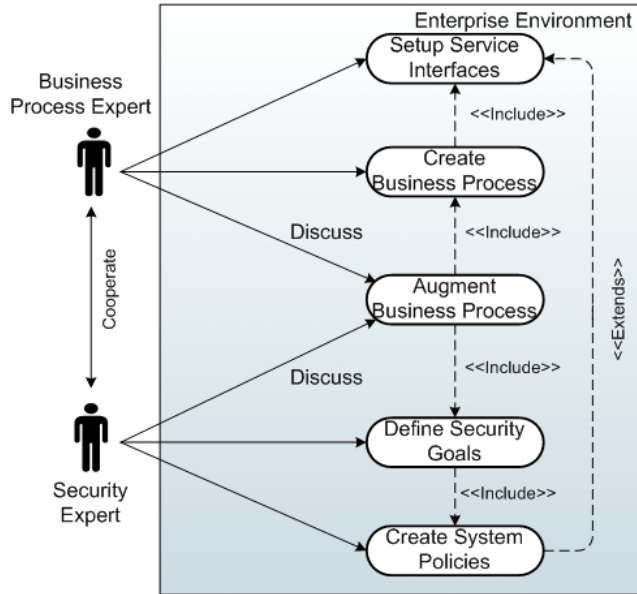


Figure 2: Combining Security and Workflow Domain

## 2.1 Task-Based Entailment Constraints

In recent work we demonstrated a method that applies existing research results for task-based entailment specification into the domain of workflow modelling [9], and applied a model-driven approach to generate platform-specific security policies based on the modelled constraints [20]. As an example modelling notation we selected the Business Process Modelling Notation (BPMN) [21]. BPMN essentially provides a graphical notation for business process modelling, with an emphasis on control-flow.

Task-based entailment constraints are based on the conflicting entities paradigm, e.g. tasks in a workflow. In the domain of workflow management systems, conflicting tasks imply that the risk of fraud increases if the associations with

those tasks are not carefully controlled and monitored [6]. A set of conflicting tasks of a workflow model is denoted as  $T_c$ , accordingly a task  $t \in T_c$  is called a constrained task [8]. In order to specify task-based constraints as part of the workflow model we extend the definition of a workflow  $W$ :

$$\begin{aligned}
 W &= \langle T, F, c_s, c_e, T_c, C, U(T_c) \rangle \\
 T_c &\subseteq T \text{ set of conflicting tasks} \\
 C &= \text{set of entailment constraints, } \{c_1, c_2, \dots, c_k\} \\
 U(T_c) &= \text{determines past task allocation of } t \in T_c
 \end{aligned}$$

In the literature each task-based constraint  $c \in C$  defines a task allocation restriction on one or two tasks  $\in T_c$ . In general, a user would be authorised to allocate a task based on his role and the role-task assignment relation of the workflow. A constraint specifies additional conditions that must hold beyond the user's role at runtime. If the allocation of a task  $t_n$  is constrained based on the previous allocation of another task  $t_m$  in the same workflow instance, with  $t_m, t_n \in T_c$ , such a constraint is called an entailment constraint. A constraint that restricts the allocation of a task  $t_n$  based on the number of previous task allocations of the same task for a given workflow instance is called a (local) cardinality constraint [8].

The constraint specification limited to sets of tasks containing one or two tasks is a drawback when it comes to history-based or operational *separation of duty* in workflows. A history-based or operational *separation of duty* [2] defines an entailment constraint on a set of tasks  $T_k \subseteq T_c$ . At least  $n_u$  users must perform a task  $t \in T_k$ . In addition no single user is allowed to perform more than  $k$  tasks  $t \in T_k$ .

This also applies to a repetitive allocation of the same task  $t \in T_k$ , used to express a local cardinality constraint as an operational *separation of duty* constraint based on  $|T_k| = 1$  and the values  $n_u$  and  $k$ . Accordingly, we define a task-based entailment constraint  $c \in C$  as:

$$c = (T_k, n_u, k); T_k \subseteq T_c; n_u, k \in \mathbb{N}.$$

$T_k$  relates to the tasks in  $T_c$  that are affected by this constraint. The number of different users that must allocate at least one task  $t \in T_k$  is denoted as  $n_u$ . The maximum number of tasks in  $T_k$  that can be allocated, such that a new task instance is created at runtime by a single user is denoted by the threshold value  $k$ . In order to avoid a situation where no task can be allocated by no user any more due to restrictive values for  $n_u$  and  $k$ , we require:

$$0 < |T_k| \leq n_u * k; \forall n_u, k \in \mathbb{N}.$$

We specified [9] such constraints as part of a workflow model. Thus, Figure 3 depicts two examples how to represent such a constraint as part of a workflow model.

A dotted association line or box is used to define the set of constrained tasks  $T_k$ . Two digits or small icons are used to represent  $n_u$  and  $k$ . Based on this specification we are able to express well-known types of entailment constraints, such as *separation of duty* or *binding of duty*. In what follows we will provide examples for classical constraints, such as the *four-eyes-principle*. It should be noted that at this point, we focus only on the specification of  $T_k, n_u$ , and  $k$  as part of the model, so the examples do not contain any control-flow information between tasks in  $T_k$ . This will be part of our discussion in Section 3 and 4.

