Checking Satisfiability Aspects of Binding Constraints in a Business Process Context

Sigrid Schefer[1], Mark Strembeck[1], and Jan Mendling[2]

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

Abstract. Binding of Duty (BOD) constraints define that the same subject (or role) who performed a certain task t_1 must also perform a corresponding bound task t_2. In this paper, we describe algorithms for checking the *satisfiability of binding constraints* in a business process context. In particular, these algorithms check the configuration of a process-related RBAC model to find satisfiability conflicts. Furthermore, we discuss options to resolve satisfiability conflicts.

Key words: access control, binding of duty, business processes

1 Introduction

Separation of duty (SOD) and *Binding of Duty (BOD)* constraints specify rules to control task allocation and execution in workflows (see, e.g., [2, 3, 5, 6, 7, 8, 9]). They constrain task authorizations by defining that two (or more) tasks must be performed by different individuals (SOD) or by the same individual (BOD). In a business process context, SOD constraints enforce conflict of interest policies. Conflict of interest arises as a result of the simultaneous assignment of two mutually exclusive tasks to the same subject. Thus, mutually exclusive tasks result from the division of powerful rights to prevent fraud and abuse.

BOD can be subdivided into subject-based and role-based constraints (see, e.g., [5, 6]). A *subject-based BOD constraint* defines that the same individual who performed the first task must also perform the bound task(s). In contrast, a *role-based BOD constraint* defines that bound tasks must be performed by members of the same role, but not necessarily by the same individual. Throughout the paper, we will use the terms *subject-binding* and *role-binding* as synonyms for subject-based and role-based BOD constraints respectively. *Satisfiability* of a business process requires that a set of authorized subjects is able to perform all tasks in the workflow (see, e.g., [4, 7]). However, process verification typically focuses on pure control flow aspects such as soundness [1]. In this paper, we look at workflow verification with a focus on workflow satisfiability aspects of process-related binding constraints.

2 Satisfiability of Binding constraints

In [5], we presented a set of algorithms to ensure the consistency of process-related RBAC models. However, a RBAC model can be consistent while at the same time the corresponding processes may still not be satisfiable. The algorithms defined below detect satisfiability conflicts of workflows that include binding constraints. Note that the checks in the if-clauses of our algorithms complement each other. Thus, checks of prior if-clauses do not have to be repeated in subsequent if-clauses. The algorithms' runtime complexity is in the worst-case scenario $\mathcal{O}(n^2)$. The worst case memory consumption for the sets of elements is $\mathcal{O}(n)$, for relations among these elements it amounts to $\mathcal{O}(n^3)$. The underlying formal definitions and consistency requirements are specified in [5, 6].

To ensure the satisfiability of a *subject-binding* (SB) constraint, subject-bound tasks must be assigned to the same subject, either directly or transitively via the role-hierarchy. Algorithm 1 checks if a SB constraint specified on two task types t_1 and t_2 is satisfiable. If a satisfiability conflict is detected, the algorithm returns the name of the respective conflict. In Algorithm 1, line 1 first checks if a SB constraint is defined on two task types t_1 and t_2. Only if a SB constraint is specified, the algorithm proceeds with the subsequent satisfiability checks.

Algorithm 1 *Check if a SB constraint on two task types is satisfiable.*

Name: $isSBconstraintSatisfiable(t_1, t_2)$
Input: $t_1, t_2 \in T_T$
1: *if* $t_1 \notin sb(t_2)$ *then return* `true`
2: *if* $\nexists s \in S, r_1, r_2 \in R \mid r_1 \in rown(s) \ \wedge \ r_2 \in rown(s) \ \wedge$
3: $\quad t_1 \in town(r_1) \ \wedge \ t_2 \in town(r_2)$
4: $\quad$ *then return* `SubjectAssignmentConflict`
5: *if* $t_1 \in dme(t_x)$ *then* (
6: $\quad$ *if* $\nexists s_1, s_x \in S, r_1, r_x \in R \mid s_1 \neq s_x \ \wedge \ r_1 \in rown(s_1) \ \wedge$
7: $\qquad r_x \in rown(s_x) \ \wedge \ t_1 \in town(r_1) \ \wedge \ t_x \in town(r_x)$
8: $\qquad$ *then return* `TransitiveDMEConflict`)
9: *if* $t_2 \in dme(t_x)$ *then* (
10: $\quad$ *if* $\nexists s_2, s_x \in S, r_2, r_x \in R \mid s_2 \neq s_x \ \wedge \ r_2 \in rown(s_2) \ \wedge$
11: $\qquad r_x \in rown(s_x) \ \wedge \ t_2 \in town(r_2) \ \wedge \ t_x \in town(r_x)$
12: $\qquad$ *then return* `TransitiveDMEConflict`)
13: *return* `true`

