Conformance Checking of RBAC Policies in Process-Aware Information Systems

Anne Baumgrass[1], Thomas Baier[2], Jan Mendling[2], and Mark Strembeck[1]

[1] Institute of Information Systems and New Media
Vienna University of Economics and Business (WU Vienna), Vienna, Austria
{firstname.lastname}@wu.ac.at
[2] Institute of Information Systems
Humboldt-Universität zu Berlin, Germany
{firstname.lastname}@wiwi.hu-berlin.de

Abstract. A process-aware information system (PAIS) is a software system that supports the definition, execution, and analysis of business processes. The execution of process instances is typically recorded in so called event logs. In this paper, we present an approach to automatically generate LTL (Linear Temporal Logic) statements from process-related RBAC (Role-based Access Control) models. These LTL statements are used to check if process executions that are recorded via event logs conform to the access control policies defined via a corresponding RBAC model. To demonstrate our approach, we implemented a RBAC-to-LTL component, and used the ProM tool to test the resulting LTL statements with event logs created from process simulations in CPN tools.

Keywords: Process-Aware Information Systems, Conformance Checking, LTL, Security, Role-Based Access Control

1 Introduction

Process-aware information systems (PAIS) support the execution of business processes [6]. In this context, access control policies define which users are allowed to perform certain tasks (see, e.g., [14,16]). In recent years, role-based access control (RBAC) [8,13] has developed into a de facto standard for access control in both, research and industry. In RBAC, roles model different work profiles. They are equipped with the exact number of permissions that is needed to perform their tasks. These roles are then assigned to human users according to their respective work profile (see [15]). To check if the process instances that are executed via a PAIS adhere to the access control policies which are defined in the corresponding RBAC model, one can use the event logs that have been recorded by the information system during process execution (see, e.g., [7]). The analysis of event data has been intensively studied in the area of process mining [1,19]. Often, Linear Temporal Logic (LTL) is used as formal language to check conformance of models and logs (see, e.g., [2]). However, most existing conformance checking approaches focuses on the control flow perspective (see, e.g., [11,12]), and do not provide operators to check access control policies.

In this paper, we present an approach to check if the data recorded in the event logs of a PAIS conforms to the corresponding process-related RBAC model including binding and mutual exclusion constraints. In particular, we automatically transform process-related RBAC models to corresponding LTL rules. These LTL rules are then used to check the event logs for violations of the policies that are defined via the RBAC model. The results of this conformance check can serve as basis for security and domain experts to detect violations that could result from misconfigurations or implementation errors and thereby help to increase the security of the respective PAIS. The LTL rules that are generated by our approach can be used in any kind of log analysis tool that is based on LTL. To demonstrate our approach we implemented a RBAC-to-LTL component that transforms the XML representation of process-related RBAC models to corresponding LTL rules. Subsequently, we use tools such as ProM [18,21] to check these LTL rules.

The remainder of this paper is structured as follows. Section 2 gives an overview of RBAC and LTL. Next, Section 3 describes our approach of transforming process-related RBAC models to LTL. In Section 4, we demonstrate our approach using event logs created from process simulations in CPN tools. After discussing related work in Section 5, Section 6 concludes the paper.

2 Background

2.1 Process-related RBAC Models

In order to transform RBAC policies to LTL rules, we need a corresponding metamodel which defines the semantics of process-related RBAC models. In our approach, we use the formal metamodel for process-related RBAC models defined in [16]. However, due to the page restrictions, we cannot repeat the corresponding definitions in this paper. Therefore, we use the BusinessActivities UML extension defined in [16] to introduce the corresponding concepts via a small example model.

The BusinessActivities extension enables the definition of process-related RBAC models via extended UML activity models. In addition to roles and role-hierarchies, it allows for the specification of static and dynamic mutual exclusion constraints, as well as binding constraints on the tasks of a business process. Figure 1a depicts an example of a BusinessActivity that models a simple credit application process. This process includes five actions, three of which are so-called Business Actions ("Negotiate contract", "Approve contract", and "Check credit worthiness") which include binding or mutual exclusion constraints.

