Modeling Support for Role-Based Delegation in Process-Aware Information Systems

The paper presents an integrated approach for the modeling and enforcement of delegation policies in process-aware information systems. Based on a generic formal metamodel, the detection of delegation-related conflicts is discussed and a set of pre-defined resolution strategies for these conflicts is provided. Moreover, a corresponding UML extension, a prototypical proof of concept implementation, and a case study with real-world business processes is provided.

DOI 10.1007/s12599-014-0343-3

The Authors

Dr. Sigrid Schefer-Wenzl (✉)
Institute for Information Systems and New Media
WU Vienna
1090 Vienna
Austria
and
Competence Center for IT-Security
University of Applied Sciences
Campus Vienna
1090 Vienna
Austria
sigrid.schefer-wenzl@wu.ac.at

Prof. Dr. Mark Strembeck
Institute for Information Systems and New Media
WU Vienna
1090 Vienna
Austria
mark.strembeck@wu.ac.at

Received: 2013-03-26
Accepted: 2014-02-03
Accepted after two revisions by Prof. Dr. Becker.
Published online: 2014-08-23

This article is also available in German in print and via http://www.wirtschaftsinformatik.de: Schefer-Wenzl S, Strembeck M (2014) Modellierungsunterstützung für die rollenbasierte Delegation in prozessgestützten Informationssystemen. WIRTSCHAFTSINFORMATIK. doi: 10.1007/s11576-014-0433-3.

Electronic Supplementary Material
The online version of this article (doi: 10.1007/s12599-014-0343-3) contains supplementary material, which is available to authorized users.

1 Introduction

A business process consists of a set of tasks which are performed by an organization to reach certain corporate goals (see, e.g., zur Muehlen and Indulska 2010). If the execution of a business process is supported via an information system, instances of a business process are also referred to as workflows (see, e.g., Weske 2012). To support the secure execution of a particular workflow, subjects participating in a workflow must own the permissions that are needed to execute the corresponding tasks (see, e.g., Georgiadis et al. 2001; Oh and Park 2003; Strembeck and Mendling 2011; Thomas and Sandhu 1997; Wainer et al. 2003). In recent years, role-based access control (RBAC) (see, e.g., Ferraiolo et al. 2007; Sandhu et al. 1996) has developed into a de facto standard for access control in research and industry. In RBAC, roles are used to model different job positions and responsibilities within an organization and/or information system. Access permissions are assigned to roles according to the tasks each role has to accomplish. These roles are then assigned to human users according to their respective work profile (see, e.g., Strembeck 2010). Roles are also used as an abstract concept for delegation (see, e.g., Crampton and Khambhammettu 2008b; Wainer et al. 2007) as well as for the assignment of duties defined via obligations (see, e.g., Schaad and Moffett 2002; Strembeck 2005; Zhao et al. 2007).

While authorization policies define a subject's permissions, obligation policies define a subject's duties (see, e.g., Cole et al. 2001; Sloman 1994; Strembeck 2005). Delegation provides a mechanism to increase flexibility in authorization and obligation management. In essence, a subject can delegate its tasks, duties, or roles to another subject (see, e.g., Schaad and Moffett 2002). Subsequently, the subject receiving the delegation (the delegatee) will act on behalf of the delegating subject (the delegator). Delegation has been identified as an important concept in workflow systems and many other application areas (see, e.g., Crampton and Khambhammettu 2008c; Hasebe et al. 2010; Ravichandran and Yoon 2006).

In addition, we support the definition of different types of entailment constraints. A task-based entailment constraint places some restriction on the subjects who can perform a $task_x$ given that a certain subject has performed $task_y$. Thus, task-based entailment constraints have an impact on the combination of subjects and roles who are allowed (or required) to execute particular tasks (see, e.g., Russell et al. 2005; Schefer et al. 2011; Strembeck and Mendling 2010, 2011; Tan et al. 2004; Warner and Atluri 2006; Wolter et al. 2008). In process-aware information systems, mutual exclusion constraints enforce conflict of interest policies by defining that two or more tasks must be performed by different individuals. Conflict of interest arises as a result of the simultaneous assignment of two mutual

exclusive entities (e.g., permissions or tasks) to the same subject. In contrast, binding constraints specify that bound tasks must always be performed by the same subject or role (see, e.g., Strembeck and Mendling 2010, 2011; Tan et al. 2004; Wainer et al. 2003). They can be subdivided into subject-based and role-based constraints (see, e.g., Strembeck and Mendling 2010, 2011).

The immanent complexity of delegations and task-based entailment constraints is a central problem for delegation in process-aware information systems (see, e.g., Crampton and Khambhammettu 2008a; Schaad 2001). Thus, when delegating tasks, roles, or duties, corresponding design-time and runtime checks need to ensure the consistency of the respective RBAC model. In particular, at design-time conflicts may result from delegations which are inconsistent with the corresponding RBAC model, especially regarding related entailment constraints. At runtime, conflicts may result from inconsistent task allocations.

The main contribution of this paper is the consideration of delegation aspects when checking and ensuring the consistency of process-related RBAC models. Our integrated modeling approach for delegation policies and corresponding processes acts as an enabler to document and communicate more efficiently which delegation aspects need to be considered when executing a certain process. To achieve this, we consolidate and extend our previous publications from Schefer and Strembeck (2011b) and Schefer-Wenzl et al. (2012): Our approach is based on a metamodel which formally integrates the core elements of process models and delegation models for roles, tasks, and duties (see **Fig. 2**). This metamodel was presented in Schefer-Wenzl et al. (2012). Corresponding modeling support is provided by extending the Unified Modeling Language (UML; OMG 2011b). In particular, we introduced modeling support for delegating roles, tasks, and duties in Schefer and Strembeck (2011b) via extended UML2 activity diagrams. In Schefer-Wenzl et al. (2012), we provide a set of algorithms to detect and name potential delegation-related conflicts. Moreover, these algorithms check and ensure the consistency of mutual-exclusion and binding constraints in business processes in the context of delegation.

We apply model-driven development (MDD) techniques (see, e.g., Schmidt 2006; Selic 2003; Stahl and Völter 2006) to support the integrated modeling and execution of delegation policies and business processes. In the MDD context, a computation-independent model (CIM) defines a certain domain (or subdomain) at a generic level. The CIM is independent of a particular modeling language or technology. A CIM can be used to build a platform-independent model (PIM) of the corresponding domain. While it is independent of any platform, and thereby neutral from an implementation point of view, the PIM is typically specified in a particular modeling language (for example via MOF-based languages such as BPMN or UML) and describes the structure of a system, the elements/results that are produced by a system, or the control and object flow in a system. Finally, a platform-specific model (PSM) describes the realization/implementation of a software system via platform-specific technologies and tools. In particular, we chose a model-driven approach to separate the business/application logic from the underlying platform technology (see, e.g., Schmidt 2006; Selic 2003; Stahl and Völter 2006).

In this paper, we consolidate the results from Schefer and Strembeck (2011b) and Schefer-Wenzl et al. (2012) and present the following novel contributions: We introduce a set of pre-defined resolution strategies for conflicts resulting from the delegation of roles, tasks and duties in business processes. In addition, we implemented all concepts introduced in the formal metamodel and in the UML extension as an extension to the BusinessActivity library and runtime engine (available from BAL 2012). This extended software platform enables a seamless mapping between modeling-level specifications of processes, RBAC policies, and delegation policies to the corresponding runtime models. Furthermore, we conducted a case study to evaluate the applicability of our UML extension for real-world processes.

The remainder of this paper is structured as follows. In Sect. 2, we shortly introduce terminology and present the formal definitions of a process-related RBAC delegation model at the CIM-layer. Section 3 discusses different conflicts that may occur when delegating roles, tasks, or duties. Moreover, we provide corresponding conflict resolution strategies that ensure the consistency of process-related RBAC delegation models. Subsequently, Sect. 4 provides algorithms to automatically detect the conflicts discussed in Sect. 3. Section 5 presents a UML extension that allows to model the delegation of roles, tasks, and duties via extended UML2 Activity diagrams at the PIM-layer. Section 6 presents a case study to illustrate the practical applicability of our UML extension in real business settings. Subsequently, Sect. 7 gives an overview of our extended software platform to manage process-related RBAC delegation models at the PSM-layer. Finally, Sect. 8 discusses related work and Sect. 9 concludes the paper.

2 Process-Related RBAC Delegation Models

Figure 1 shows a simplified credit application process visualized as standard UML2 activity diagram which will serve as a running example in this paper. When executing this process, human users and autonomous software agents have to fulfill certain tasks. Each task in a workflow (such as checking a customer's creditworthiness or negotiating a contract) is typically associated with certain access operations (e.g., to access the customer's record or to sign the contract, respectively). Thus, a subject participating in a workflow must be authorized to perform the tasks needed to complete the process (see, e.g., Georgiadis et al. 2001; Oh and Park 2003; Strembeck and Mendling 2011).

In organizational contexts, tasks are also associated with duties (see, e.g., Schefer and Strembeck 2011a). Each duty defines an action that must be performed by a certain subject in order to comply with legal or organizational regulations (see, e.g., Strembeck 2005). For example, the task "negotiate contract" from **Fig. 1** can be associated with the duty "fulfill pre-contractual information duties" stemming from respective legal requirements. A subject responsible for performing this task and the associated duty needs all necessary access permissions to perform them (see, e.g., Strembeck 2005; Zhao et al. 2007).

