Deriving Process-Related RBAC Models from Process Execution Histories

Anne Baumgrass, Sigrid Schefer-Wenzl, Mark Strembeck
Institute for Information Systems and New Media
Vienna University of Economics and Business (WU Vienna), Austria
Email: {firstname.lastname}@wu.ac.at

***Abstract*—In a business process context, access permissions grant the rights to perform certain tasks. In particular, process-related role-based access control (RBAC) models define RBAC policies for process-aware information systems (PAIS). In addition, process-related RBAC models allow for the definition of entailment constraints on tasks, such as mutual exclusion or binding constraints, for example. This paper presents an approach to derive process-related RBAC models from process execution histories recorded by a PAIS. In particular, we present algorithms to derive corresponding RBAC artifacts and entailment constraints from standardized XML-based log files. All algorithms presented in this paper have been implemented and were tested via process logs created with CPN Tools.**

***Keywords*-PAIS; RBAC; Constraints; Process Execution History**

I. Introduction

Process-aware information systems (PAIS) support the execution of tasks that are included in a business process [1]. The history of process executions in a PAIS can be recorded in special purpose log files often referred to as "event log", "audit trail", "history", or "transaction log". In the remainder of this paper, we refer to such a file as the *process execution history*.

In recent years, role-based access control (RBAC) has developed into a de facto standard for access control (see, e.g., [2]). In RBAC, permissions are assigned to roles, and roles are assigned to subjects (e.g. human users). A process-related RBAC model (see, e.g., [3], [4]) supports the definition of access control policies and entailment constraints on tasks. Entailment constraints, such as mutual exclusion and binding constraints, define which combination of subjects and roles is allowed to execute particular tasks (see, e.g., [3]–[6]). Mutual exclusion (ME) constraints result from the division of powerful privileges to avoid fraud and abuse. Thus, two (or more) mutually exclusive tasks must *not* be executed by the same subject or role. In contrast, binding constraints specify that the same subject or role has to perform two bound tasks.

From our experiences, gained in real-world projects and case studies, business processes as well as corresponding permissions or entailment constraints are often insufficiently documented – sometimes they are not documented at all. In this case, organizational mining can be applied to derive information about the subjects executing tasks in a PAIS (see, e.g., [7]–[9]). In this paper, we complement corresponding organizational mining approaches and present algorithms to derive process-related RBAC models from process execution histories (log files) of a PAIS. In particular, we derive *current-state RBAC models* that document the current state of this PAIS. These RBAC models contain roles, subjects, tasks/permissions, assignment relations, as well as binding and mutual exclusion constraints defined on tasks.

A. Approach Overview

In this paper, we focus on deriving candidate RBAC artifacts and assignment relations, as well as candidate mutual exclusion and binding constraints defined on tasks. The artifacts and artifact relations that are derived from a process execution history are *candidates* because they are subject to a subsequent refinement performed by a domain expert. One of the reasons for such a refinement is that the process execution history records all executions in a PAIS regardless of organizational changes (such as changes of the user's work profiles). For a tailored RBAC model, artifacts and artifact relations can be modified, added, or removed.

In particular, we extend the approach from [10] to derive candidate RBAC artifacts and assignment relations from a process execution history (see Section III-A). Moreover, we present algorithms to parse the process execution history of a PAIS to derive candidates for mutual exclusion and binding constraints (see Sections III-B – III-E). The candidate artifacts, assignment relations, and constraints will be combined to produce the current-state RBAC model.

The remainder of this paper is organized as follows. Section II provides the definitions for process-related RBAC models and process execution histories. In Section III, we present the derivation of current-state RBAC models in detail. Section IV discusses related work and Section V concludes the paper.[1]

II. Background

A. Role-Based Access Control Models

For the purposes of this paper, Definition 1 repeats some of the definitions for process-related RBAC models (for details see [3]).

[1]We provide an extended version of this paper on our Web page. In the extended version we re-inserted the text and examples that we had to cut from the paper due to the page restrictions for the proceedings version.