Figure 1b shows the roles for the credit application process. In this example, we have a role BankManager and a corresponding junior-role BankClerk. The role-to-role assignment relation is modeled via a dashed arrow. The arrowhead is a triangle including a "J" to indicate the end of the relation that points to the junior-role. Such a role-hierarchy is defined as a mapping rh. The mapping rh^* defines the inheritance in the role-hierarchy. It includes all direct and transitive junior-roles that the senior-role inherits from (for details see [16]).

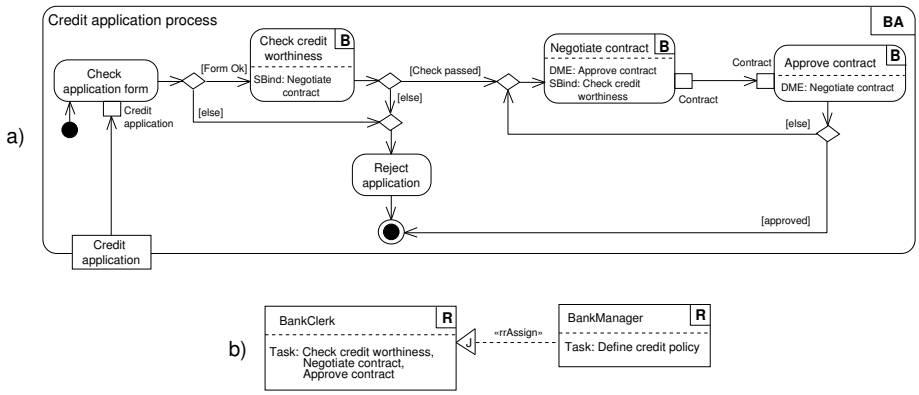

Fig. 1. A credit application process modeled as a BusinessActivity [16].

The task-to-role assignment *tra* defines which task types are assigned to a particular role. In a role-hierarchy the senior-role inherits the permissions from its (direct and transitive) junior-roles. Thus, in our example the BankManager inherits the permission to execute the tasks "Check credit worthiness", "Negotiate contract", and the "Approve contract" from the BankClerk role (see Figure 1b). The task-to-role assignment implies a mapping task ownership (*town*) which allows to determine all tasks that are assigned to a particular role. In turn, the mapping $town^{-1}$ returns all roles a task is assigned to. The role-to-subject assignment *rsa* defines which roles are assigned to a particular subject. Similar to the task-to-role assignment relation, *rsa* implies a mapping role-ownership *rown*, which allows to determine all roles that are assigned to a particular subject. Again, a mapping $rown^{-1}$ exists which returns all subjects assigned to a role (for details see [16]).

Mutual exclusive tasks result from the division of powerful rights or responsibilities to prevent fraud and abuse. Mutual exclusion constraints can be subdivided in static and dynamic mutual exclusion. In essence, a static mutual exclusion (SME) constraint defines that two mutual exclusive tasks must never be assigned to the same subject. In turn, a dynamic mutual exclusion constraint defines that two mutual exclusive tasks must never be performed by the same subject in the *same process instance*. Figure 1a depicts a DME constraint between the Business Actions "Negotiate contract" and "Approve contract". In the graphical representation, this DME constraint is indicated via the "DME" prefix in the corresponding BusinessAction elements. In contrast to mutual exclusion constraints, binding constraints define that bound tasks must be executed by the same subject (or role). In particular, a subject-binding (SB) constraint defines that two bound tasks must be performed by the same individual. 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 individual. The example from Figure 1a shows a subject binding constraint between the "Check

credit worthiness" and the "Negotiate contract" tasks. This subject-binding constraint is indicated via the "SBind" prefix in the corresponding BusinessAction elements (for details see [16]).