An organization's business processes and software systems are often modeled via graphical modeling languages. However, corresponding organizational policies are often specified via informal textual descriptions (see, e.g., Recker et al. 2006). Thus, a link between tasks, duties, and subjects/roles is usually missing. This missing link can easily result

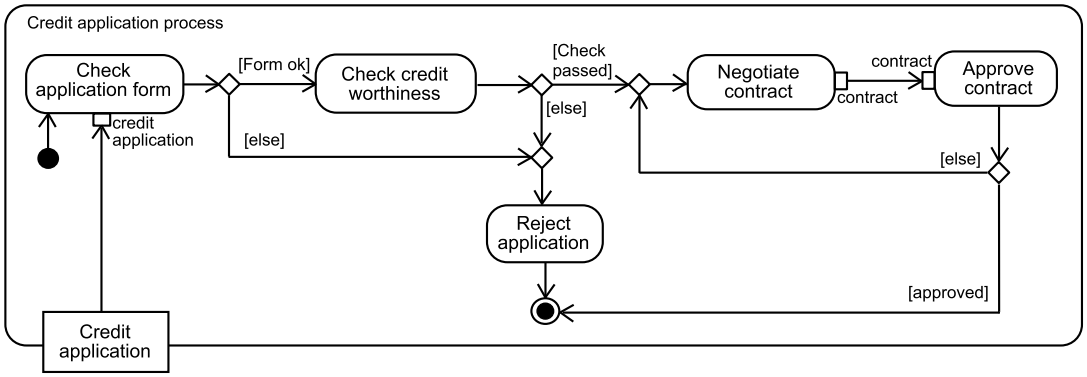

Fig. 1 Credit application process modeled as standard UML2 activity diagram

in intransparencies and inconsistencies between process descriptions and actual process executions (see, e.g., Wolter et al. 2009). Especially in the case of delegation, where the original subject assignment is changed once or even more than once (in case of multi-step delegation, see, e.g., Barka and Sandhu 2000b), the definition, monitoring, and actual enforcement of delegated tasks and duties is difficult. Therefore, we propose an integrated modeling approach for business processes and the delegation of roles, tasks, and duties. Such an integrated modeling approach also facilitates the control and reporting on a company's fulfillment of compliance requirements. Additionally, the integration also supports the elicitation and definition of task- or duty-related constraints, such as mutual exclusion or binding constraints.

2.1 Delegation in a Business Process Context

In our process-related delegation model, roles, tasks, and duties are delegable. In essence, a subject can delegate the right to perform a task and associated duties to another subject. In case of a permanent delegation, the delegation is valid for all executions of a particular business process (e.g., a credit application process). In case of a temporary delegation, it is only valid for one process instance (e.g., Mr. Mayer's credit application process).

In the context of role-based access control, several delegation approaches use the concept of so-called delegation roles (see, e.g., Joshi and Bertino 2006; Shang and Wang 2008; Zhang et al. 2003b). In our delegation model, a delegation role is created by the delegator and comprises a set of delegated tasks and duties (similar to Zhang et al. 2003b). In this way, each duty is associated with a certain task (see, e.g., Schefer and Strembeck 2011a). A delegator can delegate all or a subset of his/her delegable tasks, duties, or roles by assigning them to a delegation role. Subsequently, delegation roles are assigned to delegatees and can either be defined for temporary or for permanent delegation. Permanent delegation roles authorize the delegatee to perform the delegated tasks and duties in all instances of the process. In contrast, a temporary delegation role (DRT) authorizes the delegatee to perform the delegated tasks and duties only in particular process instances. In general, delegation roles and all assignments to delegation roles are managed by the delegating subject. All other roles are called regular roles (RR) and are usually managed by the security officer of the respective company. **Figure 2** visualizes the elements of the formal delegation metamodel which will be integrated into process-related RBAC models below (see Definition 4 from Sect. 2.2).

Different kinds of entailment constraints can be defined on tasks to restrict which subjects are allowed to execute a particular task. In this paper, we focus on mutual-exclusion and binding constraints. Static mutual-exclusion (SME) and dynamic mutual-exclusion (DME) constraints are used in workflows to enforce conflict of interest policies (see, e.g., Botha and Eloff 2001; Casati et al. 2001; Strembeck and Mendling 2010, 2011; Tan et al. 2004; Wainer et al. 2003; Warner and Atluri 2006). A SME constraint defines that two statically mutual exclusive tasks must not be assigned to the same subject. In turn, a DME constraint defines that two dynamically mutual exclusive tasks must not be executed by the same subject in the same process instance. Moreover, subject-binding (SB) and role-binding (RB) constraints can be defined which are used to enforce process-related binding-of-duty constraints (see, e.g., Strembeck and Mendling 2010, 2011). In particular, a SB constraint defines that two bound tasks must be performed by the same individual. In turn, a RB constraint defines that bound tasks must be performed by members of the same role, but not necessarily by the same individual.

To ensure the proper enforcement of entailment constraints, we also need to consider these constraints in the context of delegation (see Sect. 4). For example, the delegation of tasks, duties, or roles must not authorize the delegatee to perform two SME tasks. In contrast, when delegating a task which has a subject-binding to another task, both tasks have to be delegated to the same subject. Otherwise, the subject-binding constraint cannot be fulfilled.

2.2 Formal Metamodel for Process-Related RBAC Delegation Models

In this Section, we provide the formal definitions for process-related RBAC delegation models at the CIM layer. For the purposes of this paper, Definitions 1–3 summarize the definitions for process-related RBAC models (for further details see Strembeck and Mendling 2011). The formal definitions then serve as a basis for extending arbitrary process modeling languages and process engines with support for process-related RBAC delegation models. In this paper, we will demonstrate this by extending the UML (see Sect. 5). Definition 1 specifies the essential elements of process-related RBAC models and their basic interrelations. In particular, we model authority via roles, role-hierarchies, and task-to-role assign-

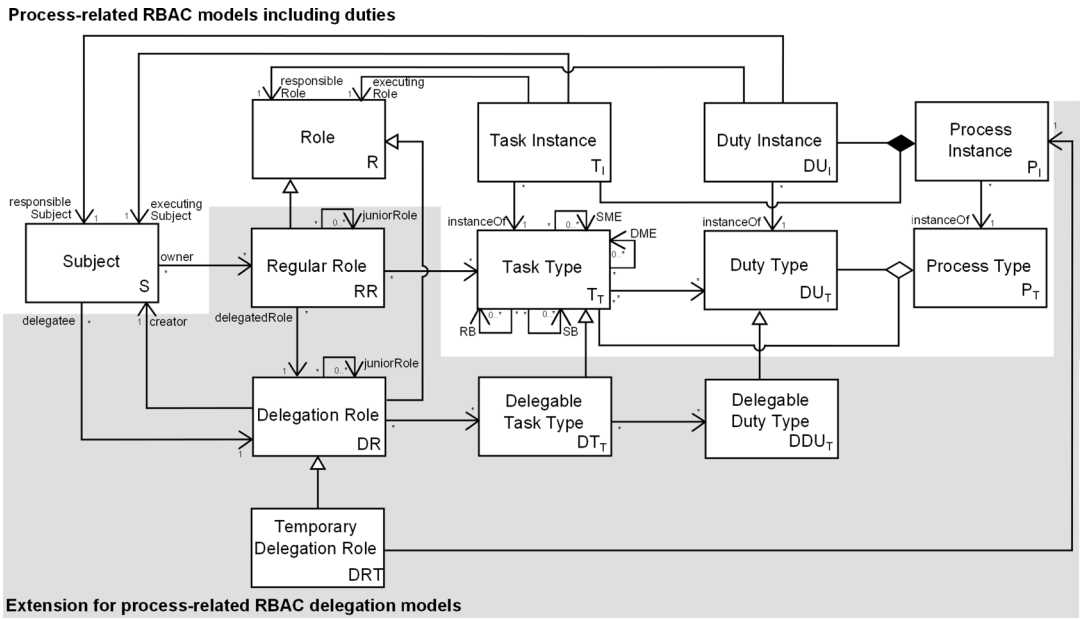

Fig. 2 Conceptual overview: Main elements of process-related RBAC delegation models

ments. Responsibility is modeled via duties which are linked to tasks and assigned to subjects. Subjects and corresponding duties together form obligation policies. Competence is modeled via role-to-subject assignments.

Definition 1 (Process-related RBAC model) 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, $Q = rh \cup rsa \cup tra \cup es \cup er \cup ar \cup pi \cup ti$ to mappings that establish relationships, and $D = sb \cup rb \cup sme \cup dme$ to binding and mutual-exclusion constraints, such that:

- For the sets of the meta model:
  - An element of S is called *Subject*. $S \neq \emptyset$.
  - An element of R is called *Role*. $R \neq \emptyset$.
  - An element of P_T is called *Process Type*. $P_T \neq \emptyset$.
  - An element of P_I is called *Process Instance*.
  - An element of T_T is called *Task Type*. $T_T \neq \emptyset$.
  - An element of T_I is called *Task Instance*.
- For the partial mappings of the meta model ($\mathcal{P}$ refers to the power set):
  1. The mapping $rh : R \mapsto \mathcal{P}(R)$ is called *role hierarchy*. For $rh(r_s) = R_j$ we call r_s *senior role* and R_j the set of direct *junior roles*. The transitive closure rh^* defines the inheritance in the role-hierarchy such that $rh^*(r_s) = R_{j^*}$ includes all direct and transitive junior-roles that the senior-role r_s inherits from. The role-hierarchy is cycle-free, i.e. for each $r \in R : rh^*(r) \cap \{r\} = \emptyset$.
  2. The mapping $rsa : S \mapsto \mathcal{P}(R)$ is called *role-to-subject assignment*. For $rsa(s) = R_s$ we call s *subject* and $R_s \subseteq R$ the set of *roles assigned to this subject* (the set of roles owned by s).

     This assignment implies a mapping *role ownership* $rown : S \mapsto \mathcal{P}(R)$, such that for each subject s all direct and inherited roles are included, i.e. $rown(s) = \bigcup_{r \in rsa(s)} rh^*(r) \cup rsa(s)$. The mapping $rown^{-1} : R \mapsto \mathcal{P}(S)$ returns all subjects assigned to a role (directly or transitively via a role-hierarchy).
  3. The mapping $es : T_I \mapsto S$ is called *executing-subject mapping*. For $es(t) = s$ we call s the *executing subject* and t is called *executed task instance*.
  4. The mapping $er : T_I \mapsto R$ is called *executing-role mapping*. For $er(t) = r$ we call r the *executing role* and t is called *executed task instance*.
  5. The mapping $tra : R \mapsto \mathcal{P}(T_T)$ is called *task-to-role assignment*. For $tra(r) = T_r$ we call r *role* and $T_r \subseteq T_T$ is called the set of *tasks assigned to r*.