Subject-Assignment Conflict: Algorithm 1, lines 2-4 check if at least one subject is assigned to a role which owns the subject-bound tasks (see [5, 6]). Otherwise, a subject-assignment conflict occurs. In Figure 1a, two subject-bound task types t_1 and t_2 are assigned to r_1, but no subject is assigned to r_1 which causes an unsatisfiable SB constraint. Moreover, if subject-bound tasks are assigned to different roles, at least one subject must be assigned to all roles that own the subject-bound tasks. This type of subject-assignment conflict is

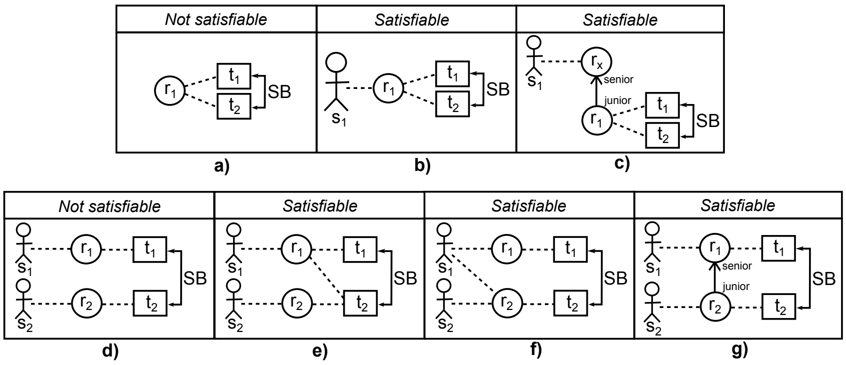

Fig. 1. Subject-assignment conflicts affecting the satisfiability of SB constraints

shown in Figure 1d where two subject-bound task types t_1 and t_2 are assigned to different roles and different subjects.

Resolutions to Subject-Assignment Conflict: Figures 1b and 1c show two options to resolve a subject-assignment conflict. The SB constraint is satisfiable if at least one subject is authorized to perform both task types. To resolve the satisfiability conflict in Figure 1d the following resolutions are applicable. Firstly, t_1 and t_2 can be assigned to the same subject s_1 by assigning both tasks to r_1 (Figure 1e). Secondly, s_1 can be assigned to the role r_2 which owns t_2 (Figure 1f). Alternatively, r_1 can be defined as senior role of r_2 (Figure 1g). Subsequently, s_1 can perform t_1 and the inherited task t_2.

Transitive DME-Conflict: The simultaneous definition of SB and dynamic mutual exclusion (DME) constraints on tasks is not possible as they cannot be satisfied at the same time (see [5, 6]). Yet, a DME constraint can be defined on one of the subject-bound tasks and a third task.

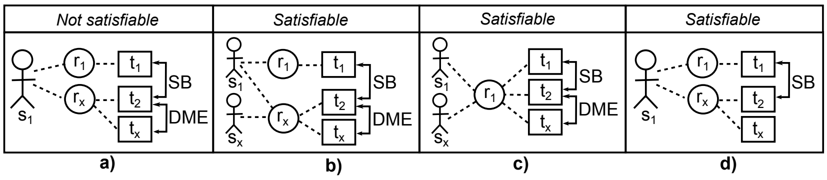

Fig. 2. Transitive DME-conflicts affecting the satisfiability of SB constraints

Algorithm 1, lines 5-12 check the satisfiability of a SB constraint on two tasks t_1 and t_2 if a DME constraint is defined on t_1 or t_2 and some other task type t_x. This configuration is not satisfiable if only a single subject s_1 is authorized to perform these tasks (see Figure 2a). Due to the DME constraint, instances of t_2 and t_x cannot be performed by the same subject in the same process instance. Therefore, we need at least two subjects to perform the three tasks. Consequently, either the SB or the DME constraint is not satisfiable in Figure 2a.