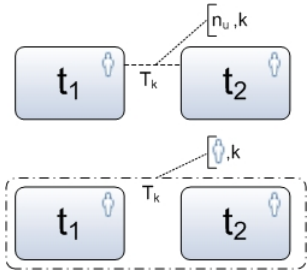


Figure 3: Workflow Annotation

## 2.2 Constraint Examples

Based on the given entailment specification we provide modelling examples for *four-eyes-principle*, *binding of duty*, and history-based *separation of duty*. The latter one is a general case of the more specialised *four-eyes-principle*.

### Four-Eyes-Principle

The *four-eyes-principle* is the classical example for a *separation of duty* constraint in terms of defining two tasks that must be executed by two different users. Using our entailment constraint definition, we express the *four-eyes-principle* on  $T_k$  as:

$$\begin{aligned} c_{SoD} &= (T_k, 2, 1) \\ |T_k| &= 2 \\ n_u &= 2 \\ k &= 1 \end{aligned}$$

The four eyes-principle is enforced by a set size  $|T_k| = 2$ , the requirement that 2 users must allocate at least one task  $t \in T_k$ , and that not more than one task  $t \in T_k$  can be allocated in a workflow instance by the same user. The combination of  $n_u = 2, k = 1$  is equivalent to the predicate  $\neq$  that is used in [8] to denote a *separation of duty* constraint. Referring to Figure 4 a *separation of duty* entailment constraint example would be  $c_{SoD} = (\{t_1, t_2\}, 2, 1)$ .

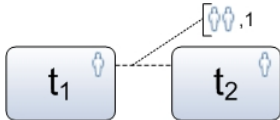


Figure 4: Separation of Duty Pattern

### Binding of Duty

A *binding of duty* constraint is another special case of history-based *separation of duty*. In principle this type of constraint enforces that two tasks must be performed by the same person. Such constraints reflect requirements that, for instance confidential data is only accessed by a minimal set of users. A *binding of duty* on an arbitrary set of tasks can be defined by the following constraint:

$$\begin{aligned} c_{BoD} &= (T_k, 1, 2) \\ 1 &< |T_k| \\ 1 &= n_u \\ 2 &= |T_k| \end{aligned}$$

We enforce the *binding of duty* by specifying that one user must allocate all the tasks in  $T_k$ . In order to do so we must

also set the threshold value  $k = |T_k|$ . Referring to Figure 5 an example *binding of duty* entailment constraint would be  $c_{BoD} = (\{t_1, t_2\}, 1, 2)$ .

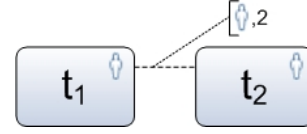


Figure 5: Binding of Duty Pattern

### History-based Separation of Duty

According to [2] a history-based *separation of duty* constraint states that a principal may execute a set of tasks that may cover an entire workflow, though must not execute all tasks on the same workflow instance. Thus, a general history-based *separation of duty* constraint on  $T_k$  is defined as:

$$\begin{aligned} c_{hSoD} &= (T_k, n_u, k) \\ 1 &< |T_k| \leq n_u * k \\ 1 &< n_u \leq |T_k| \\ \lfloor (|T_k|/n_u) \rfloor &\leq k \leq \lceil (|T_k|/n_u) \rceil \end{aligned}$$

Referring to Figure 6 a history-based *separation of duty* entailment constraint modelling example would be  $c_{hSoD} = (\{t_1, t_2, t_3\}, 2, 2)$ .

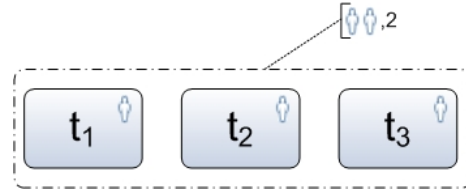


Figure 6: History-Based Separation of Duty Pattern

## 3. BASIC WORKFLOW PATTERNS

In the research field of workflow definition, various modelling patterns have emerged. These patterns can be classified as control-flow, data-flow, or resource centric. In this section we investigate on basic control-flow patterns focusing on the aspect of workflow execution semantics, such as sequence flows, routing, synchronization, and merging. Patterns related to the data perspective address issues such as data visibility, data interaction, data transfer, and data-based routing. Patterns dealing with resources, or rather those describing the manner in which work is distributed and managed by human resources associated with a workflow [22] are not addressed. As stated before, our selection of basic control-flow patterns is motivated by the fact that almost all modelling notations and workflow execution engines support these basic patterns [10, 11].

### 3.1 Control-Flow Pattern

The set of workflow patterns in the control-flow perspective are derived from a detailed examination of contemporary workflow systems and business process modelling notations in order to identify generic, recurring constructs [23]. We present basic control-flow related patterns for workflow tasks and their potential observed runtime behaviour in terms of observable execution paths. Depending on the control-flow semantics of each pattern, several execution paths are likely at runtime. Which path is chosen depends on the context of the current workflow instance - which is an instantiation of a workflow model - and the processing time of each human task. The graph denotes all alternative execution paths based on the patterns, without considering their likelihood of occurrence at runtime. Further we assume that each edge in a graph is of equal weight. As the basic patterns do not address multiple instances of a task in a workflow, all paths are open, in that they are acyclic and share common root and end nodes ( $c_s$  and  $c_e$  respectively), with  $c_s \neq c_e$ .