     This assignment implies a mapping *task ownership town* $: R \mapsto \mathcal{P}(T_T)$, such that for each role r the tasks inherited from its junior-roles are included, i.e. $town(r) = \bigcup_{r_{inh} \in rh^*(r)} tra(r_{inh}) \cup tra(r)$. The mapping $town^{-1} : T_T \mapsto \mathcal{P}(R)$ returns the set of roles a task is assigned to (directly or transitively via a role-hierarchy).
  6. 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*.
  7. The mapping $ar : S \mapsto R$ is called *active role mapping*. For $ar(s) = r$ we call s the subject and r the active-role of s.[1]
  8. Further, we allow the definition of subject-binding, role-binding, static mutual-exclusion, and dynamic mutual-exclusion constraints on task types. Related consistency requirements are specified in Strembeck and Mendling (2011):

     The mapping $sb : T_T \mapsto \mathcal{P}(T_T)$ is called ***subject-binding***. For $sb(t) = T_{sb}$, we call t the *subject binding task* and $T_{sb} \subseteq T_T$ the set of *subject-bound tasks*.

[1] We assume that each subject can (at the subject's discretion) activate the roles that are directly assigned to this subject as well as the junior-roles of its directly assigned roles (see, e.g., Ferraiolo et al. 1999; Sandhu et al. 1996)

The mapping $rb : T_T \mapsto \mathcal{P}(T_T)$ is called ***role-binding***. For $rb(t) = T_{rb}$, we call t the *role binding task* and $T_{rb} \subseteq T_T$ the set of *role-bound tasks*.

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 and $t_x \in T_{sme}$ *statically mutual exclusive tasks*.

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 and $t_x \in T_{dme}$ *dynamically mutual exclusive tasks*.

Definition 2 provides rules for the static correctness of process-related RBAC models to ensure the design-time consistency of the included elements and relationships.

Definition 2 Let $PRM = (E, Q, D)$ be a Process-related RBAC Model. *PRM* is said to be statically correct if the following requirements hold:

1. Tasks cannot be mutual exclusive to themselves:
   $\forall t_2 \in sme(t_1)$: $t_1 \neq t_2$ and
   $\forall t_2 \in dme(t_1)$: $t_1 \neq t_2$
2. Mutuality of mutual exclusion constraints:
   $\forall t_2 \in sme(t_1)$: $t_1 \in sme(t_2)$ and
   $\forall t_2 \in dme(t_1)$: $t_1 \in dme(t_2)$
3. Tasks cannot be bound to themselves:
   $\forall t_2 \in sb(t_1)$: $t_1 \neq t_2$ and
   $\forall t_2 \in rb(t_1)$: $t_1 \neq t_2$
4. Mutuality of binding constraints:
   $\forall t_2 \in sb(t_1)$: $t_1 \in sb(t_2)$ and
   $\forall t_2 \in rb(t_1)$: $t_1 \in rb(t_2)$
5. Tasks are either statically or dynamically mutual exclusive:
   $\forall t_2 \in sme(t_1)$: $t_2 \notin dme(t_1)$
6. Either SME constraint or binding constraint:
   $\forall t_2 \in sme(t_1)$: $t_2 \notin sb(t_1) \wedge t_2 \notin rb(t_1)$
7. Either DME constraint or subject-binding constraint:
   $\forall t_2 \in dme(t_1)$: $t_2 \notin sb(t_1)$
8. Consistency of task-ownership and SME:
   $\forall t_2 \in sme(t_1)$:
   $town^{-1}(t_2) \cap town^{-1}(t_1) = \emptyset$
9. Consistency of role-ownership and SME:
   $\forall t_2 \in sme(t_1), r_2 \in town^{-1}(t_2)$,
   $r_1 \in town^{-1}(t_1)$:
   $rown^{-1}(r_2) \cap rown^{-1}(r_1) = \emptyset$

Definition 3 provides the rules for dynamic correctness of a process-related RBAC model, i.e. the rules that can only be checked in the context of runtime process *instances*.

Definition 3 Let $PRM = (E, Q, D)$ be a Process-related RBAC Model and P_I its set of process instances. *PRM* is said to be dynamically correct if the following requirements hold:

1. In the same process instance, the executing subjects of SME tasks must be different: $\forall t_2 \in sme(t_1), pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) : es(t_x) \cap es(t_y) = \emptyset$
2. In the same process instance, the executing subjects of DME tasks must be different: $\forall t_2 \in dme(t_1), pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) : es(t_x) \cap es(t_y) = \emptyset$
3. In the same process instance, role-bound tasks must have the same executing-role: $\forall t_2 \in rb(t_1), pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) : er(t_x) = er(t_y)$
4. In the same process instance, subject-bound tasks must have the same executing-subject: $\forall t_2 \in sb(t_1), pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) : es(t_x) = es(t_y)$

Figure 2 shows a conceptual model (visualized as a class diagram) that includes the essential relations of process-related RBAC delegation models. However, while a graphical metamodel is a good means to visualize the connection of different artifacts, it cannot express all formal relations and invariants of these artifacts. Therefore, Definition 4 formally specifies the essential elements of a metamodel for process-related RBAC delegation models and the basic interrelations between these elements. In particular, we combine well-known concepts of several existing delegation models (see, e.g., Crampton and Khambhammettu 2008a; Hasebe et al. 2010; Schaad and Moffett 2002; Zhang et al. 2003b) and integrate them into a metamodel for process-related RBAC models.

Definition 4 (Process-Related RBAC Delegation Model) Let $PRDM = (E, Q, D, DL)$ be a Process-Related Delegation Model, where E refers to the pairwise disjoint sets of the metamodel, Q to mappings that establish relationships, D to binding and mutual-exclusion constraints, and DL to mappings for delegation policies.

The additional sets of the Process-Related RBAC Delegation Model are:

- An element of RR is called *Regular Role*. $RR \subseteq R$.
- An element of DR is called *Delegation Role*. $DR \subseteq R$
- An element of DRT is called *Temporary Delegation Role*. $DRT \subseteq DR$.
- An element of DT_T is called *Delegable Task Type*. $DT_T \subseteq T_T$.
- An element of DU_T is called *Duty Type*.
- An element of DU_I is called *Duty Instance*.
- An element of DDU_T is called *Delegable Duty Type*. $DDU_T \subseteq DU_T$.

For the mappings of the *Process-Related RBAC Model* (Q, D) see Definition 1. Below, we define the additional mappings for delegation: $DL = rrh \cup drh \cup creator \cup drpi \cup trra \cup trdel \cup dta \cup rrsa \cup drsa \cup dui \cup res \cup rer$ ($\mathcal{P}$ refers to the power set):

1. Roles R are partitioned into regular roles (RR) and delegation roles (DR). In RBAC, roles can be arranged in a role-hierarchy, where senior-roles inherit the permissions from their junior-roles. For example, in a bank, the senior-role "Bank manager" inherits all permissions from its junior-role "Bank clerk". Moreover, if "Bank clerk" itself has junior-roles, "Bank manager" transitively inherits all permissions from its transitive junior-roles. To avoid invalid permission inheritance, the regular role-hierarchy only consists of regular roles. This mapping replaces the role-hierarchy mapping rh in Definition 1.def:pcm-rh:

   The mapping $rrh : RR \mapsto \mathcal{P}(RR)$ is called ***regular role-hierarchy***. For $rrh(r_s) = RR_j$, we call $r_s \in RR$ *senior regular role* and $RR_j \subseteq RR$ the *set of direct junior regular roles*. The transitive closure rrh^* defines the inheritance in the role-hierarchy such that $rrh^*(r_s) = RR_{j*}$ includes all direct and transitive junior-roles that the senior-role r_s inherits from. The regular role-hierarchy is cycle-free, i.e. for each $r \in RR : rrh^*(r) \cap r = \emptyset$.

2. Delegation roles can be arranged in a delegation role-hierarchy. Note that each delegation role may have junior regular roles or junior delegation roles (see, e.g., Zhang et al. 2003b). However, delegation roles must not have senior regular roles to avoid invalid permission inheritance in the regular role hierarchy:

   The mapping $drh : DR \mapsto \mathcal{P}(R)$ is called ***delegation role-hierarchy***. For $drh(dr_s) = R_j$, we call $dr_s \in DR$ *senior delegation role* and $R_j \subseteq R$ the *set of direct junior-roles*. The transitive closure drh^* defines the inheritance in the role-hierarchy such that $drh^*(dr_s) = R_{j*}$ includes all direct and transitive junior-roles that the senior-role dr_s inherits from. The delegation role-hierarchy is cycle-free, i.e. for each $r \in R$: $drh^*(r) \cap r = \emptyset$.
3. Each subject can create an arbitrary number of delegation roles. Subsequently, the creator will act as the delegator of its delegation roles. For example, if subject "Alice" creates a delegation role "SummerIntern", she can delegate parts (or all) of her assigned tasks and duties to "SummerIntern":

   The mapping $creator(dr) : DR \mapsto S$ is called ***delegation role creator***. For $creator(dr) = s$, we call $dr \in DR$ *delegation role* and $s \in S$ the *creator of this delegation role*.
4. Each delegation role can be specified either for permanent or for temporary delegation. By default, a delegation role is permanent and is valid for all process types. In case of temporary delegation, a temporary delegation role has to be specified which is valid only for particular process instances. For example, the subject "Alice" wants to go on holidays and still has unfinished credit applications. Thus, she creates a delegation role "SummerIntern" that is only valid for the unfinished applications:

   The mapping $drpi : DRT \mapsto \mathcal{P}(P_I)$ is called ***temporary delegation role mapping***. For $drpi(drt) = P_{drt}$, we call $drt \in DRT$ *temporary delegation role* for $P_{drt} \subseteq P_I$ the *set of process instances*.
5. Task types are assigned to regular roles to define the permissions of the corresponding role. This mapping replaces the task-to-role assignment mapping *tra* in Definition 1.5:

   The mapping $trra : RR \mapsto \mathcal{P}(T_T)$ is called ***task-to-regular role assignment***. For $trra(r) = T_r$, we call $r \in RR$ *regular role* and $T_r \subseteq T_T$ is called the *set of tasks assigned to r*. The mapping $trra^{-1} : T_T \mapsto \mathcal{P}(RR)$ returns the set of regular roles a particular task is assigned to. Further, *trra* implies a mapping task ownership (*town*) which allows to determine all tasks that are assigned to a particular role (see Definition 1.5).
6. Task types can be defined as being delegable. Only delegable tasks can be assigned to delegation roles. Thus, a subject can delegate a task by assigning this task to a delegation role:

   The mapping $trdel : DR \mapsto \mathcal{P}(DT_T)$ is called *task-to-role delegation*. For $trdel(dr) = DT_{dr}$, we call $dr \in DR$ *delegation role* and $DT_{dr} \subseteq DT_T$ is called the *set of delegated tasks assigned to dr*. The mapping $trdel^{-1} : DT_T \mapsto \mathcal{P}(DR)$ returns the set of delegation roles a particular delegable task is assigned to.
7. A duty defines an action that must be performed by a certain subject in order to comply with legal or organizational regulations. In a business process context, each duty is associated with a task (Schefer and Strembeck 2011a):

   The mapping $dta : T_T \mapsto \mathcal{P}(DU_T)$ is called ***duty-to-task assignment***. For $dta(t) = DU_x$, we call $t \in T_T$ *task type* and $DU_x \subseteq DU_T$ is called the *set of duties assigned to this task type*.
8. Delegable tasks can only be delegated, if all associated duties are also delegable: $\forall t_x \in trdel(dr) : \forall du \in dta(t_x) : du \in DDU_T$.
9. Regular roles are assigned to subjects. Thereby, subjects acquire the rights to execute the corresponding tasks and duties. This mapping replaces the role-to-subject assignment mapping *rsa* in Definition 1.2:

   The mapping $rrsa : S \mapsto \mathcal{P}(RR)$ is called ***regular role-to-subject assignment***. For $rrsa(s) = RR_s$, we call $s \in S$ *subject* and $RR_s \in RR$ *the set of regular roles owned by s*. The mapping $rrsa^{-1} : RR \mapsto \mathcal{P}(S)$ returns all subjects assigned to a regular role. Further, *rrsa* implies a mapping role-ownership *rown*, which allows to determine all roles that are assigned to a particular subject.
10. Delegation roles are assigned to delegatees who are subsequently authorized and responsible to perform the corresponding delegated tasks and duties:

    The mapping $drsa : S \mapsto \mathcal{P}(DR)$ is called ***delegation role-to-subject assignment***. For $drsa(s) = DR_s$, we call $s \in S$ *delegatee* and $DR_s \in DR$ *the set of delegation roles owned by s*. The mapping $drsa^{-1} : DR \mapsto \mathcal{P}(S)$ returns all delegatees assigned to a delegation role.
11. For each task type, we can create an arbitrary number of respective task instances via the task instantiation mapping *ti* (see Definition 1.6). Similarly, each duty type is instantiated by a number of duty instances:

    The mapping $dui : (DU_T \times P_I) \mapsto \mathcal{P}(DU_I)$ is called ***duty instantiation***. For $dui(du_T, p_I) = DU_i$, we call $DU_i \subseteq DU_I$ set of *duty instances*, $du_T \in DU_T$ is called *duty type* and $p_I \in P_I$ is called *process instance*.
12. The executing-subject mapping *es* returns the subject executing a particular task instance (see Definition 1.3). The subject responsible for discharging a duty is called the responsible subject of this duty instance:

    The mapping $res : DU_I \mapsto S$ is called ***responsible-subject mapping***. For $res(du) = s$, we call $s \in S$ the *responsible subject* and $du \in DU_I$ is called *discharged duty instance*.
13. Within the same process instance, a subject executing a task is also responsible for discharging all associated duties:

    $\forall du \in dta(t_1), p_i \in P_I$:
    $\forall t_x \in ti(t_1, p_i), du_x \in dui(du, p_i)$:
    $es(t_x) = res(du_x)$

14. The role used to actually execute a certain task instance is called the executing-role *er* of this instance (see Definition 1.4). The role being responsible for actually discharging a certain duty instance is called the responsible-role of this instance:

    The mapping $rer : DU_I \mapsto R$ is called ***responsible-role mapping***. For $rer(du) = r$ we call $r \in R$ the *responsible role* and $du \in DU_I$ is called *discharged duty instance*.

3 Identifying and Resolving Delegation Conflicts

Due to the immanent complexity of process-related delegation models, sev-

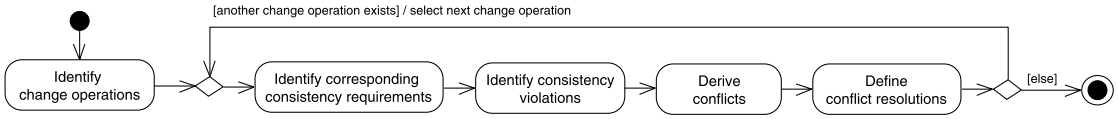

Fig. 3 General process of identifying conflicts and conflict resolutions

Table 1 Connection between the formal consistency requirements and the algorithms

Delegation conflict	Consistency requirement
Creator conflict	Definition 4.3
Delegable task conflict	Definition 4.6
Delegable duty conflict	Definition 4.8
Delegator task ownership (town) conflict	Definitions 4.6 and 4.9
Delegator role ownership (rown) conflict	Definitions 4.3 and 4.9
Task-assignment SME conflict	Definitions 1.8, 2.8, and 4.5
Role-assignment SME conflict	Definitions 1.8, 2.9, and 4.5
SB delegation conflict	Definitions 1.8, 2.3, 2.4, 3.2, and 4.5
RB delegation conflict	Definitions 1.8, 2.3, 2.4, 3.3, and 4.5
SB duty delegation conflict	Definitions 1.8, 4.5, 4.7, and 4.14
RB duty delegation conflict	Definitions 1.8, 4.5, 4.7, and 4.14
Self-delegation conflict	Definitions 4.1 and 4.2
Cyclic delegation conflict	Definition 4.2
Temporary delegation role conflict	Definition 4.4

eral types of potential conflicts may occur. In our delegation model, we identified 14 potential conflicts which need to be checked when delegating roles, tasks, or duties in order to prevent invalid task assignments. **Figure 3** shows an UML activity diagram that depicts the general process we applied to identify the different conflicts and conflict resolutions. In particular, we first identify the operations that change a (consistent) RBAC model (such as adding new task-to-role assignment relations or new constraints). Next, we identify the consistency requirements that have to be applied when using the respective change operation. Based on these consistency requirements we then identify potential consistency violations and derive corresponding conflicts. In the final step, we define resolution strategies for each of the conflicts. These steps are repeated for each change operation.

Thus, each of the conflicts we identified directly relates to the consistency requirements for process-related delegation models (see Sect. 2.2). Most of these consistency requirements were previously identified by other researchers (see, e.g., Botha and Eloff 2001; Crampton and Khambhammettu 2008a, 2008b; Strembeck and Mendling 2011; Zhang et al. 2003b). **Table 1** shows the connection between different delegation conflicts and the corresponding formal consistency requirement(s). A formal description of the resolution strategies is provided in Online-Appendix A.

Below, each potential delegation conflict and corresponding resolution strategies are discussed in detail. The conflicts can be detected by applying the algorithms presented in Sect. 4 and can be resolved by applying one of 21 predefined resolution strategies. These resolutions prevent inconsistent delegation assignments or runtime allocations. For most of the conflicts, several alternative resolutions are applicable to resolve the conflict. The decision on which of the possible resolutions should be applied usually involves human judgment as it highly depends on the respective organizational context and on the desired RBAC configuration. In Online-Appendix A, all conflict resolutions are defined with respect to the formal definitions of process-related RBAC delegation models (see Sect. 2 and Strembeck and Mendling 2010, 2011).

3.1 Identifying and Resolving Delegability Conflicts

Creator conflict: A creator conflict exists if a subject tries to delegate to a delegation role which he/she has not created. Only the creator of a delegation role can delegate to it and assign delegatees (see Definition 4.3). For example, in **Fig. 4a** subject s_1 tries to delegate task t_x to delegation role dr_y. t_x is delegable which is visualized in **Fig. 4a** by an arrow attached to the task-symbol including the letter D. However, s_1 is not the creator of dr_y.

Resolutions to creator conflicts: To resolve this conflict, we can delegate the task to one of the delegator's delegation roles (see Resolution 1 in Online-Appendix A). As an alternative, the conflict can be resolved by first removing the respective delegation role. Then the delegator can create a new delegation role with the same name and is now able to delegate to it (see Resolution 2 in Online-Appendix A). In **Fig. 4a**, s_1 can delegate task t_x to one of its delegation roles dr_z (see **Fig. 4a** and Resolution 1). Alternatively, dr_y is removed. Subsequently, s_1 can create a new delegation role named dr_y and delegates task t_x to dr_y.

Delegable task conflict: A delegable task conflict arises if a subject tries to delegate a task which is not defined as delegable. In **Fig. 4b**, task t_x cannot be delegated to delegation role dr_y, because t_x is not delegable. Similarly, this conflict can occur if a delegator tries to delegate a role and one of the tasks assigned to this role is not delegable.

Resolutions to delegable task conflict: This conflict can only be resolved by defining the task(s) that should be delegated as delegable (see **Fig. 4b** and Resolution 3).