Resolutions to Transitive DME-Conflict: Figure 2 illustrates three options to resolve this satisfiability conflict. Firstly, another subject s_x can be assigned to the role owning t_x (see Figure 2b). In a particular process instance, s_1 can perform t_1 and t_2 and thereby satisfy the SB constraint and s_x performs t_x to satisfy the DME constraint. A similar resolution is shown in Figure 2c, where all three tasks are assigned to r_1. However, as two subjects are authorized to perform the three tasks, the SB and the DME constraints are satisfiable. Alternatively, the conflicting DME constraint can be removed (see Figure 2d).

To ensure the satisfiability of a *role-binding (RB)* constraint, role-bound tasks must be assigned to the same role. In Algorithm 2, line 1 first checks if a RB constraint is defined on two task types t_1 and t_2. Only if a RB constraint is specified, the algorithm proceeds with the subsequent satisfiability checks.

Algorithm 2 *Check if a RB constraint on two task types is satisfiable.*

Name: $isRBConstraintSatisfiable(t_1, t_2)$
Input: $t_1, t_2 \in T_T$
1: *if* $t_1 \notin rb(t_2)$ *then return* `true`
2: *if* $\nexists r \in R \mid t_1 \in town(r) \ \wedge \ t_2 \in town(r)$
3: *then return* `RoleAssignmentConflict`
4: *if* $\nexists s \in S, r \in R \mid r \in rown(s) \ \wedge \ t_1 \in town(r) \ \wedge \ t_2 \in town(r)$
5: *then return* `SubjectAssignmentConflict`
6: *if* $t_1 \in dme(t_2)$ *then* (
7: *if* $\nexists s_1, s_2 \in S, r \in R \mid s_1 \neq s_2 \ \wedge \ r \in rown(s_1) \ \wedge$
8: $r \in rown(s_2) \ \wedge \ t_1 \in town(r) \ \wedge \ t_2 \in town(r)$
9: *then return* `DirectDMEConflict`)
10: *if* $t_1 \in dme(t_x)$ *then* (
11: *if* $\nexists s_1, s_x \in S, r \in R \mid s_1 \neq s_x \ \wedge \ r \in rown(s_1) \ \wedge$
12: $r \in rown(s_x) \ \wedge \ t_1 \in town(r) \ \wedge \ t_x \in town(r)$
13: *then return* `TransitiveDMEConflict`)
14: *if* $t_2 \in dme(t_x)$ *then* (
15: *if* $\nexists s_2, s_x \in S, r \in R \mid s_2 \neq s_x \ \wedge \ r \in rown(s_2) \ \wedge$
16: $r \in rown(s_x) \ \wedge \ t_2 \in town(r) \ \wedge \ t_x \in town(r)$
17: *then return* `TransitiveDMEConflict`)
18: *return* `true`

Role-Assignment Conflict: Line 2 of Algorithm 2 checks if a role r exists which owns both role-bound tasks t_1 and t_2, either directly or transitively via the role-hierarchy. Otherwise, Algorithm 2, line 3 returns a role-assignment conflict. Figure 3a shows an example where the current task-to-role assignments defined for t_1 and t_2 result in an unsatisfiable RB constraint.

Resolutions to Role-Assignment Conflict: Figure 3 illustrates two options to resolve a role-assignment conflict. Firstly, both tasks can be assigned to the same role (see Figure 3b). Secondly, t_1 and t_2 can be assigned to two roles where one of the roles, e.g., r_1 is a senior role of the second role, e.g., r_2 (see Figure 3c). As a result, members of r_1 can perform t_1 and the inherited task t_2.

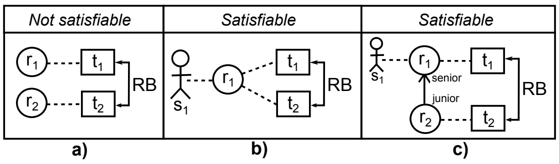

Fig. 3. Role-assignment conflict affecting the satisfiability of RB constraints

Subject-Assignment Conflict: Algorithm 2, line 4 checks if there is at least one subject assigned to a role owning two role-bound tasks. Otherwise, Algorithm 2, line 5 returns a subject-assignment conflict.

Resolutions to Subject-Assignment Conflict: The RB constraint in Figure 3b is satisfiable if at least one subject is assigned to r_1. Alternatively, each subject owning a senior-role of r_1 can perform t_1 and t_2 (see Figure 3c).