Definition 1 (Process-related RBAC Model)**.** *Let S be a set of subjects, R a set of roles, P_T a set of process types, P_I a set of process instances, T_T a set of task types, and T_I a set of task instances. A process-related RBAC Model $PRM = (E, Q, D)$ where $E = S \cup R \cup P_T \cup P_I \cup T_T \cup T_I$ refers to pairwise disjoint sets of the model, $Q = rsa \cup tra \cup pi \cup ti \cup es \cup er$ to mappings that establish relationships, and $D = sme \cup dme \cup sb \cup rb$ to mutual exclusion and binding constraints. For the partial mappings of the meta-model ($\mathcal{P}$ refers to the power set):*

1) *The mapping $rsa : S \mapsto \mathcal{P}(R)$ is called* role-to-subject assignment. *For $rsa(s) = R_s$ we call $s \in S$ subject and $R_s \subseteq R$ the set of roles assigned to this subject (the set of roles owned by s).*
2) *The mapping $tra : R \mapsto \mathcal{P}(T_T)$ is called* task-to-role assignment. *For $tra(r) = T_r$ we call $r \in R$ role and $T_r \subseteq T_T$ is called the set of tasks assigned to r.*
3) *The mapping $pi : P_T \mapsto \mathcal{P}(P_I)$ is called* process instantiation. *For $pi(p_T) = P_i$ we call $p_T \in P_T$ process type and $P_i \subseteq P_I$ the set of process instances instantiated from process type p_T.*
4) *The mapping $ti : (T_T \times P_I) \mapsto \mathcal{P}(T_I)$ is called* task instantiation. *For $ti(t_T, p_I) = T_i$ we call $T_i \subseteq T_I$ set of task instances, $t_T \in T_T$ is called task type and $p_I \in P_I$ is called process instance.*
5) *The mapping $es : T_I \mapsto S$ is called* executing-subject mapping. *For $es(t) = s$ we call $s \in S$ the executing-subject and $t \in T_I$ is called executed task instance.*
6) *The mapping $er : T_I \mapsto R$ is called* executing-role mapping. *For $er(t) = r$ we call $r \in R$ the executing-role and $t \in T_I$ is called executed task instance.*
7) *The mapping $sme : T_T \mapsto \mathcal{P}(T_T)$ is called* static mutual exclusion. *For $sme(t_1) = T_{sme}$ with $T_{sme} \subseteq T_T$ we call each pair $t_1 \in T_T$ and $t_x \in T_{sme}$ statically mutual exclusive tasks.*
8) *The mapping $dme : T_T \mapsto \mathcal{P}(T_T)$ is called* dynamic mutual exclusion. *For $dme(t_1) = T_{dme}$ with $T_{dme} \subseteq T_T$ we call each pair $t_1 \in T_T$ and $t_x \in T_{dme}$ dynamically mutual exclusive tasks.*
9) *The mapping $sb : T_T \mapsto \mathcal{P}(T_T)$ is called* subject-binding. *For $sb(t_1) = T_{sb}$ we call $t_1 \in T_T$ the subject binding task and $T_{sb} \subseteq T_T$ the set of subject-bound tasks.*
10) *The mapping $rb : T_T \mapsto \mathcal{P}(T_T)$ is called* role-binding. *For $rb(t_1) = T_{rb}$ we call $t_1 \in T_T$ the role binding task and $T_{rb} \subseteq T_T$ the set of role-bound tasks.*

Tasks can be defined as statically mutual exclusive (on the process type level) or dynamically mutual exclusive (on the process instance level). A static mutual exclusion (SME) constraint defines that two SME tasks must never be assigned to the same role. In contrast, two dynamically mutual exclusive (DME) tasks can be assigned to the same role, but must be performed by two different subjects within the same process instance (see, e.g., [3]).

Bound tasks can be subdivided into subject-bound and role-bound tasks. A subject-binding (SB) constraint defines that two bound tasks must be performed by the same subject. In turn, a role-binding (RB) constraint defines that bound tasks must be performed by members of the same role, but not necessarily by the same subject.

B. Process Execution History

The process execution history contains information about the tasks performed in a PAIS. We refer to an entry in the process execution history as *process history entry* (see Figure 1). Each entry includes a task instance and the subject executing the task instance. Definition 2 resembles the definition from [8] and specifies the essential elements of a process execution history.