Delegable duty conflict: A delegable duty conflict exists if a subject tries to delegate a task which is associated with a non-delegable duty. Duties always need to be discharged by the subject executing the corresponding task. Thus, if a task is delegated, the corresponding duty also needs to be delegable. This conflict can also occur if a delegator tries to delegate a role and one of the tasks assigned to this role is associated to a non-delegable duty. In **Fig. 4c**, task t_x can not be delegated to delegation role dr_y, because the duty du_x associated to t_x is not defined as delegable.

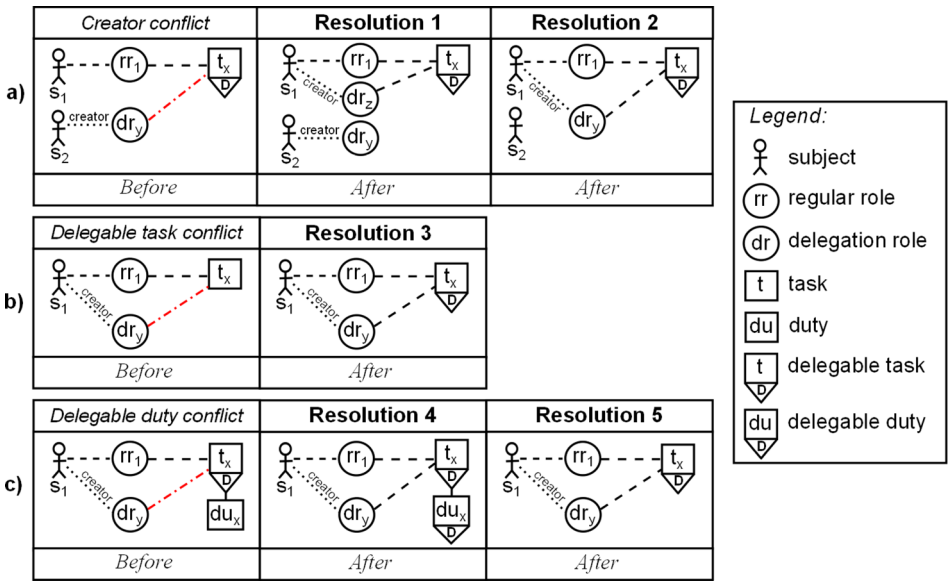

Fig. 4 Resolving creator (**a**), delegable task (**b**), and delegable duty (**c**) conflicts

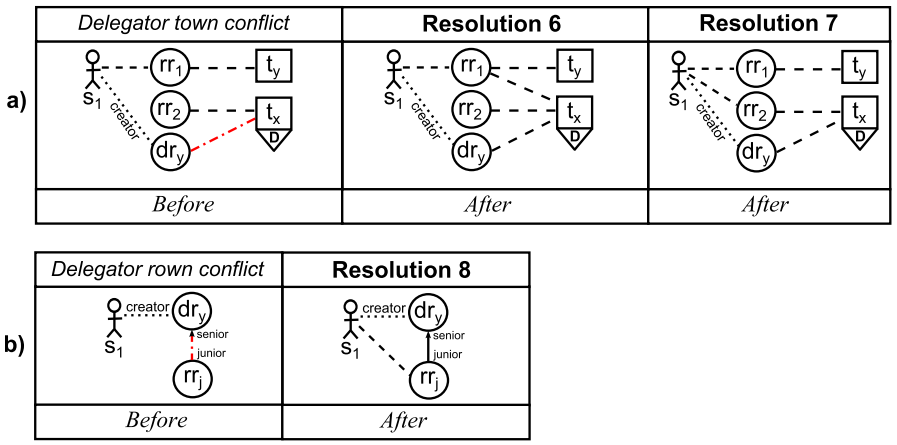

Fig. 5 Resolving delegator town (**a**) and delegator rown (**b**) conflicts

Resolutions to delegable duty conflict: This conflict can be resolved by defining the relevant duties as delegable (see Resolution 4). Alternatively, the conflicting duties could be removed to resolve the conflict (see Resolution 5). However, this resolution will rarely be applicable in real-world scenarios and is thus only presented for the sake of completeness. In **Fig. 4c**, t_x can be delegated if the associated duty du_x is defined as delegable or if du_x is deleted.

3.2 Identifying and Resolving Ownership Conflicts

Delegator task ownership (town) conflict: A delegator town conflict occurs if a subject tries to delegate a task which he/she is not assigned to via its regular role-ownership assignments. A subject can only delegate tasks and roles which he/she owns directly or transitively. In **Fig. 5a**, subject s_1 tries to delegate task t_x to its delegation role dr_y. However, none of the regular roles owned by s_1 is assigned to t_x.

Resolutions to delegator town conflicts: The conflict can be resolved by assigning the task to one of the regular roles owned by the delegator (see Resolution 6). Alternatively, the delegator can be assigned to one of the regular roles owning the corresponding task (see Resolution 7). In **Fig. 5a**, s_1 can delegate t_x after assigning t_x to the regular role rr_1 which is owned by s_1. Alternatively, regular role rr_2 owning t_x can be assigned to the delegator s_1.

Delegator role ownership (rown) conflict: A delegator rown conflict occurs if a subject tries to delegate a role which he/she is not assigned to. In **Fig. 5b**, subject s_1 tries to delegate the regular role rr_j to its delegation role dr_y by assigning rr_j as junior-role of dr_y. However, s_1 is not assigned to rr_j.

Resolutions to delegator rown conflicts: The conflict can be resolved by assigning the delegator (directly or transitively) to the role he/she tries to delegate. In **Fig. 5b**, s_1 is assigned to rr_j in order to be able to delegate it to dr_y (see Resolution 8).

3.3 Identifying and Resolving SME Conflicts

Task-assignment SME conflict: A task-assignment SME conflict may occur if a new task-to-role or role-to-role delegation would result in the assignment of two SME tasks to the same role. **Figure 6a** depicts an example where a delegation role dr_y owns a task t_y which is defined as SME to another task t_x. Thus, delegating t_x to dr_y would result in a task-assignment conflict.

Resolutions to task-assignment SME conflicts: To avoid the *task-assignment SME conflict* in **Fig. 6a**, the conflicting SME constraint between the two task types can be removed or changed into a DME constraint (see Resolutions 9

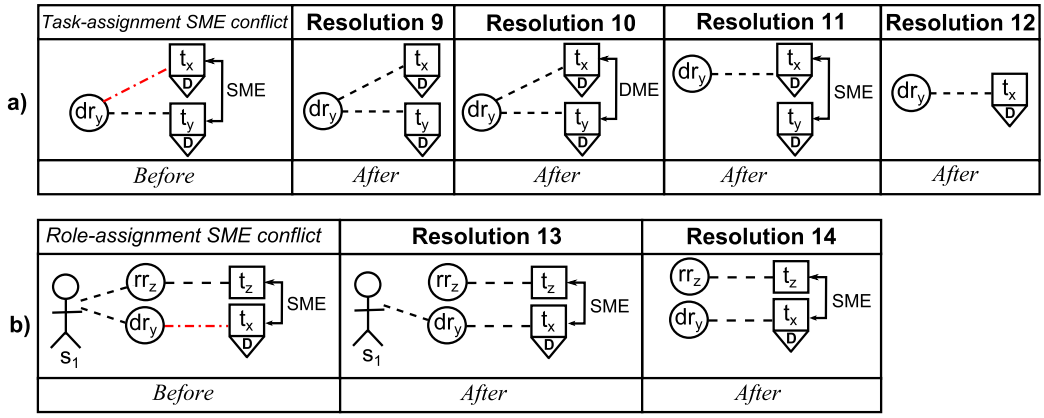

Fig. 6 Resolving task- (**a**) and role-assignment (**b**) SME conflicts

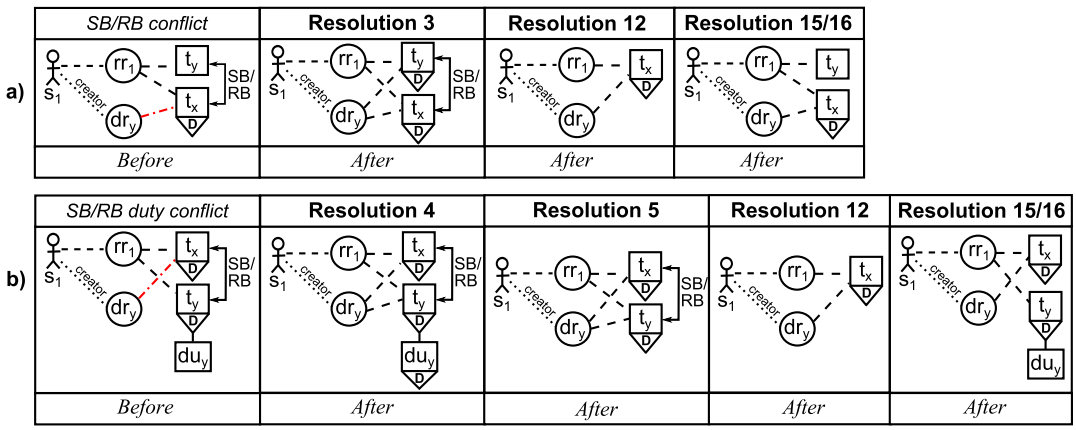

Fig. 7 Resolving SB/RB (**a**) and SB/RB duty (**b**) delegation conflicts

and 10). Alternatively, task t_y can be revoked from dr_y, or the conflicting task t_y can be deleted (see Resolutions 11 and 12). Note that some of these resolutions, such as removing a constraint or a task, will rarely be applicable in real-world scenarios and are thus primarily presented for the sake of completeness.

Role-assignment SME conflict: A role-assignment SME conflict arises if a task-to-role or role-to-role delegation would result in the assignment of two SME tasks to the same subject. As a consequence, the delegatee would be authorized to perform two SME tasks. **Figure 6b** shows an example, where the delegation of task t_x to the delegation role dr_y would result in a role-assignment conflict because delegatee s_1 would then be authorized to perform the two SME tasks t_z and t_x. Similarly, when delegating a role to a delegation role or when assigning a delegatee to a delegation role, we need to check for role-assignment conflicts.