2.2 Checking Process Conformance with Linear Temporal Logic

In the area of process mining, Linear Temporal Logic (LTL) is used as a language to check the conformance of process models with executed business processes. For instance, van der Aalst et al. propose an approach to verify certain properties in event logs using LTL [2]. LTL is a modal temporal logic developed by Pnueli which introduces modalities referring to time that can be used to verify different properties in a linear path [10]. The language includes the basic logical operators ($\wedge, \vee, \Rightarrow, \Leftrightarrow, \exists, \forall$) and additionally the following operators to express time-related properties:

Nexttime ($\bigcirc A$) specifies that a property A holds in the next state of the path.
Eventually ($\Diamond A$) specifies that a property A evaluates to true at least at one point in the path.
Always ($\Box A$) specifies that a property A has to hold in every state of the path.
Until ($A \cup B$) specifies that a property A has to hold until property B holds.

Van der Aalst et al. [2] extended the language to exploit the structure of event logs. In particular, they introduce operands to access the different properties contained in an event log, such as attributes of the process instance and the audit trail entries. Furthermore, they provide tool support by introducing the LTL Checker in the ProM framework [4]. The LTL Checker provides a set of predefined formulas that can be used out of the box and can be easily extended. In our approach, we rely on LTL since it has proven to be a valuable means to check conformance of event logs. Further, LTL gives us the flexibility to extend our approach to enable the checking of a process's control flow with respect to the corresponding process-related RBAC model.

3 Transformation of RBAC Models to LTL Statements

In this section, we present our approach to check if business process executions comply with a corresponding process-related RBAC model. Figure 2 depicts the main concepts of our approach and their interrelations.

At first, we transform a particular process-related RBAC model that is modeled via the BusinessActivities extension (see Section 2.1) to corresponding LTL statements (see Section 2.2). Below, we describe this automated transformation in detail. The resulting LTL statements then represent the properties of the RBAC model that need to hold for each process execution. Subsequently, we use the automatically derived LTL rules to assess event logs using the LTL checker plug-in of the process mining workbench ProM (see also [2]). In this way, we check if a process instance conforms to the RBAC model and reveal violations

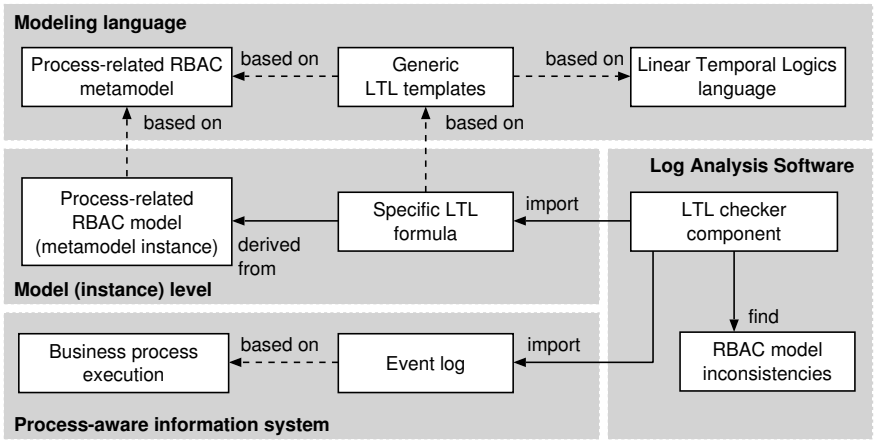

Fig. 2. Conformance checking for process-related RBAC models

in case a particular process execution (resp. the corresponding event log) is not consistent with the respective RBAC model.

In order to support the automated transformation of process-related RBAC models (modeled via the BusinessActivities extension), we developed a corresponding RBAC-to-LTL component. Figure 3 shows the conceptual structure of this component. In particular, we first generate the XML representation of a particular process-related RBAC model. Subsequently, we parse this XML representation to create a corresponding in-memory object model. This in-memory object model is then used to derive specific LTL rules. To generate the LTL rules we use special purpose LTL templates. In essence, these LTL templates define patterns for the different properties of a RBAC model, including static and dynamic mutual exclusion, subject-binding and role-binding, as well as task-to-role assignment relations (see also Section 2.1).