Definition 2 (Process Execution History)**.** *Let T_I be a set of task instances, P_I a set of process instances, and S a set of subjects performing task instances (see Definition 1). An element of $E = T_I \times S$ is called process history entry. E denotes the set of process history entries and $p_i \in P_I$ denotes a process instance for which E_{P_I} is the set of possible sequences of entries describing the particular process instance p_i.*

A process execution history $P_H \in \mathcal{B}(p_i)$ is a multi-set of all possible process instances, such that:

1) *For $e_x = (t_i, s_j)$, the subject $s_j \in S$ performed the task instance t_i in process history entry $e_x \in E$. This definition corresponds to the executing-subject mapping $es(t_i) = s_j$ of Definition 1.5 where s_j is the executing-subject of task instance t_i.*
2) *For $p_i = (e_0, e_1, ..., e_k)$, $p_i \in P_H$ is called process instance and $e_0, ..., e_k \in E$ are the process history entries belonging to the process instance $p_i \in P_I$. In correspondence to Definition 1.4, the task instance in a process history entry is the instantiation of a specific task type.*

Figure 1 shows an example of a *process execution history*. A process execution history contains several process instances. Each process instance consists of several process history entries which describe the sequence of task instances executed in the corresponding process instance. Figure 1 also shows the *task execution history* (T_H) which records all task instances, process instances, and the corresponding executing-subjects for a specific task type. Definition 3 specifies task execution histories (cf. [8]).

Definition 3 (Task Execution History)**.** *Let T_T be a set of task types, T_I a set of task instances, S a set of subjects, and P_I a set of process instances (see also Definition 1). The* task execution history *$T_H \in (T_I \times S \times P_I)$ refers to a record of the set of task instances T_I, the set of executing-subjects S, and the set of process instances P_I such that:*

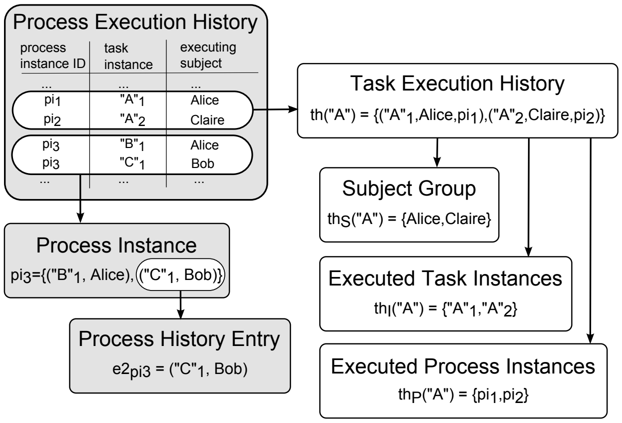

Figure 1. Example of a process execution history

1) *The* task execution history *of a particular task type is defined as mapping* $th : T_T \mapsto \mathcal{P}(\{(t_x, s_x, p_x)|t_x \in T_I, s_x \in S, p_x \in P_I\})$. *The task execution history of a task type returns a record of triples, each consisting of a task instance, a subject, and a process instance. For each task type* $t \in T_T$, *the* task execution history *is defined as* $th(t) = \{(t_x, s_x, p_x)|p_x \in P_I, t_x \in T_I, s_x \in S : t_x \in ti(t, p_i) \wedge es(t_x) = s_x\}$.
2) *The task execution history implies a mapping* subject group $th_S : T_T \mapsto \mathcal{P}(S)$ *that returns all executing-subjects of a task type. The mapping* executed task instances $th_I : T_T \mapsto \mathcal{P}(T_I)$ *returns all task instances of a task type. Likewise, the mapping* executed process instances $th_P : T_T \mapsto \mathcal{P}(P_I)$ *returns all process instances in which a instance of the task type was executed.*

III. Deriving Process-Related RBAC Models

We discuss our approach using a process execution history created with CPN Tools (see [11]). For demonstration purposes, we use a simplified example of a credit application process (for details see [3]) for which we generated corresponding XML-based log files.

For the sake of simplicity, the examples of process execution histories shown below include only two process instances respectively. However, to derive useful RBAC artifacts and constraints, we tested our algorithms with histories that include a much larger number of process instances, of course.