Therefore, we define a set of workflow execution paths for a given workflow  $W = \langle T, F, c_s, c_e \rangle$  as a rooted tree graph  $G$  consisting of vertices  $V$  and edges  $E$ . We refer to a path by the natural sequence of its vertices by writing  $t_1Pt_2 := t_1, t_2$ :

$$\begin{aligned} G &= (V, E) \\ V(G) &= \{T \cup \{c_s, c_e\}\} \\ E(G) &\subseteq (c_s \times T) \cup (T \times c_e) \cup (T \times T) \\ c_s &= \text{the root node, } t_1 \leq t_2 | t_1 \in c_sPt_2 \text{ in } G \\ |P_i| &= \text{length of a path } P_i \text{ in } G \end{aligned}$$

In order to enumerate all vertices we label each vertex with the name of the related task. Unlike in classical graph theory, vertices related to the same task in the workflow model have the same label if they occur multiple times in the graph  $G$ .

#### Sequence Pattern:

The Sequence pattern serves as the fundamental building block for any workflow. It is used to construct a series of consecutive tasks which execute once, one after the other (cf. Figure 7(a)). Two tasks  $t_1$  and  $t_2$  form part of a sequence if there is a control-flow edge  $t_1t_2$  which has no conditions associated with it (e.g., OR-Split). The execution path  $P_1$  is depicted in Figure 7(b):

$$\begin{aligned} V(G) &= V(P_1) \\ P_1 &= (\{c_s, t_1, t_2, t_3, c_e\}, \{c_st_1, t_1t_2, t_2t_3, t_3c_e\}) \\ |P_1| &= 4 \end{aligned}$$

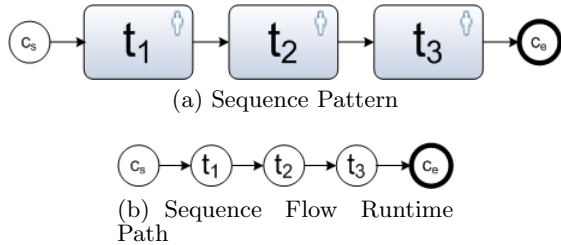


Figure 7: Sequence Pattern

#### Parallel Split Pattern:

The Parallel Split or *AND-Split* pattern allows a single thread of execution to be split into two or more branches, which can execute tasks concurrently. These branches may or may not be synchronized at some point in future time. This second basic pattern provides two execution paths  $P_1$  and  $P_2$  in  $G$  with an alternating execution orders of the tasks  $t_2$  and  $t_3$  as shown in Figure 8(a).

$$\begin{aligned} V(G) &= V(P_1) \cup V(P_2) \\ P_1 &= (\{c_s, t_1, t_2, t_3, c_e\}, \{c_st_1, t_1t_2, t_2t_3, t_3c_e\}) \\ P_2 &= (\{c_s, t_1, t_2, t_3, c_e\}, \{c_st_1, t_1t_3, t_3t_2, t_2c_e\}) \\ |P_1| &= |P_2| = 4 \end{aligned}$$

Note that while the two flow relations  $t_2t_3$  and  $t_3t_2$  are equivalent under any interleaving equivalence notion, we would however, consider them to be semantically different [24] as their execution order differs.

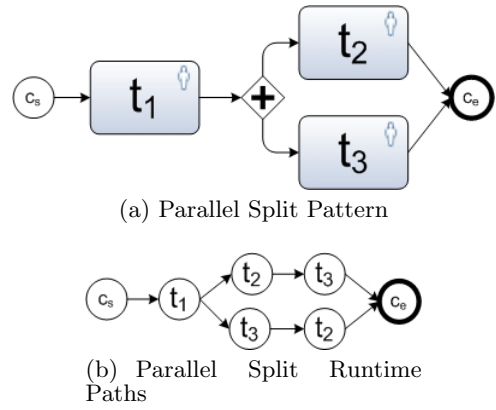


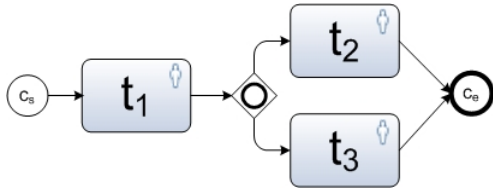
Figure 8: Parallel Split Pattern

#### Multi Choice Pattern:

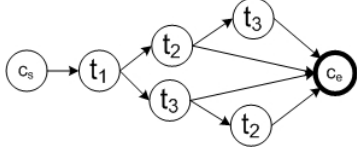
The multi choice or *OR-Split* pattern describes the divergence of a branch into two or more branches. When the task  $t_1$  is completed, the thread of control is passed to one or more of the outgoing branches based on the context of the workflow instance. This means that any combination of task  $t_2$  and task  $t_3$  is possible to occur at runtime as depicted by the four paths in  $G$  depicted by Figure 9(b).