Direct DME-Conflict: A DME constraint can be defined on role-bound tasks or on one of the role-bound tasks and a third task. Figures 4a and 4d show corresponding example configurations. Usually, DME constraints and RB constraints do not conflict (see [5, 6]). However, in case only a single subject is assigned to a role owning role-bound and DME tasks, either the DME or the RB constraint cannot be satisfied.

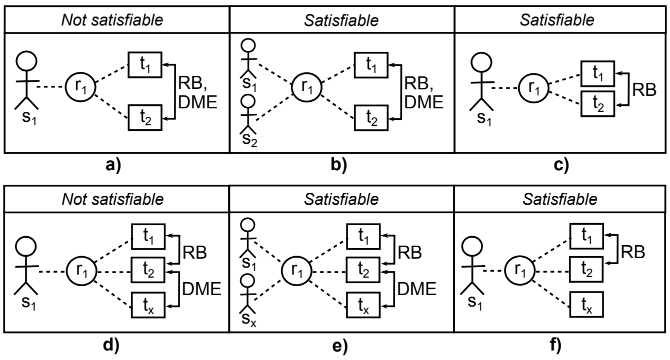

Fig. 4. DME-conflicts affecting the satisfiability of RB constraints

Algorithm 2, lines 6-9 check the satisfiability of a RB constraint on two task types t_1 and t_2 if a DME constraint is defined on t_1 and t_2 at the same time. This configuration is shown in Figure 4a. In order to fulfill both constraints, at least two subjects need to be assigned to r_1.

Resolutions to Direct DME-Conflict: Figure 4 shows two options for resolving this conflict. A second subject s_2 can be assigned to r_1 (see Figure 4b). Then, each of the two subjects can perform one of the two role-bound and DME tasks. Thus, the RB as well as the DME constraint are satisfiable, because t_1 and t_2 can be performed by two different subjects. Alternatively, the RB constraint is satisfiable if the conflicting DME constraint is removed (see Figure 4c).

Transitive DME-Conflict: Algorithm 2, lines 10-17 check the satisfiability of a RB constraint on two task types t_1 and t_2 if a DME constraint is defined on either t_1 or t_2 and some other task type t_x. This configuration is shown in Figure 4d. Due to the DME constraint, instances of t_2 and t_x cannot be performed by a single individual in the same process instance (see [5, 6]). Thus, we need at least two subjects to execute instances of these three tasks. Consequently, either the RB or the DME constraint is not satisfiable if only a single subject s_1 is assigned to r_1.

Resolutions to Transitive DME-Conflict: Figure 4 illustrates two options to resolve this satisfiability conflict. Firstly, a second subject s_x can be assigned to r_1 (see Figure 4e). Alternatively, the RB constraint is satisfiable if the conflicting DME constraint is removed (see Figure 4f).

3 Conclusion

Satisfiability of a workflow guarantees that there is always a set of authorized subjects that allows a process to proceed. In this paper, we addressed satisfiability aspects of workflows that include subject-binding and/or role-binding constraints in a process-related RBAC context. For this purpose, we provided algorithms to check if a given binding constraint is satisfiable. In addition, we discussed different options to resolve satisfiability conflicts.

References

1. W. v. d. Aalst. *Business Process Management*, LNCS 1806, chapter Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques, pages 161–183. 2000.
2. R. A. Botha and J. H. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3), 2001.
3. F. Casati, S. Castano, and M. Fugini. Managing Workflow Authorization Constraints through Active Database Technology. *Inf. Sys. Frontiers*, 3(3), 2001.
4. J. Crampton and H. Khambhammettu. Delegation and Satisfiability in Workflow Systems. In *Proceedings of ACM SACMAT*, 2008.
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 CoopIS*, LNCS 6426, 2010.
6. M. Strembeck and J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Inf. Software Techn.*, 53(5), 2011.
7. K. Tan, J. Crampton, and C. A. Gunter. The Consistency of Task-Based Authorization Constraints in Workflow Systems. In *Proceedings of the 17th IEEE workshop on Computer Security Foundations*, June 2004.
8. J. Wainer, P. Barthelmess, and A. Kumar. W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints. *International Journal of Cooperative Information Systems*, 12(4), 2003.
9. J. Warner and V. Atluri. Inter-instance authorization constraints for secure workflow management. In *Proc. of ACM SACMAT*, 2006.