A. Deriving Candidate Artifacts and Assignment Relations

With respect to [10], we can derive candidate RBAC artifacts from process execution history files in MXML or XES format. Figure 2 shows the main relations between elements of a process execution history and corresponding RBAC artifacts. Based on Definitions 1–3, the following *derivation rules* are applied (see also Figure 2 and [10]):

- The executing-subject stored in a process execution history serves as candidate subject for the current-state RBAC model.
- The task types stored in a process execution history identify candidate tasks/permissions for the current-state RBAC model.
- The subject group of a task type is used to derive a candidate role. This is because, a subject group defines all subjects performing a certain task type (see Definition 3).
- A candidate role-to-subject assignment (rsa) for the current-state RBAC model is derived for each executing-subject in a subject group.
- A task-to-role assignment (tra) relation is derived via the relation between a task type and its subject group.

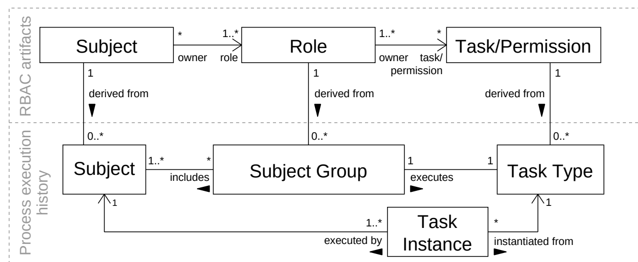

Figure 2. Process execution history elements and RBAC artifacts

To reduce the number of candidate roles, as well as corresponding task-to-role and role-to-subject assignment relations, we can apply organizational mining techniques (see, e.g., [7]–[9]).

In XES, a subject's position in the organization can also be stored in the process execution history. In particular, the XES standard defines the precise semantics of its attributes via extensions. The "Organizational" extension provides the "org:role" attribute which identifies a subject's position in the organization (for details see [12]). Thus, the "org:role" attribute in XES can also be used to derive candidate roles for the current-state RBAC model.

B. Deriving Candidate SME Constraints

We assume that a SME constraint on two tasks can be derived from a process execution history if both tasks are never performed by the same subject. However, this assumption can result in an RBAC model that includes a potentially large number of false positives. In other words, two task types that are included in two different process types are not necessarily SME tasks, even if they are never executed by the same subject. To reduce the number of false positives, we therefore derive candidate SME constraints for each process type separately. Procedure 1 compiles the set of all executing-subjects of a task type for a particular process type.

Procedure 1. *Compile the subject group of a task type for a particular process type.*

Name: $allExecutingSubjects$
Input: $task_x \in T_T, process_t \in P_T$

create $allsubjects$ ▷ empty set that disallows duplicates
for each $p \in pi(process_t)$ **do**
  for each $t \in ti(task_x, p)$ **do**
    add $es(t)$ **to** $allsubjects$
  end for
end for
return $allsubjects$

A candidate SME constraint defined on two tasks $task_x, task_y \in T_T$ for a process type $p_t \in P_T$ is derived if $allExecutingSubjects(task_x, p_t) \cap allExecutingSubjects(task_y, p_t) = \emptyset$ applies (see Algorithm 1).

Algorithm 1 Derive candidate SME constraints

Input: $T_T, p_t \in P_T$
for each $task_x \in T_T$ **do**
  for each $task_y \in T_T \mid task_y \neq task_x$ **do**
    if $allExecutingSubjects(task_x, p_t) \cap$
        $allExecutingSubjects(task_y, p_t) = \emptyset$ **then**
      set $task_x \in sme(task_y)$ ▷ marked as candidate
    end if
  end for
end for

Figure 3 shows a process execution history excerpt in MXML standard format[2]. The two task instances "Check credit worthiness" and "Approve contract" are executed by different subjects (shown via arrow 1). In this excerpt, "Alice" and "Susan" perform the first task ($th_S($`Check for credit worthiness`$) = \{Alice, Susan\}$) and "Bob" performs the second task ($th_S($`Approve contract`$) = \{Bob\}$) (shown via arrow 2). Using the process execution history shown in Figure 3, we can derive a candidate SME constraint defined on these two tasks because these tasks are not performed by the same subject.

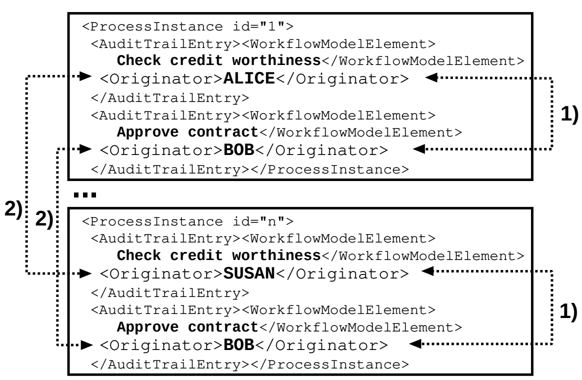

Figure 3. Example detection of SME tasks

[2]The detection of constraints from XES is conducted in the same way.