In this example four different paths exists. Unlike the previous examples the paths are not of equal length, meaning not all tasks must be executed in a given workflow instance in order to reach the end condition  $c_e$ . Figure 9(b) depicts the various execution paths that may occur:

$$\begin{aligned} V(G) &= V(P_1) \cup V(P_2) \cup V(P_3) \cup V(P_4) \\ P_1 &= (\{c_s, t_1, t_2, t_3, c_e\}, \{c_st_1, t_1t_2, t_2t_3, t_3c_e\}) \\ P_2 &= (\{c_s, t_1, t_2, c_e\}, \{c_st_1, t_1t_2, t_2c_e\}) \\ P_3 &= (\{c_s, t_1, t_3, c_e\}, \{c_st_1, t_1t_3, t_3c_e\}) \\ P_4 &= (\{c_s, t_1, t_3, t_2, c_e\}, \{c_st_1, t_1t_3, t_3t_2, t_2c_e\}) \\ |P_1| &= |P_4| = 4 \\ |P_2| &= |P_3| = 3 \end{aligned}$$



(a) Multi Choice Pattern

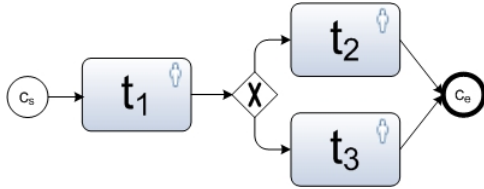


(b) Multi Choice Runtime Paths

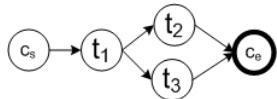
Figure 9: Multi Choice Pattern

#### Exclusive Choice Pattern:

The exclusive choice pattern or *XOR-Split* describes the divergence of a single branch into two or more branches. When the task  $t_1$  is completed, the thread of control is immediately passed to exactly one of the tasks  $t_2$  or  $t_3$  based on the outcome of a logical expression associated with the branch at runtime. The two possible paths  $P_1$  and  $P_2$  both contain an exclusive task such that it does not occur in the other path. This is illustrated in Figure 10(b) where  $P_1$  does not contain any occurrence of task  $t_3$  and  $P_2$  does not contain any occurrence of task  $t_2$ .



(a) Exclusive Choice Pattern



(b) Exclusive Runtime Paths

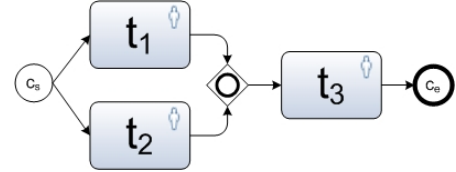
Figure 10: Exclusive Choice Pattern

$$\begin{aligned}
 V(G) &= V(P_1) \cup V(P_2) \\
 P_1 &= (\{c_s, t_1, t_2, c_e\}, \{c_s t_1, t_1 t_2, t_2 c_e\}) \\
 P_2 &= (\{c_s, t_1, t_3, c_e\}, \{c_s t_1, t_1 t_3, t_3 c_e\}) \\
 |P_1| &= |P_2| = 3
 \end{aligned}$$

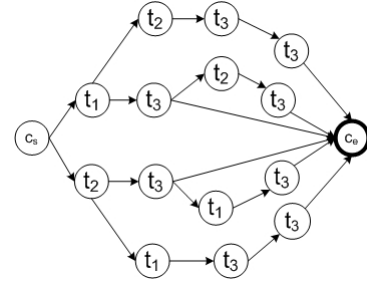
#### Simple Merge Pattern:

The simple merge or *OR-Join* describing the convergence of two or more branches into a single subsequent branch is one of the mostly widely discussed patterns in the domain of workflow management [10]. Essentially, each completion of an incoming branch results in the occurrence of a task in the

subsequent branch. This provides the means of merging two or more distinct branches without synchronising them. As a result, this affords the simplification of a process model by removing the need to explicitly replicate a sequence of activities that is common to two or more branches as illustrated by Figure 11(b). Instead, these branches can be joined with a simple merge construct and the common set of tasks need only to be depicted once in the process model. As shown in Figure 11(b) this leads to a total of 8 different paths. Note however, that path  $P_1, P_6$  have two occurrences of task  $t_3$  in a row. Unlike the other paths in this case it is not possible to judge if either task  $t_1$  or task  $t_2$  triggered the first execution of task  $t_3$ . This is however, decidable for the other paths at runtime.



(a) Simple Merge Pattern



(b) Simple Merge Runtime Paths

Figure 11: Simple Merge Pattern

$$\begin{aligned}
 V(G) &= V(P_1) \cup V(P_2) \cup V(P_3) \cup V(P_4) \cup V(P_5) \cup V(P_6) \\
 P_1 &= (\{c_s, t_1, t_2, t_3, t_3, c_e\}, \{c_s t_1, t_1 t_2, t_2 t_3, t_3 t_3, t_3 c_e\}) \\
 P_2 &= (\{c_s, t_1, t_2, t_3, t_3, c_e\}, \{c_s t_1, t_1 t_3, t_3 t_2, t_2 t_3, t_3 c_e\}) \\
 P_3 &= (\{c_s, t_1, t_3, c_e\}, \{c_s t_1, t_1 t_3, t_3 c_e\}) \\
 P_4 &= (\{c_s, t_2, t_3, c_e\}, \{c_s t_2, t_2 t_3, t_3 c_e\}) \\
 P_5 &= (\{c_s, t_1, t_2, t_3, t_3, c_e\}, \{c_s t_2, t_2 t_3, t_3 t_1, t_1 t_3, t_3 c_e\}) \\
 P_6 &= (\{c_s, t_1, t_2, t_3, t_3, c_e\}, \{c_s t_2, t_2 t_1, t_1 t_3, t_3 t_3, t_3 c_e\}) \\
 |P_3| &= |P_4| = 3 \\
 |P_j| &= 5; \forall P_j \text{ in } G \setminus \{P_3, P_4\}
 \end{aligned}$$