Resolutions to role-assignment SME conflicts: To avoid a role-assignment SME conflict, the same resolutions as for task-assignment conflicts can be applied (see Resolutions 9–12). In addition, Resolution 13 can be applied by revoking the conflicting assignment between regular role rr_z and subject s_1 (see **Fig. 6b**). Moreover, the conflict can (theoretically) be resolved by removing the conflicting subject s_1 which is assigned to the two SME tasks (see Resolution 14).

3.4 Identifying and Resolving Binding Conflicts

SB delegation conflict: A SB delegation conflict exists if a subject tries to delegate a task which has a subject-binding to one or more non-delegable task(s). However, subject-bound tasks always have to be performed by the same subject. Thus, if a task is delegated, all subject-bound tasks also need to be assigned to the same delegation role. Otherwise, the SB constraint cannot be fulfilled. In **Fig. 7a**, a SB constraint is defined on t_x and t_y. Therefore, the subject performing t_x also has to perform t_y. When delegating t_x to dr_y a SB conflict arises, because t_y is not defined as delegable. However, to fulfill the SB constraint, both tasks need to be delegated to dr_y.

Resolutions to SB delegation conflicts: This conflict can be resolved by defining all subject-bound tasks as delegable (see Resolution 3). Alternatively, the conflicting task or the SB constraint can be removed (see Resolutions 12 and 15). In **Fig. 7a**, t_x can be delegated, if t_y is defined as delegable. Subsequently, both tasks can be delegated to dr_y. The delegation is also possible, if t_y or the SB constraint on t_x and t_y is removed.

RB delegation conflict: A RB delegation conflict exists if a subject tries to delegate a task which has a role-binding to one or more non-delegable task(s). However, role-bound tasks always have to be performed by members of the same role. Thus, if a task is delegated, all role-bound tasks also need to be assigned to the same delegation role. Otherwise, the RB constraint cannot be fulfilled. In **Fig. 7a**, a RB

constraint is defined on t_x and t_y. Therefore, t_x and t_y always have to be assigned to the same role. When delegating t_x to dr_y a RB conflict arises, because t_y is not defined as delegable. However, to fulfill the RB constraint, both tasks need to be delegated to dr_y.

Resolutions to RB delegation conflicts: This conflict can be resolved by defining all role-bound tasks as delegable (see Resolution 3). Alternatively, the conflicting task or the RB constraint can be removed (see Resolutions 12 and 16). In **Fig. 7a**, t_x can be delegated, if t_y is defined as delegable. Subsequently, both tasks can be delegated to dr_y. The delegation is also possible, if t_y or the RB constraint on t_x and t_y is removed.

SB duty delegation conflict: In case a subject tries to delegate a task which has a subject-binding to other tasks, a SB duty delegation conflict arises, if one of the subject-bound tasks is associated with a non-delegable duty. In this case, the corresponding subject-bound task cannot be delegated. However, to fulfill the SB constraint, all subject-bound tasks need to be delegated. In **Fig. 7b**, a SB constraint is defined on t_x and t_y. Moreover, t_y is associated with a duty du_y. If subject s_1 tries to delegate t_x to dr_y, it also has to delegate all subject-bound tasks and associated duties. In this example, du_y is not delegable. Thus, a SB duty conflict arises.

Resolutions to SB duty delegation conflicts: The conflict can be resolved by defining the respective duty du_y as delegable (see Resolution 4). Alternatively, the conflicting duty du_y can be deleted, the subject-bound task t_y being associated with du_y can be deleted, or the SB constraint can be removed (see Resolutions 5, 12, and 15).

RB duty delegation conflict: In case a subject tries to delegate a task which has a role-binding to other tasks, a RB duty conflict arises, if one of the role-bound tasks is associated with a non-delegable duty. In this case, the corresponding role-bound task cannot be delegated. However, to fulfill the RB constraint, all role-bound tasks need to be delegated to the same delegation role. If subject s_1 tries to delegate t_x to dr_y in **Fig. 7b**, it also has to delegate all role-bound tasks and associated duties to dr_y. In this example, du_y is not delegable. Thus, a RB duty conflict arises.

Resolutions to RB duty delegation conflicts: The conflict can be resolved by defining the respective duty du_y as delegable (see Resolution 4). Alternatively, the conflicting duty du_y can be deleted, the role-bound task t_y being associated with du_y can be deleted, or the RB constraint can be removed (see Resolutions 5, 12, and 16).

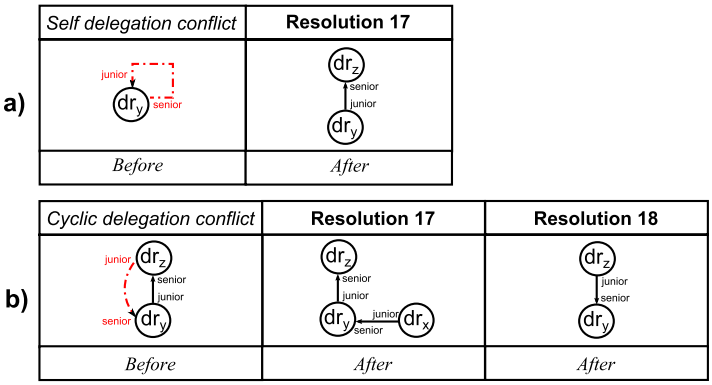

Fig. 8 Resolving self-delegation (**a**) and cyclic delegation (**b**) conflicts

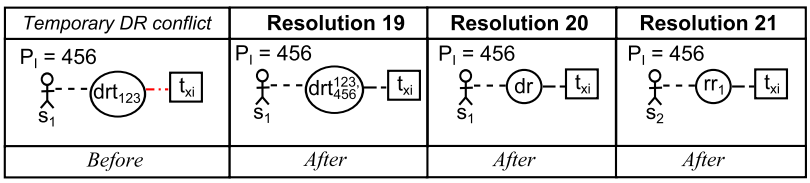

Fig. 9 Resolving runtime conflicts

3.5 Identifying and Resolving Inheritance Conflicts

Self-delegation conflict: A self-delegation conflict may arise when delegating a role to itself. In general, a role cannot be its own junior-role (see **Fig. 8a** and Strembeck and Mendling 2010, 2011).

Resolution to self-delegation conflicts: This conflict can be resolved by selecting another junior- or senior-role so that the inheritance relation is defined between two different roles (see **Fig. 8a** and Resolution 17).

Cyclic delegation conflict: A cyclic delegation conflict results from delegating a role to a delegation role which is already defined as senior-role of this delegation role. In particular, a role-hierarchy must not include a cycle because all roles within such a cyclic inheritance relation would own the same permissions which would again render the respective part of the role-hierarchy redundant (see **Fig. 8b** and Strembeck and Mendling 2010, 2011).

Resolutions to cyclic delegation conflicts: This conflict can be resolved by delegating another role to this delegation role which is not already part of the same role-hierarchy (see Resolution 17). In **Fig. 8b**, Resolution 17 is applied by defining a new inheritance relation between dr_x and dr_y while keeping the existing inheritance relation between dr_y and dr_z. Moreover, the existing inheritance relation between dr_y and dr_z can be removed before defining the inverse inheritance relation with dr_z as junior-role of dr_y (see Resolution 18).

3.6 Identifying and Resolving Runtime Conflicts

In addition to the runtime conflicts presented in Schefer et al. (2011) and Strembeck and Mendling (2010), one additional conflict in the context of temporary delegation roles may occur.

Temporary delegation role conflict: A temporary delegation role conflict occurs if the selected subject is not allowed to execute a certain task instance because the temporary delegation role is not valid for the corresponding process instance. Each temporary delegation role is only valid for particular process instances. Thus, a delegatee assigned to a temporary delegation role is authorized to execute all delegated tasks only within these process instances. In **Fig. 9**, subject s_1 is assigned to the temporary delegation role *drt*, and *drt* is only valid for the process instance 123. However, the actual process instance

Algorithm 1 Check if it is allowed to delegate a particular task type to a particular delegation role

Name: isT2RDelegationAllowed
Input: $task_x \in T_T$, $drole_y \in DR$, $delegator \in S$
1: **if** $delegator \neq creator(drole_y)$ **then return** creatorConflict
2: **if** $task_x \notin DT_T$ **then return** delegableTaskConflict
3: **if** $\exists\ duty_x \in dta(task_x) \mid duty_x \notin DDU_T$ **then return** delegableDutyConflict
4: **if** $\nexists\ r \in rown(delegator) \mid task_x \in town(rr) \ \wedge\ r \in RR$ **then return** delegatorTownConflict
5: **if** $\exists\ task_y \in town(drole_y) \mid task_y \in sme(task_x)$ **then return** taskAssignmentSMEConflict
6: **if** $\exists\ role_z \in allSeniorRoles(drole_y) \mid task_z \in town(role_z) \wedge$
7: $task_z \in sme(task_x)$ **then return** taskAssignmentSMEConflict
8: **if** $\exists\ s \in S \mid drole_y \in rown(s) \wedge role_z \in rown(s) \wedge$
9: $task_z \in town(role_z) \wedge task_z \in sme(task_x)$ **then return** roleAssignmentSMEConflict
10: **if** $\exists\ task_y \in sb(task_x) \mid task_y \notin DT_T$ **then return** SBDelegationConflict
11: **if** $\exists\ task_y \in rb(task_x) \mid task_y \notin DT_T$ **then return** RBDelegationConflict
12: **if** $\exists\ task_y \in sb(task_x) \mid duty_y \in dta(task_y) \wedge$
13: $duty_y \notin DDU_T$ **then return** SBDutyDelegationConflict
14: **if** $\exists\ task_y \in rb(task_x) \mid duty_y \in dta(task_y) \wedge$
15: $duty_y \notin DDU_T$ **then return** RBDutyDelegationConflict
16: **return** true