C. Deriving Candidate DME Constraints

We derive candidate DME constraints if two tasks are never executed by the same subject in the *same* process instance. In contrast to SME constraints, DME tasks can be executed by the same subjects within *different* process instances (see also Definition 1.8 and [3]). Therefore, a candidate DME constraint defined on two task types $task_x, task_y \in T_T$ is derived if for each process instance $p_i \in P_I$ and for all corresponding task instances $t_x, t_y \in T_I : t_x \in ti(task_x, p_i) \wedge t_y \in ti(task_y, p_i) \wedge es(t_x) \neq es(t_y)$ applies (see Algorithm 2).

Algorithm 2 Derive candidate DME constraints

Input: T_T, P_I
for each $task_x \in T_T$ **do**
  for each $task_y \in T_T \mid task_y \neq task_x$ **do**
    for each $p_i \in P_I | \forall t_x, t_y \in T_I :$
        $t_x \in ti(task_x, p_i) \wedge t_y \in ti(task_y, p_i) \wedge$
        $es(t_x) \neq es(t_y)$ **do**
      set $task_x \in dme(task_y)$ ▷ marked as candidate
    end for
  end for
end for

Again, Figure 4 depicts a process execution history excerpt with two process instances. In Figure 4, the two task instances "Negotiate contract" and "Approve contract" were both performed by the same subjects "Alice" and "Bob" (shown via arrow 1). However, in each process instance both tasks were performed by a different subject (shown via arrow 2). Therefore, we derive a candidate DME constraint for the tasks from Figure 4.

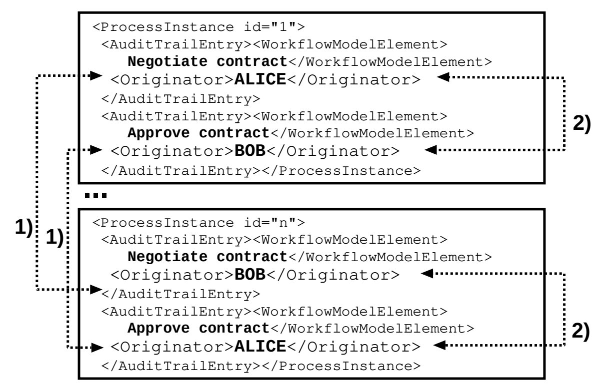

Figure 4. Example detection of DME tasks

D. Deriving Candidate Subject-Binding (SB) Constraints

A SB constraint defines that two (or more) subject-bound tasks must be performed by the same subject (see also Definition 1.9 and [3]). Two tasks $task_x, task_y \in T_T$ are candidate SB tasks if for each process instance $p_i \in P_I$ and for all corresponding task instances $t_x, t_y \in T_I : t_x \in$

$ti(task_x, p_i) \wedge t_y \in ti(task_y, p_i) \wedge es(t_x) = es(t_y)$ applies (see Algorithm 3).

Algorithm 3 Derive candidate SB constraints

Input: T_T, P_I
for each $task_x \in T_T$ **do**
  for each $task_y \in T_T \mid task_y \neq task_x$ **do**
    for each $p_i \in P_I | \forall t_x, t_y \in T_I$:
      $t_x \in ti(task_x, p_i) \wedge t_y \in ti(task_y, p_i) \wedge$
      $es(t_x) = es(t_y)$ **do**
      set $task_x \in sb(task_y)$ ▷ marked as candidate
    end for
  end for
end for