## 4. CONTROL-FLOW DEPENDENCIES

In the previous section we discussed control-flow semantics for basic patterns that are well known in the domain of workflow management systems. We listed all possible execution paths and discovered that some patterns even result in paths of different length. In addition, we presented a specification for entailment constraints for an arbitrary set of conflicting tasks  $T_k$  without considering their control-flow relation. In this section we define two sets of constrained tasks  $\{t_1, t_2\}$  and  $\{t_2, t_3\}$  and define a control-flow relationship between them based on the basic control-flow patterns.

## 4.1 Constrained Tasks Properties

In order to reveal a potential dependency between the set  $T_k$ , the applied control-flow patterns, and the latter entailment constraints we specify three properties for  $T_k$  that are based on the execution path evaluation of Section 3:

### 4.1.1 Strict Entailment

In case a constrained set of tasks  $T_k$  has exactly one partial order observable at runtime when applied to a workflow pattern we call  $T_k$  a *strict* set of constrained tasks. Thus, the tuple  $\langle T_k, < \rangle$  defines an antisymmetric order on  $T_k$ . In other words  $T_k$  is strict when applied to a control-flow pattern if:

$$\exists(t_i, t_j) \in T_k, \text{ with } t_i < t_j \forall P \text{ in } G$$

### 4.1.2 Loose Entailment

In case a set of constrained tasks  $T_k$  has more than one partial order observable at runtime when applied to a workflow pattern we call  $T_k$  a *loose* set of constrained tasks. Thus, the tuple  $\langle T_k, = \rangle$  defines a symmetric order on  $T_k$ . According to the pattern discussion in Section 3 a set  $T_k$  is *loose* if:

$$\begin{aligned} \exists(t_i, t_j) \in T_k, \text{ with } t_i < t_j \in P_k \text{ in } G \\ \exists(t_i, t_j) \in T_k, \text{ with } t_j < t_i \in P_l \text{ in } G \\ P_k \neq P_l \end{aligned}$$

### 4.1.3 Safe Entailment

The evaluation of the basic control-flow patterns in Section 3 revealed that some patterns imply execution paths of differing length. Different path lengths mean that not all tasks may occur at runtime, i.e. for a set of constrained tasks  $T_k$  there may be no partial order between two tasks  $t_i, t_j$ . We call a set of constrained tasks  $T_k$  *safe* if:

$$\exists(t_i, t_j) \in T_k, \text{ with } (t_j < t_i) \vee (t_i < t_j) \forall P \text{ in } G$$

## 4.2 Pattern Evaluation

The following Figure 12 depicts the properties of two constrained sets of tasks  $\{t_1, t_2\}$  and  $\{t_2, t_3\}$  when applied to the basic control-flow patterns discussed previously. An immediate observation is the mutual exclusivity of the properties *strict* and *loose*. Thus, a set of constrained tasks may be either *strict*, *loose*, or possess neither (*none*). The *safety* property eventually holds true for *strict* or *loose* sets, but not for sets that are none of both.

Based on the relationship between the constrained tasks set properties and the influence of the control-flow patterns execution semantics we argue that there are constraint specifications that are correctly applied to the workflow model, but could lead to ambiguous or non-occurring situations. In some cases this may even lead to a disruption of the whole workflow execution when it comes to constraint evaluation at runtime.

### 4.2.1 Dead Entailment

We define a dead constraint in case there exists no execution path that contains all of the tasks in  $T_k$ . The *XOR-Split* contains a set of constrained tasks  $\{t_2, t_3\}$ , whose temporal

Pattern	$T_k$	Strict	Loose	Safe
Sequence	$\{t_1, t_2\}$	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	$\{t_2, t_3\}$	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
AND-Split	$\{t_1, t_2\}$	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	$\{t_2, t_3\}$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
OR-Split	$\{t_1, t_2\}$	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	$\{t_2, t_3\}$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
XOR-Split	$\{t_1, t_2\}$	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	$\{t_2, t_3\}$	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OR-Join	$\{t_1, t_2\}$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	$\{t_2, t_3\}$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Figure 12: Pattern-Property Dependency

order is neither *strict* nor *loose*, given that there exists no path  $P$  in  $G$  with  $t \in V(P); \forall t \in T_k$ . Therefore, such a constraint will never be enforced at runtime for any potential execution path.