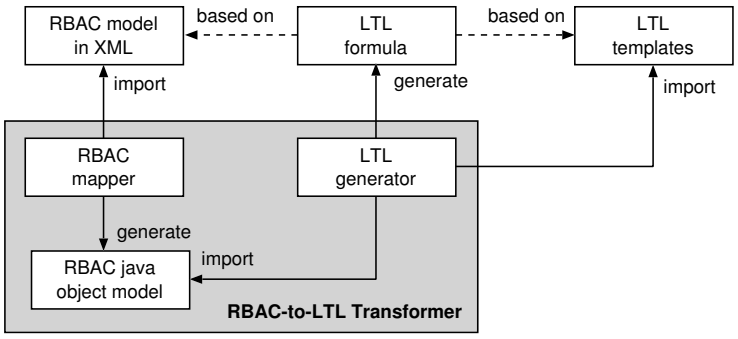

Fig. 3. Transformation of RBAC models in XML representation to LTL formulas

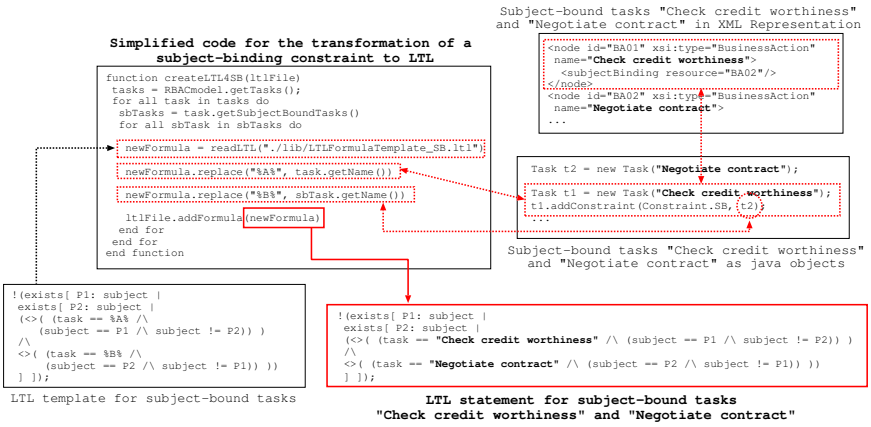

Fig. 4. Example transformation of a subject-binding in XML representation to LTL

Figure 4 shows an actual example for the transformation of a subject-binding constraint to a corresponding LTL statement. In particular, the upper right corner shows an excerpt from the XML representation of the respective RBAC model. The source code excerpt beneath shows the creation of corresponding Java objects from XML representation. The subsequent generation of the corresponding LTL statement via a respective LTL template is shown in the source code excerpt on the left-hand side. The LTL template for checking subject-binding constraints between two tasks is also shown in Listing 1.1. In particular, we check that in a certain process instance two subject-bound tasks are executed by the same individual. In LTL we achieve this by testing if no two persons P1 and P2 exist who executed task A and task B respectively (see Listing 1.1).

```
1 !(exists[ P1: subject |
2    exists[ P2: subject |
3     (<>( (task == %A% /\ (subject == P1 /\ subject != P2)) )
4      /\
5      <>( (task == %B% /\ (subject == P2 /\ subject != P1)) ))
6    ]
7 ]);
```

Listing 1.1. LTL formula to check subject-binding for two tasks

Listing 1.2 shows the LTL template for checking dynamic mutual exclusion constraints. The DME constraint defined in LTL checks that no person exists who executes two DME tasks A and B. While DME is checked within single process instances, SME requires two tasks to be mutual exclusive over all process instances (PI) (see [16]). However, using the LTL checker we can only assess constraints within individual process instances. In order to check SME with the LTL formula shown in Listing 1.2, we can, for example, combine all process instances of a certain event log into a single PI.

```
!(exists[ P: subject |
 (<>( (subject == P /\ task == %A%) )
  /\
  <>( (subject == P /\ task == %B%) ))
]);
```

Listing 1.2. LTL formula to check mutual exclusion for two tasks

```
(
<>((task == %A% /\
   (subject == P1 \/ (... \/ (subject == P_N-1 \/ subject == P_N ...)))))
\/ !( <>( task == %A% ) )
);
```

Listing 1.3. LTL formula to check task-to-role assignment (excerpt)

From the perspective of event logs, constraints involving roles – such as role-binding (RB) and task-to-role assignment – are more complex. This is due to the fact that normally role information is not included in event logs. Therefore, we indirectly check if a certain rule for task-to-role assignments holds in the event log by checking the subjects who perform the corresponding task. At first, we retrieve the roles a task is assigned to ($town^{-1}$, see also Section 2.1). Next, we check if one of the subjects assigned to theses roles ($rown^{-1}$) performed the respective task instance. Listing 1.3 shows the form of a corresponding check in LTL. In particular, we check if for a role R that owns a task A one of the subjects assigned to R (or to one of R's senior-roles) actually executed task A.