Figure 5 depicts a process execution history excerpt with two process instances that include the tasks “Check credit worthiness” and “Negotiate contract”. This example shows that both tasks are always performed by the same subject within the same process instance (shown via arrows). Therefore, we derive a candidate SB constraint from this process execution history.

```
<ProcessInstance id="1">
 <AuditTrailEntry><WorkflowModelElement>
    Check credit worthiness</WorkflowModelElement>
  <Originator>CLAIRE</Originator>
 </AuditTrailEntry>
 <AuditTrailEntry><WorkflowModelElement>
    Negotiate contract</WorkflowModelElement>
  <Originator>CLAIRE</Originator>
 </AuditTrailEntry></ProcessInstance>
...
<ProcessInstance id="n">
 <AuditTrailEntry><WorkflowModelElement>
    Check credit worthiness</WorkflowModelElement>
  <Originator>ALICE</Originator>
 </AuditTrailEntry>
 <AuditTrailEntry><WorkflowModelElement>
    Negotiate contract</WorkflowModelElement>
  <Originator>ALICE</Originator>
 </AuditTrailEntry></ProcessInstance>
```

Figure 5. Example detection of SB tasks

E. Deriving Candidate Role-Binding (RB) Constraints

A RB constraint defines that two (or more) role-bound tasks must always be performed by members of the same role (see also Definition 1.10 and [3]). However, role-bound tasks are not necessarily executed by the same subject. To derive candidate RB constraints, the process execution history must contain role information, i.e. information on which role a subject has activated to execute the task. Two tasks $task_x, task_y \in T_T$ are candidate RB tasks if in each process instance $p_i \in P_I$ and for all corresponding task instances $t_x, t_y \in T_I : t_x \in ti(task_x, p_i) \wedge t_y \in ti(task_y, p_i) \wedge er(t_x) = er(t_y)$ applies (see Algorithm 4).

Typically, MXML and XES files do not contain the executing-role of a task instance. Therefore, we cannot derive candidate role-bound tasks directly. However, we can derive role information from the “Organizational” extension provided in the XES standard for process execution histories

Algorithm 4 Derive candidate RB constraints

Input: T_T, P_I
for each $task_x \in T_T$ **do**
  for each $task_y \in T_T \mid task_y \neq task_x$ **do**
    for each $p_i \in P_I | \forall t_x, t_y \in T_I$:
      $t_x \in ti(task_x, p_i) \wedge t_y \in ti(task_y, p_i) \wedge$
      $er(t_x) = er(t_y)$ **do**
      set $task_x \in rb(task_y)$ ▷ marked as candidate
    end for
  end for
end for

(see also Section III-A). Thus, we can only derive candidate RB constraints if this organizational extension is used in a process execution history.

```
<trace><string key="concept:name" value="1"/>
 <event><string key="concept:name"
    value="Check credit worthiness"/>
 <string key="org:role" value="Clerk"/>
 </event>
 <event><string key="concept:name"
    value="Reject application"/>
 <string key="org:role" value="Clerk"/>
 </event></trace>
...
<trace><string key="concept:name" value="2"/>
 <event><string key="concept:name"
    value="Check credit worthiness"/>
 <string key="org:role" value="Manager"/>
 </event>
 <event><string key="concept:name"
    value="Reject application"/>
 <string key="org:role" value="Manager"/>
 </event></trace>
```

Figure 6. Example detection of RB tasks

In Figure 6, the same role performed the tasks “Check credit worthiness” and “Reject application” in each process instance (shown via arrows). In the first process instance both tasks are executed by subjects owning the role “Clerk”. In the second process instance, these tasks are executed by subjects owning the role “Manager”. Thus, for each process instance subjects owning the same role execute the two tasks. Therefore, we derive a candidate RB constraint from the process execution history.

IV. Related Work

Our work complements work from the area of process mining, especially organizational mining. Organizational mining aims to discover information about the relationship between subjects and executed tasks in a PAIS (see, e.g., [7], [8]). Song and van der Aalst [8] introduce an approach to mine organizational models and social structures from process execution histories. For example, they use mining techniques to cluster subjects into groups based on their execution of similar tasks. Jin et al. [9] introduce an algorithm to produce organizational models from process execution histories. Similar to [8], Jin et al. use a similarity measure to cluster subjects of a process execution history into roles if they (partially) completed the same tasks in a similar frequency. Moreover, Rembert and Ellis [13] provide a formal definition to mine different aspects from process execution histories. One aspect the authors present is the role assignment perspective which captures the relationship between roles and tasks. They introduce an algorithm to calculate this relationship by clustering subjects using a similarity metric.