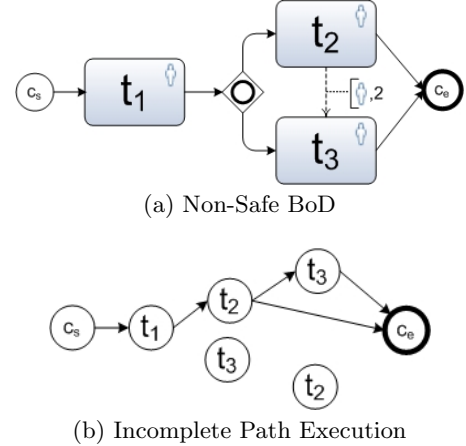


Figure 13: Non-Safe Binding of Duty

### 4.2.2 Blocking Entailment

According to Figure 12, *strict* entailment constraints for sequential tasks or *loose* entailment constraints on parallel tasks are considered *safe*. In case a *strict* entailment constraint is defined on two parallel tasks, a blocking situation may occur in case the evaluation of the current constrained tasks depends on a task that has not yet been executed. For example, considering Figure 13, we defined a *binding of duty* constraint on  $\{t_2, t_3\}$ . In case a constraint requires a strict partial order this may lead to a blocking situation at runtime [25].

### 4.2.3 Nested Entailment

In a business scenario, workflows are constructed out of various control-flow patterns. Therefore, we consider the workflow depicted in Figure 14. In this case an *AND-split* is combined with an *XOR-Split*. The *AND-Split* singularly is considered safe because all resulting paths are of equal length. As  $t_5$  is part of the nested *XOR-Split* control-flow pattern however, the constrained tasks  $\{t_5, t_6\}$  may lead to a deadlock situation where  $t_4$  is executed instead of  $t_5$ , and  $t_5$  must occur before  $t_6$ . This is related to the *non-safety* property implied by the *XOR-Split*. Therefore, a combination of a *safe* and a *non-safe* pattern results in a *non-safe* combination.

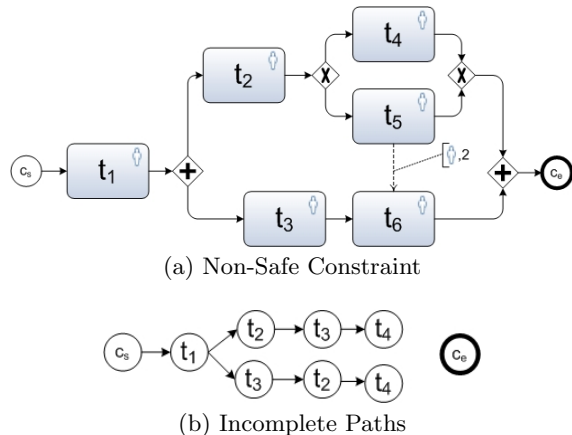


Figure 14: Constraint-Based Deadlock

### 4.2.4 Non-Decidable

Another issue is related to the simple merge pattern or *OR-Join* (cf. Figure 11(a)). In this scenario there exist two paths with two neighboured vertices that are related to the occurrence of task  $t_3$ . Consider a constraint defined on  $\{t_1, t_3\}$ . Given the highlighted paths in Figure 15 deciding which occurrence of  $t_3$  is based on the completion of  $t_1$  is non-decidable: either the first or the second could be executed because of the completion of task  $t_1$ .

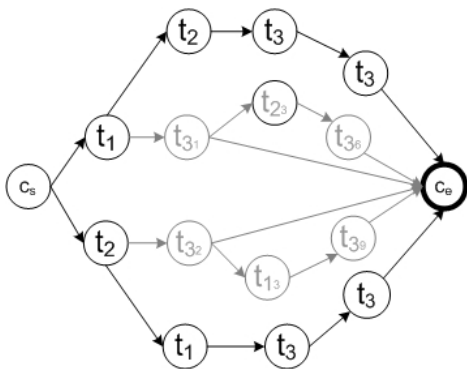


Figure 15: Ambiguous Paths

## 5. RELATED WORK

The definition of fine-grained authorisation constraints in the domain of workflow management systems received a lot of attention recently.

In [7] a language is presented for defining constraints on role assignment and user assignment to tasks in a workflow. This language is enriched with constraints supporting the expression of both static and dynamic *separation of duties*. While it is not explicitly stated that constrained tasks must be adjacent they only define constraints on exactly two tasks with a single observable temporal order at runtime. Their example workflows are based on sequences and *And-Splits*. Nevertheless, by only applying constraints to *strict* and *safe* sets of tasks with respect to the control-flow they are able to provide a constraint validation algorithm.

A complete algebra for complex *separation of duty* constraints is given in [26]. The algebra is used to specify policies that associate a task with a term in the algebra and require that all sets of users that perform the task satisfy the term. Complete syntax and semantics are given along with a study on their algebraic properties. Their algebra is limited to express constraints based on exactly two tasks and without supporting *binding of duty* constraints. In addition, they point out some computational problems related to their algebra.

SecureFlow [27] is a workflow management system that enforces authorisation constraints at runtime for users, roles, and workflow tasks. These authorisations are based on two partially ordered tasks. Their approach is based on a formal model, but their model does not consider any dependency on the control-flow of the constrained tasks, such as *XOR-Splits*.

Similar, in [8, 13] entailment and cardinality constraints are defined for two adjacent tasks. They also argue that the order of execution for two constrained tasks must not be fixed due to parallelism. In order to evaluate the consistency of defined entailment constraints they calculate a linear extension defining a task execution order that does not violate any constraints. Their proposed solution is applicable in case all potential execution paths implied by the control-flow are of equal length. They do not consider control-flow patterns that result in paths of arbitrary length, such as *OR-Splits* or *OR-Joins* order at runtime. This is a drawback regarding their proposed mapping of their constraints to the process execution languages BPEL in [17], due to the fact that BPEL supports all basic control-flow patterns, ambiguous situation as discussed in Section 4 may arise.