Checking a role-binding constraint in LTL is similar to checking task-to-role assignments. Because typically role information is not included in the event logs, we have to use the executing subjects in order to check if two role-bound tasks have been executed by the same role. Thus, we build subformulas for each role R and check if two role-bound tasks A and B have been executed by a subject assigned to this role R. Algorithm 1 shows how such a LTL formula is build. An excerpt of a LTL formula created with this algorithm is shown in Listing 1.4. It checks if two tasks A and B (assigned to role R1 and R2) were executed by subjects either owning role R1 or R2. We use placeholders and replacements in order to dynamically derive the correct structure of brackets in the LTL formula[3].

4 Consistency Checking of an Example Process

We test our approach using an event log created with CPN Tools (see [5]). In particular, we modeled the credit application process from Figure 1 in the CPN Tools environment and generated corresponding event logs in MXML format [20]. We use the CPN Tools event log simulation to determine the structure and content of the event log for our conformance check. This also allows us to integrate all kinds of violations of the access control policies in a controlled

[3] Constructs in LTL have to be structured similar to binary trees. Thus, we cannot write $(A \vee B \vee C)$, but must use $(A \vee (B \vee C))$ or $((A \vee B) \vee C)$.

```
(
 ((<>((task == %A%) /\ (subject == S1_R1 \/ (subject == S2_R1 \/ ...)))
  /\
  <>((task == %B%) /\ (subject == S1_R1 \/ (subject == S2_R1 \/ ...))))
 \/
  (<>((task == %A%) /\ (subject == S1_R2 \/ (subject == S2_R2 \/ ...)))
  /\
  <>((task == %B%) /\ (subject == S1_R2 \/ (subject == S2_R2 \/ ...)))))
 \/ !( ( <>( task == %A% ) /\ <>( task == %B% ) ) )
);
```

Listing 1.4. LTL formula to check role-binding of two tasks (excerpt)

Algorithm 1 Generation of role-binding formulas

```
1: function getLTL4RBind(task1, task2, roles)
2:     formula = '%RF%'
3:     for all role ∈ roles do
4:         formulaR = '(%TF% ∨ !((◊(task == "task1.getName()")
5:                                ∧ ◊(task == "task2.getName()")))'
6:         formulaT1 = '◊((task == "task1.getName()" ∧ %SF%))'
7:         formulaT1 = '◊((task == "task2.getName()" ∧ %SF%))'
8:         subFormula = ''
9:         subjects = role.getSubjects()
10:        for all subject ∈ subjects do
11:            subFormula = 'subject == "subject.getName()"'
12:            if !isLastSubject() then
13:                subFormula = '( subFormula ∨ %SF% )'
14:            end if
15:            formulaT1 = formulaT1.replace('%SF%', subFormula)
16:            formulaT2 = formulaT2.replace('%SF%', subFormula)
17:        end for
18:        formulaR = formulaR.replace('%TF%', '( formulaT1 ∧ formulaT2 )')
19:        if !isLastRole() then
20:            formulaR = '( formulaR ∨ %RF% )'
21:        end if
22:        formula = formula.replace('%RF%', formulaR)
23:        return formula
24:    end for
25: end function
```

manner. For example, we can manipulate the event log and include tasks and performers that do not conform to the corresponding RBAC model. In this way, we can check event logs that include all kinds of inconsistencies. In Listing 1.5 we show an excerpt of an event log created with CPN Tools.