Moreover, our approach is related to role mining. In role mining, data mining techniques are used to detect patterns in a set of access permissions [14]. These patterns are used to derive candidate RBAC policy sets. Furthermore, several approaches exist that include preexisting business information (e.g. job or department descriptions) to create RBAC policy sets via role mining (see [14]–[17]).

In [10], we presented a preliminary approach to derive candidate RBAC artifacts and assignment relations from a process execution history. This paper extends the approach of [10] and provides algorithms to derive entailment constraints, such as mutual exclusion and binding constraints, from a process execution history. Thereby, our work complements the approaches mentioned above by considering process-related information to automatically derive candidate RBAC models that include task-based entailment constraints.

V. Conclusion

PAIS record process execution histories that contain information about the subjects performing the tasks included in a business process. In this paper, we presented a systematic approach for the automated derivation of candidate RBAC models from process execution histories. Such RBAC models specify roles, subjects, tasks/permissions, assignment relations, as well as mutual exclusion and binding constraints defined on tasks. Therefore, RBAC models derived from process execution histories can serve as a basis for understanding how an organization actually enforces access control policies in PAIS.

In our future work, we will examine how other types of candidate constraints (such as context constraints) can be derived from process execution histories.

References

[1] M. Dumas, W. van der Aalst, and A. ter Hofstede, *Process-Aware Information Systems*. John Wiley & Sons, Inc., 2005.

[2] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-Based Access Control, Second Edition*. Artech House, 2007.

[3] M. Strembeck and J. Mendling, "Modeling process-related RBAC models with extended UML activity models," *Information and Software Technology*, vol. 53, no. 5, 2011.

[4] J. Warner and V. Atluri, "Inter-Instance Authorization Constraints for Secure Workflow Management," in *Proc. of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2006.

[5] M. Strembeck and J. Mendling, "Generic Algorithms for Consistency Checking of Mutual-Exclusion and Binding Constraints in a Business Process Context," in *Proc. of the 18th International Conference on Cooperative Information Systems (CoopIS), Lecture Notes in Computer Science (LNCS), Vol. 6426, Springer*, October 2010.

[6] N. Russell, W. van der Aalst, A. 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), Lecture Notes in Computer Science (LNCS), Vol. 3520, Springer*, 2005.

[7] W. van der Aalst, H. A. Reijers, and M. Song, "Discovering Social Networks from Event Logs," *Computer Supported Cooperative Work (CSCW)*, vol. 14, no. 6, December 2005.

[8] M. Song and W. van der Aalst, "Towards comprehensive support for organizational mining," *Decision Support Systems*, vol. 46, no. 1, 2008.

[9] T. Jin, J. Wang, and L. Wen, "Organizational Modeling from Event logs," in *Proc. of the 6th International Conference on Grid and Cooperative Computing (GCC)*. IEEE Computer Society, 2007.

[10] A. Baumgrass, "Deriving Current-State RBAC Models from Event Logs," in *International Workshop on Security Aspects of Process-aware Information Systems (SAPAIS), Proc. of the 6th International Conference on Availability, Reliability and Security (ARES)*. IEEE Computer Society, 2011.

[11] A. de Medeiros and C. W. Günther, "Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms," in *Proc. of the 6th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, 2005.

[12] C. W. Günther, "XES Standard Definition," 1.0, Draft. available at: http://www.xes-standard.org/, November 2009.

[13] A. J. Rembert and C. S. Ellis, "An Initial Approach to Mining Multiple Perspectives of a Business Process," in *Proc. of the 5th Richard Tapia Celebration of Diversity in Computing Conference (TAPIA)*. ACM, 2009.

[14] L. Fuchs and S. Meier, "The Role Mining Process Model," in *Proc. of the 6th International Conference on Availability, Reliability and Security (ARES)*. IEEE Computer Security, 2011.

[15] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo, "Mining Roles with Semantic Meanings," in *Proc. of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 2008.

[16] A. Colantonio, R. Di Pietro, A. Ocello, and N. V. Verde, "A Formal Framework to Elicit Roles with Business Meaning in RBAC Systems," in *Proc. of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2009.

[17] C. Giblin, M. Graf, G. Karjoth, A. Wespi, I. Molloy, J. Lobo, and S. Calo, "Towards an integrated approach to role engineering," in *Proc. of the 3rd ACM workshop on Assurable and usable security configuration (SafeConfig)*. ACM, 2010.