Algorithm 2 Check if it is allowed to delegate a particular role to a delegation role

Name: isR2RDelegationAllowed
Input: $junior \in R$, $senior \in DR$, $delegator \in S$
1: **if** $delegator \neq creator(senior)$ **then return** creatorConflict
2: **if** $junior \notin rown(delegator)$ **then return** delegatorRownConflict
3: **if** $junior == senior$ **then return** selfDelegationConflict
4: **if** $\exists\ task_x \in town(junior) \mid task_x \notin DT$ **then return** delegableTaskConflict
5: **if** $\exists\ task_x \in town(junior) \mid duty_x \in dta(task_x) \wedge$
6: $duty_x \notin DDU_T$ **then return** delegableDutyConflict
7: **if** $junior \in DR$ **then** $\exists\ r \in rown(delegator) \mid task_x \in town(junior) \wedge$
8: $task_x \in town(r) \ \wedge\ r \in RR$ **else return** delegatorTownConflict
9: **if** $senior \in drh^*(junior)$ **then return** cyclicDelegationConflict
10: **if** $\exists\ task_j \in town(junior) \mid task_s \in town(senior) \wedge$
11: $task_j \in sme(task_s)$ **then return** taskAssignmentSMEConflict
12: **if** $\exists\ role_x \in allSeniorRoles(senior) \mid task_x \in town(role_x) \wedge$
13: $task_j \in town(junior) \ \wedge task_x \in sme(task_j)$
14: **then return** taskAssignmentSMEConflict
15: **if** $\exists\ s \in S \mid senior \in rown(s) \wedge role_x \in rown(s) \wedge$
16: $task_x \in town(role_x) \wedge task_j \in town(junior) \wedge task_x \in sme(task_j)$
17: **then return** roleAssignmentSMEConflict
18: **if** $\exists\ task_x \in town(junior) \mid task_y \in sb(task_x) \wedge$
19: $task_y \notin DT_T$ **then return** SBDelegationConflict
20: **if** $\exists\ task_x \in town(junior) \mid task_y \in sb(task_x) \wedge$
21: $\exists\ duty_y \in dta(task_y) \wedge duty_y \notin DDU_T$ **then return** SBDutyDelegationConflict
22: **return** true

is 456. Thus, s_1 is not allowed to execute the delegated tasks in this process instance.

Resolving temporary delegation role conflicts: The conflict can be resolved by adding the corresponding process instance to the process instances a temporary delegation role is valid for (see Resolution 19). Another solution is to change a temporary delegation role into a permanent delegation role (see Resolution 20). Subsequently, delegatees are authorized to perform all instances of the delegated tasks. In **Fig. 9** Resolution 19 is applied by defining that delegation role *drt* now is also valid for the process instance 456. As an alternative, *drt* can be changed into a permanent delegation role. Alternatively, one can allocate an executing subject that actually owns the permission to perform the respective task (see Resolution 21). Subject s_2 is authorized to perform instances of t_x. Thus, we can allocate s_2 as executing subject for this particular task instance.

4 Algorithms for Detecting Delegation Conflicts

In this Section, we provide algorithms to detect the delegation conflicts introduced in Sect. 3 in process-related RBAC models at design-time and runtime. To support a systematic conflict handling, we suggest to perform the following three steps. First, one needs to detect a conflict when delegating tasks, duties, or roles or when assigning a subject to a delegation

Algorithm 3 Check if it is allowed to assign a particular delegation role to a certain delegatee

Name: isR2SDelegationAllowed
Input: $drole_x \in DR$, $delegatee$, $delegator \in S$
1: **if** $delegator \neq creator(drole_x)$ **then return** creatorConflict
2: **if** $\exists\ role_y \in rown(delegatee) \mid task_y \in town(role_y) \wedge$
3: $task_x \in town(drole_x) \wedge task_y \in sme(task_x)$ **then return** roleAssignmentSMEConflict
4: **return** true

Algorithm 4 Check if a particular task instance that is executed in a certain process instance can be allocated to a specific delegatee

Name: isDelegateeAllocationAllowed
Input: $drole \in DRT$, $delegatee \in S$, $task_{type} \in T_T$, $process_{type} \in P_T$,
$process_{instance} \in pi(process_{type})$, $task_{instance} \in ti(task_{type}, process_{instance})$
1: **if** $\exists\ instance_y \in ti(type_y, process_{instance}) \mid ar(delegatee) = drole \wedge$
2: $process_{instance} \notin drpi(drole)$ **then return** temporaryDelegationRoleConflict
3: **return** true

role. After detecting a conflict, we have to decide on how to resolve this conflict, i.e., decide which resolution strategy to apply. After applying the resolution strategy, the requested delegation can be performed.

Algorithms 1–3 check the design-time consistency of a process-related RBAC delegation model before defining a new task-to-role, role-to-role, or role-to-subject delegation relation. Algorithm 4 checks the runtime consistency of a process-related RBAC delegation model. In particular, it checks, if a temporary delegation role authorizes a subject to perform a delegated task in a certain process instance. These algorithms complement the set of algorithms presented in Schefer et al. (2011) and Strembeck and Mendling (2010) which detect potential conflicts of process-related RBAC models not related to delegation. If a delegation conflict is detected, the algorithms presented below return the name of the respective conflict.

5 A UML Extension for Modeling Process-Related Delegation Models

An organization's business processes and software systems are often modeled via graphical modeling languages. The Unified Modeling Language (UML; OMG 2011b) offers a comprehensive and well-defined modeling framework and is the de facto standard for modeling and specifying information systems. The UML's main intention is to capture modeling artifacts throughout the whole development lifecycle with the same modeling language OMG (2011b). The UML metamodel builds upon the OMG Meta Object Facility (MOF OMG 2011a) and defines the abstract syntax of all UML diagram types. Modeling support for the delegation of roles, tasks, and duties via a standard notation can help to bridge the communication gap between software engineers, security experts, experts of the application domain, and other stakeholders (see, e.g., Mouratidis and Jürjens 2010). Several approaches already exist that consider different kinds of security properties in a UML context (see, e.g., Basin et al. 2006; Jürjens 2005; Rodriguez et al. 2006; Rodriguez and de Guzman 2007).

UML2 activity models offer a process modeling language that allows to model the control flows and object flows between different actions. The main element of an activity diagram is an activity. Its behavior is defined by a decomposition into different actions. A UML2 activity thus models a process while the actions that are included in the activity can be used to model tasks (for details on UML2 activity models, see OMG 2011b). However, sometimes UML diagrams can not provide all relevant aspects of a specification. Therefore, there is a need to define additional constraints about the modeling elements. The Object Constraint Language (OCL) provides a formal language that enables the definition of constraints on UML models (OMG 2014). We apply the OCL to define additional delegation-specific constraints for our UML extension. In particular, the OCL invariants defined in Sect. 5.2 ensure the consistency and correctness of UML models using our new modeling elements.

The UML standard basically provides two options to adapt its metamodel to a specific area of application (OMG 2011b): (a) defining a UML profile specification using stereotypes, tag definitions, and constraints. A UML profile must not change the UML metamodel but can only extend existing UML meta-classes for special domains. Thus, UML profiles are not a first-class extension mechanism (see OMG 2011b, p. 660); (b) extending the UML metamodel, which allows for the definition of new elements with customized semantics.

In this paper, we apply the second option (extending the UML metamodel) because the newly defined modeling elements for process-related delegation require new semantics which are not available in the UML metamodel. Thus, we introduce the *BusinessActivityDelegations* extension for the UML metamodel which is designed for modeling the delegation of roles, tasks, and duties based on the formal metamodel definitions presented in Sect. 2. For this purpose, we extend the *BusinessActivities* package (Strembeck and Mendling 2011), which provides UML modeling support for process-related RBAC models. We also implemented the extended metamodel as a delegation extension to the BusinessActivity library and runtime engine (see Sect. 7).

5.1 UML Metamodel Overview

Based on the formal CIM-layer metamodel for process-related RBAC delegation models presented in Sect. 2, **Fig. 10** presents our corresponding extension to the UML at the PIM layer. In our UML extension, a *BusinessActivity* is a special UML Activity (see **Fig. 10**). In addition to our newly introduced elements, it can

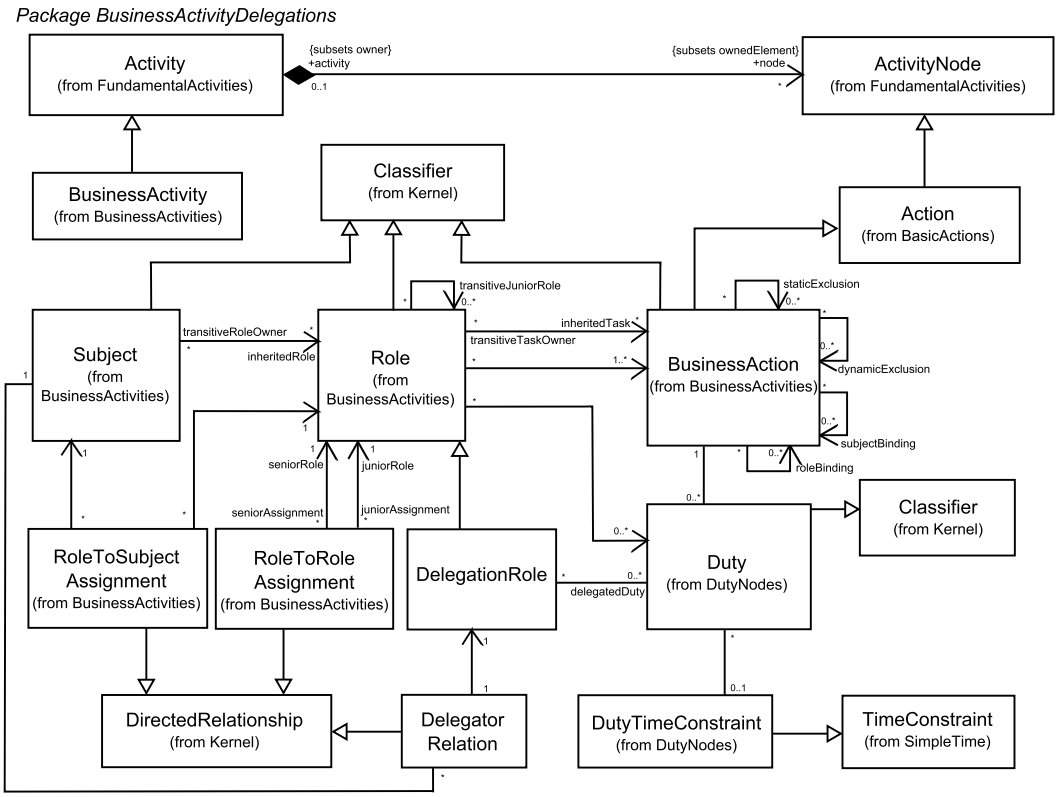

Fig. 10 UML metamodel extension *BusinessActivityDelegations* for Activity diagrams