Wang and Li presented a role-relation-based access control model for workflow systems in [28] providing an extended set of authorisation constraints that also considers user relationship. They discussed the workflow resiliency problem and outlined an approach to determine a valid set of users that must execute tasks in order to successfully complete the constrained workflow. Their definition of a workflow is based on the assumption that there exists exactly one partial order among workflow tasks for a given workflow instance not considering various orders of task execution and execution path lengths depending on the control-flow semantics of more complex workflows patterns, such as *OR-Splits* or *OR-Joins*.

In [29], *separation of duty* constraints are defined in the context of Petri Nets. Knorr and Stromer discussed the potential different partial orders observable at runtime and

therefore defined constraints for a non-fixed execution order on the modelled control-flow of the Petri Net. Their approach supports the *Sequence* and *AND-Split*, but they do not investigate implications on other basic control-flow patterns.

Model-driven security and the automated generation of security enhanced software artefacts and security configurations has been a topic of interest in recent years. For instance, SecureUML [30] is a model-driven security approach for process-oriented systems focusing on access control. Similar to SecureUML, Jürjens presented the UMLSec extension for UML [31] in order to express security relevant information within a system specification diagram. One focus of UMLSec lies on the modelling of communication-based security goals, such as confidentiality, for software artefacts, while SecureUML describes desired state transitions and access control configurations for process-aware information systems, both do not address the specification of fine-grained entailment constraints in the context of workflow models.

Recently an approach was presented to describe authorisation concepts as part of UML-Activity diagrams and BPMN models [32]. Their approach follows the same idea of creating a common level of abstraction for both security expert and workflow expert, but their approach only supports role-based task assignments without any consideration of fine-grained authorisation concepts, such as *separation of duty*.

## 6. CONCLUSION

The analysis of basic control-flow patterns, and their impact on entailment constraints is based on the industrial need to better understand the specification of security aspects. This is typified by authorisation constraints on the abstract level of workflow models.

Recent work has discussed task-based entailment constraints and their specification in the context of example workflows, such as a classical approval scenario. To the best of our knowledge existing work on consistency and resilience properties of authorisation constrained workflows assume *safe* task constraints, implying only execution paths of equal length at runtime have been considered.

First proposals for security annotated process models have emerged [14, 9, 17, 32] that can be directly enforced by a process engine or an authorisation monitor at runtime [13]. We highlighted in this paper that depending on the control-flow semantics of the constrained tasks, situations may occur at workflow runtime that could lead to a disruption of the workflow execution based on the current definitions of task-based entailment constraints. Therefore, we evaluated the execution semantics of basic control-flow patterns that were supported by all major BPM standards, such as BPEL, BPMN, EPC, XPDL or UML [33].

We applied dynamic authorisation constraints as part of business process models and developed first modelling prototypes. This pragmatic approach revealed that even basic control-flow patterns lead to execution paths with varying length at runtime that may also affect the consistency and resilience properties of authorisation constrained workflows. Related work was focused on consistency analysis methods limited to *safe* constraints related to the *Sequence* and *And-Split* Patterns, but access control models and evaluation methods must be suitable for complex control-flow workflows and today's real world business scenarios in order to become accepted and supported by the industry. This has

been the case for workflow and business process management for decades. It is therefore our intention to initiate a discussion into the impact of control-flow semantics for task-based access control models and authorisation constraints in the context of process modelling notations.

## 6.1 Future Work

We are aware that this paper is pointing towards a possible direction of research in the domain of task-based access control models and workflow management systems where several topics remain unaddressed. In this work, we evaluated 5 out of over 40 different control-flow patterns. We have not yet investigated certain aspects, such as task iteration or task cancellation. Such patterns are of importance, especially when considering advanced authorisation concepts, such as delegation and revocation.

A control-flow perspective is only one aspect of workflow modelling, a resource perspective could be suitable in order to define organisation role and role-task assignments in the context of workflow models. An evaluation of data-flow patterns may reveal valuable input in order to specify Chinese Wall, compliance artefacts, or data-driven authorisation concepts as part of the workflow model.

A very important aspect is the evaluation or validation of authorisation constraints being part of the workflow model. We are considering applying evaluation techniques, such as binary decision trees or high-level Petri Net analysis that are well known in the domain of workflow management for some time, in order to detect potential deadlock situations, dead entailment constraints, and contradicting constraints. For bound Petri Nets such an evaluation could be used to provide immediate feedback at workflow design-time and to avoid potential workflow execution disruption based on inconsistent authorisation constraints at runtime.