Manually checking an event log for inconsistencies is error-prone and time-consuming. Therefore, our approach supports the automated definition of LTL statements from the XML representation of process-related RBAC models (see

```
...
<AuditTrailEntry>
 <WorkflowModelElement>Check credit worthiness</WorkflowModelElement>
 <Originator>Bob</Originator>
 ...
</AuditTrailEntry>
<AuditTrailEntry>
 <WorkflowModelElement>Negotiate contract</WorkflowModelElement>
 <Originator>Bob</Originator>
 ...
</AuditTrailEntry>
```

Listing 1.5. Excerpt of a simulated event log for a credit application process

```
...
<node id="BA01" xsi:type="BusinessAction" name="Check credit worthiness">
  <subjectBinding resource="BA02"/>
</node>
<node id="BA02" xsi:type="BusinessAction" name="Negotiate contract">
  <dynamicExclusion resource="BA03"/>
  <subjectBinding resource="BA01"/>
</node>
...
```

Listing 1.6. Excerpt of the process-related RBAC model instance in XML representation

Section 3). For this purpose, we converted the graphical model from Figure 1 into its corresponding XML representation. Listing 1.6 shows an excerpt of a corresponding XML document including the two subject-bound tasks "Check credit worthiness" and "Negotiate contract".

Now we use our RBAC-to-LTL component (see Section 3) to parse the XML document and derive LTL statements for all properties defined in the RBAC model. Listing 1.7 shows the generated LTL construct for subject-binding of the tasks "Check credit worthiness" and "Negotiate contract".

Subsequently, the automatically generated LTL statements can be imported in a software such as ProM to analyze the corresponding event logs and to reveal violations of the policies defined via the respective process-related RBAC model. Figure 5 shows the result of a corresponding analysis in ProM for our event log of the credit application process. In general, we have two different views

```
formula SB_task_check_credit_worthiness_and_negotiate_contract () :=
{
    <p>Task "Check credit worthiness" and task "Negotiate contract"
      must be executed by the same person.</p>
}
SB_task_A_and_B( "Check credit worthiness", "Negotiate contract" );
```

Listing 1.7. LTL formula to check the subject-bound tasks "Check credit worthiness" and "Negotiate contract"

in ProM: the rule perspective and the instance perspective. Both perspectives are composed similarly. As shown in Figure 5, the rule perspective has a tab for satisfied rules and a tab for unsatisfied rules. For each of the unsatisfied rules it shows which process instances (cases) in the event log satisfy this rule and which do not. To directly see the respective violation we can select the case and inspect its event log entries. For example, Figure 5 shows a violated subject-binding constraint in case 15 for the subject-bound tasks "Check credit worthiness and "Negotiate contract". In the rightmost view from Figure 5 we can see the event logs entries for this process instance. Furthermore, we see which two subjects executed these subject-bound tasks as well as the date and time of this execution event. In this case, two users named Lea and Bob have been executing these subject-bound tasks.

Fig. 5. LTL Checker results for the simulated credit application process

5 Related Work

In [9], Hansen and Oleshchuk introduce an approach to check the implementation of RBAC constraints using the Spin[4] model checker. They formally express the given RBAC properties via LTL. To check the conformance in Spin, Hansen and Oleshchuk use the RBAC implementation in PROMELA, the internal specification language of Spin. In contrast to our approach, however, they do not assess the compliance of actual process executions with a corresponding RBAC model. A similar approach was presented by Ahmed and Tripathi [3], who specify and statically verify security requirements for CSCW (Computer Supported Cooperative Work) systems with Spin.

[4] http://spinroot.com/

Moreover, the work on business process compliance checking from the area of process mining is directly related to the work presented in this paper. Van der Aalst et al. [2] presented a general approach to verify different properties in event logs. Furthermore, in [17] van der Aalst and de Medeiros apply process mining to address security issues. They analyze event logs to discover security violations in process execution. In particular, they check if new events in a certain process execution comply with a process model that defines acceptable behavior. In addition, they introduce an approach to check whether new audit trails conform to the predefined ordering relation of tasks. Another approach in this area has been introduced by Rozinat and van der Aalst [12], where the stream of events is replayed from a log in order to reveal inconsistencies. However, the focus of these approaches is on analyzing the process flow perspective. Thereby, our work supplements these approaches with a perspective on process-related RBAC models.

6 Conclusion and Outlook

In this paper, we presented an approach to automatically generate LTL statements from process-related RBAC models (see Sections 2.1 and 3). The LTL statements generated by our RBAC-to-LTL software component can be applied to check the event logs of business process instances for violations of the corresponding RBAC policies. For example, these checks help to find misconfigurations or implementation errors in PAIS. Thus, the results of such an event log analysis can give a first insight into the modifications of the corresponding PAIS or its configuration which are necessary to comply with a tailored RBAC model. To test our approach, we used the ProM tool to import the LTL statements that are generated by our RBAC-to-LTL component and checked the event logs of a credit application process. In our future work, we plan to integrate our work with related approaches for analyzing the control flow.

References

1. van der Aalst, W.M.P., Weijters, A.J.M.M.: Process mining: a research agenda. Computers in Industry 53 (April 2004)
2. van der Aalst, W., de Beer, H., van Dongen, B.: Process Mining and Verification of Properties: An Approach based on Temporal Logic. In: On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science (LNCS), Vol. 3760, Springer Verlag. pp. 130–147 (2005)
3. Ahmed, T., Tripathi, A.R.: Static verification of security requirements in role based CSCW systems. In: Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT) (2003)
4. de Beer, H.: The LTL Checker Plugins: A Reference Manual. Eindhoven University of Technology, Eindhoven (2004)
5. de Medeiros, A., Günther, C.W.: Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In: Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. pp. 177–190 (2005)

6. Dumas, M., van der Aalst, W., ter Hofstede, A.: Process-Aware Information Systems. John Wiley & Sons, Inc. (2005)
7. El Kharbili, M., Alves de Medeiros, A., Stein, S., van der Aalst, W.: Business Process Compliance Checking: Current State and Future Challenges. In: MobIS. pp. 107–113 (2008)
8. Ferraiolo, D., Kuhn, D., Chandramouli, R.: Role-Based Access Control, Second Edition. Artech House (2007)
9. Hansen, F., Oleshchuk, V.: Conformance Checking of RBAC Policy and its Implementation. In: Information Security Practice and Experience, Lecture Notes in Computer Science (LNCS), Vol. 3439, Springer Verlag (2005)
10. Pnueli, A.: The Temporal Logic of Programs. In: Foundations of Computer Science. pp. 46–57 (1977)
11. Rozinat, A., van der Aalst, W.: Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In: Bussler, C., Haller, A. (eds.) Business Process Management Workshops, Lecture Notes in Computer Science (LNCS), Vol. 3812, Springer Verlag (2006)
12. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64 – 95 (2008)
13. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-Based Access Control Models. IEEE Computer 29(2) (February 1996)
14. Sandhu, R., Samarati, P.: Access Control: Principles and Practice. IEEE Communications 32(9) (September 1994)
15. Strembeck, M.: Scenario-Driven Role Engineering. IEEE Security & Privacy 8(1) (January/February 2010)
16. Strembeck, M., Mendling, J.: Modeling process-related RBAC models with extended UML activity models. Information and Software Technology 53(5), 456 – 483 (2011)
17. van der Aalst, W., de Medeiros, A.: Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. Electronic Notes in Theoretical Computer Science 121, 3 – 21 (2005)
18. van der Aalst, W., van Dongen, B., C. Günther, A.R., Verbeek, H.M.W., Weijters, A.J.M.M.: ProM: The Process Mining Toolkit. In: Proceedings of the BPM 2009 Demonstration Track. vol. 489. CEUR-WS.org (September 2009)
19. van der Aalst, W., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering 47(2) (2003)
20. van Dongen, B., van der Aalst, W.: A Meta Model for Process Mining Data. In: Proceedings of the Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability (2005)
21. Verbeek, H.M.W., Buijs, J., van Dongen, B., van der Aalst, W.: ProM 6: The Process Mining Toolkit. In: Proceedings of BPM 2010 Demonstration Track. vol. 615, pp. 34–39. CEUR-WS.org (September 2010)