## 7. REFERENCES

- [1] Dimitrios Georgakopoulos, Mark F. Hornick, and Amit P. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
- [2] Andreas Schaad, Volkmar Lotz, and Karsten Sohr. A model-checking Approach to Analysing Organisational Controls in a Loan Origination Process. In *SACMAT '06: Proceedings of the eleventh ACM Symposium on Access Control Models and Technologies*, pages 139–149, New York, NY, USA, 2006. ACM.
- [3] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. In *Proc. IEEE*, volume 63, pages 1278–1308. IEEE Computer Society, 1975.
- [4] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *Security and Privacy*, 00:184, 1987.
- [5] M. Nash and K. Poland. Some Conundrums Concerning Separation of Duty. In *In IEEE Symposium on Security and Privacy*, pages 201–209, Oakland, CA, 1990.
- [6] R. A. Botha and J. H. P. Eloff. Separation of Duties for Access Control Enforcement in Workflow Environments. *IBM System Journal*, 40(3):666–682, 2001.

- [7] Elisa Bertino, Elena Ferrari, and Vijay Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. In *ACM Transactions on Information and System Security*, volume 2, pages 65–104, 1999.
- [8] Kaijun Tan, Jason Crampton, and Carl A. Gunter. The Consistency of Task-Based Authorization Constraints in Workflow Systems. In *CSFW*, pages 155–, 2004.
- [9] Christian Wolter and Andreas Schaad. Modeling of Task-Based Authorization Constraints in BPMN. In *Proceedings of the 5th International Conference on Business Process Management (BPM)*, pages 64–79, 2007.
- [10] W. M. P. van der Aalst and A. H. M. ter Hofstede. Workflow Patterns: On the Expressive Power of Workflow Languages. In *Proc. of the 4th Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, August 28-30, 2002 / Kurt Jensen (Ed.)*, pages 1–20. Technical Report DAIMI PB-560, August 2002.
- [11] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. On the Suitability of BPMN for Business Process Modelling. In *Proceedings of the 4th International Conference on Business Process Management (BPM)*, 2006.
- [12] W. M. P. van der Aalst. Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype. In *BPTrends 3*, volume 5, pages 1–11, 2005.
- [13] Jason Crampton. A Reference Monitor for Workflow Systems with Constrained Task Execution. In *SACMAT '05: Proceedings of the tenth ACM Symposium on Access Control Models and Technologies*, pages 38–47, New York, NY, USA, 2005. ACM.
- [14] Shazia Wasim Sadiq, Guido Governatori, and Kioumars Namiri. Modeling Control Objectives for Business Process Compliance. In *BPM*, pages 149–164, 2007.
- [15] James H. Lambert, Rachel K. Jennings, and Nilesh N. Joshi. Integration of Risk Identification with Business Process Models. *Syst. Eng.*, 9(3):187–198, 2006.
- [16] Roshan K. Thomas and Ravi S. Sandhu. Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Authorization Management. In *IFIP Workshop on Database Security*, pages 166–181, 1997.
- [17] Elisa Bertino, Jason Crampton, and Federica Paci. Access Control and Authorization Constraints for WS-BPEL. In *ICWS*, pages 275–284. IEEE Computer Society, 2006.
- [18] Jacques Thomas, Federica Paci, Elisa Bertino, and Patrick Eugster. User Tasks and Access Control over Web Services. In *ICWS*, pages 60–69. IEEE Computer Society, 2007.
- [19] N. Nagaratnam, A. Nadalin, M. Hondo, M. McIntosh, and P. Austel. Business-driven application security: From Modeling to Managing Secure Applications. *IBM Systems Journal*, Vol 44, No 4, 2005.
- [20] Christian Wolter, Andreas Schaad, and Christoph Meinel. Deriving XACML Policies from Business Process Models. In *WISE Workshops*, pages 142–153, 2007.
- [21] Object Management Group. Business Process Modeling Notation Specification. [www.bpmn.org](http://www.bpmn.org), 2006.
- [22] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *In Proc. of 17th Int. Conf. on Advanced Information Systems Engineering (CAiSE05)*, 2005.
- [23] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.
- [24] B. Kiepuszewski, A. Hofstede, and W. van der Aalst. Fundamentals of Control Flow in Workflows, 2002.
- [25] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
- [26] Ninghui Li and Qihua Wang. Beyond Separation of Duty: An Algebra for Specifying high-level Security Policies. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 356–369, New York, NY, USA, 2006. ACM.
- [27] Wei-kuang Huang and Vijayalakshmi Atluri. SecureFlow: A Secure Web-Enabled Workflow Management System. In *ACM Workshop on Role-Based Access Control*, pages 83–94, 1999.
- [28] Qihua Wang and Ninghui Li. Satisfiability and Resiliency in Workflow Systems. In *ESORICS*, pages 90–105, 2007.
- [29] Konstantin Knorr and Henrik Stromer. Modeling and Analyzing Separation of Duties in Workflow Environments. In *Sec '01: Proceedings of the 16th international conference on Information security: Trusted information*, pages 199–212, 2001.
- [30] David Basin, Jürgen Doser, and Torsten Lodderstedt. Model Driven Security for Process-Oriented Systems. In *SACMAT '03: Proceedings of the eighth ACM Symposium on Access Control Models and Technologies*, pages 100–109, 2003.
- [31] Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 412–425, 2002.
- [32] Alfonso Rodríguez, Eduardo Fernández-Medina, and Mario Piattini. Towards a UML 2.0 Extension for the Modeling of Security Requirements in Business Processes. In *TrustBus*, pages 51–61, 2006.
- [33] Nick Russell, Arthur, Wil M. P. van der Aalst, and Natalya Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPMcenter.org, 2006.