Fig. 11 Visualizing (**a**) delegation roles and (**b**) delegator relations

include all elements available for ordinary UML Activities. A *BusinessAction* corresponds to a task in a business processes and comprises all permissions to perform the respective task (see Strembeck and Mendling 2011 for further details on BusinessActivities and BusinessActions). A *Duty* is a special UML Classifier (see **Fig. 10**) and is used to model that an action must be performed by a certain Subject (Schefer and Strembeck 2011a). The link between Duties and BusinessActions assures that a Subject being assigned to a Duty also receives all permissions to perform these Duties. *Roles* and *Subjects* are specialized UML Classifiers Strembeck and Mendling (2011) which are linked to BusinessActions and Duties (see **Fig. 10**). Furthermore, a Duty may be linked to a *DutyTimeConstraint* which is a specialized UML TimeConstraint. If a DutyTimeConstraint has expired, a *Compensation Action* is triggered which is defined as stereotype of the Action metaclass (see Schefer and Strembeck 2011a for further details).

A *DelegationRole* is a special type of Role which is assigned to a set of delegated Roles, BusinessActions, and/or Duties (see **Fig. 10**). A *DelegatorRelation* is a special UML DirectedRelationship and indicates that a certain Subject acts as a delegator for a certain DelegationRole. Only delegators may delegate Roles, BusinessActions, or Duties to DelegationRoles (see OCL Constraint 1 in Sect. 5.2). **Figure 11** illustrates presentation options to visualize delegation roles and delegator relations. Note that these relations are formally defined through our UML metamodel extension and therefore exist independent of their actual graphical representation.

DelegationRoles are assigned to delegatees which thereby are authorized to perform the respective BusinessActions and Duties (see OCL Constraint 2). A delegator can *delegate a Role* by defining this Role as junior-role to one of his or her DelegationRoles. All BusinessActions and Duties assigned to this Role need to be delegable (see below). Note that DelegationRoles must not have senior regular Roles to avoid invalid permission inheritance (see OCL Constraint 3). For *delegating a BusinessAction*, the delegator assigns the BusinessAction to the respective DelegationRole. Only if a BusinessAction is delegable, it can be delegated to a DelegationRole (see OCL Constraints 4 and 6). To realize *delegation of Duties* in UML models, a Duty also needs to be defined as being delegable (see OCL Constraints 5, 7, and 9). After assigning a delegatee, the delegator loses his obligation to perform this Duty. Yet, a review duty can be defined (Schaad and Moffett 2002) which obliges the delegator to control the proper enforcement of his delegated Duties (see OCL Constraints 8 and 9).

To consider the aspect of permanence in delegation (Barka and Sandhu 2000b), our DelegationRoles can either be defined for temporary or for permanent delegation, i.e., for a single or for all instances of a business process (see OCL Constraints 10 and 11). Furthermore, we support single- and multi-step delegation for BusinessActions and Duties. Single-step delegation means that a delegated BusinessAction or Duty can not be further delegated by the delegatee (Barka and Sandhu 2000b). This is achieved by defining an attribute called *isDelegated* for each BusinessAction and for

OCL Constraint 1 The delegator of a Duty, a BusinessAction, or a Role needs to be the Subject who is directly assigned to the respective delegation unit.

```
context Subject
inv: self.delegatorRelation->forAll(d |
       d.delegationRole.delegatedDuty->forAll(dd |
         dd.role->exists(r |
           r.roleToSubjectAssignment->exists(rsa |
             rsa.subject = self ))))
inv: self.delegatorRelation->forAll(d |
       d.delegationRole.businessaction->forAll(ba |
         ba.role->exists(r |
           r.roleToSubjectAssignment->exists(rsa |
             rsa.subject = self ))
         or
         ba.transitiveTaskOwner->exists(to |
           to.roleToSubjectAssignment->exists(torsa |
             torsa.subject = self ))))
inv: self.delegatorRelation->forAll(d |
       d.delegationRole.seniorAssignment->notEmpty() implies
         d.delegationRole.seniorAssignment->forAll(sa |
           if sa.juniorRole.oclIsTypeOf(Role) then
               sa.juniorRole.roleToSubjectAssignment->exists(rsa | rsa.subject = self)
               or
               sa.juniorRole.transitiveRoleOwner->exists(tro | tro = self )
           else true endif ))
```

OCL Constraint 6 Each BusinessAction defines an attribute called "isDelegated" stating if a special BusinessAction has already been delegated or not. If it has already been delegated, it cannot be delegated further (single-step delegation, see Barka and Sandhu 2000b).

```
context BusinessAction
inv: self.instanceSpecification->forAll(i | i.slot->exists(s | s.definingFeature.name = isDelegated))
inv: self.instanceSpecification->forAll(i |
  let baid : Slot = i.slot->select(s | s.definingFeature.name = isDelegated) in
    let bdgb : Slot = i.slot->select(d | d.definingFeature.name = delegable) in
      if baid.value = true then bdgb.value = false
      else true endif )
```

OCL Constraint 11 If a DelegationRole is intended for temporary delegation only (isTemporary=true), it defines an attribute called "relatedProcessInstance" to ensure that the respective DelegationRole can only be used in the defined process instance.

```
context DelegationRole
inv: self.instanceSpecification->forAll(i | i.slot->exists(s | s.definingFeature.name = relatedProcessInstance ))
inv: self.instanceSpecification->forAll(i |
   self.businessAction->exists(ba |
     ba.activity.instanceSpecification->exists(a |
       let drit : Slot = i.slot->select(si | si.definingFeature.name = isTemporary) in
         if drit.value = true then
            let rpi : Slot = i.slot->select(so | so.definingFeature.name = relatedProcessInstance) in
               let apid : Slot = a.slot->select(sa | sa.definingFeature.name = processID) in
                  rpi.value = apid.value
         else true endif )))
```

each Duty. The isDelegated attribute is set to true as soon as the respective BusinessAction or Duty has been delegated. If a BusinessAction's or a Duty's isDelegated-attribute is set to true, its *delegable*-attribute is set to false (see OCL Constraints 6 and 7). One advantage of single-step delegation is that the delegator who might be obliged to supervise the enforcement keeps control about who is responsible for actually performing the delegated BusinessAction or discharging the delegated Duty. However, multi-step delegation can easily be activated by using OCL Constraints 15 and 16 instead.

5.2 OCL Constraints

Often a structural UML model cannot capture all types of constraints which are relevant for describing a target domain. Thus, additional constraints can be defined, for example, by using a constraint expression language, such as the OCL (OMG 2014). In this paper, we use OCL invariants to define the semantics by encoding delegation-specific constraints. For the sake of readability, this Section only shows three example OCL invariants. The complete list of OCL invariants for the Business Activity Delegations extension can be found in Online-Appendix B. In addition, **Table 2** gives an overview of how each of the definitions from Sect. 2.2 is mapped to our UML extension for Business Activity Delegation Models.

5.3 Example for Business Activity Delegation Models

In **Fig. 12**, the credit application process from **Fig. 1** is extended by including the

Table 2 Consistency between generic metamodel and UML extension

Generic Definition	Covered through
Definition 4.1: $rrh : RR \mapsto \mathcal{P}(RR)$	Metamodel extension: RoleToRoleAssignment relation (see **Fig. 10** and Strembeck and Mendling 2011)
Definition 4.2: $drh : DR \mapsto \mathcal{P}(R)$	Metamodel extension: RoleToRoleAssignment relation (see **Fig. 10**) and OCL Constraint 3
Definition 4.3: $creator(dr) : DR \mapsto S$	Metamodel extension: DelegatorRelation metaclass (see **Fig. 10**) and OCL Constraint 1
Definition 4.4: $drpi : DRT \mapsto \mathcal{P}(P_I)$	OCL Constraints 10 and 11
Definition 4.5: $trra : RR \mapsto \mathcal{P}(T_T)$	Metamodel extension: Association between Role and BusinessAction (see **Fig. 10** and Strembeck and Mendling 2011)
Definition 4.6: $trdel : DR \mapsto \mathcal{P}(DT_T)$	Metamodel extension: Association between Role and BusinessAction (see **Fig. 10**) and OCL Constraints 4, 6, and 15
Definition 4.7: $dta : T_T \mapsto \mathcal{P}(DU_T)$	Metamodel extension: Association between Duty and BusinessAction (see **Fig. 10**)
Definition 4.8: $\forall t_x \in trdel(dr) : \forall du \in dta(t_x) : du \in DDU_T$	OCL Constraints 5, 7, 8, 9, and 16
Definition 4.9: $rrsa : S \mapsto \mathcal{P}(RR)$	Metamodel extension: RoleToSubjectAssignment relation (see **Fig. 10** and Strembeck and Mendling 2011)
Definition 4.10: $drsa : S \mapsto \mathcal{P}(DR)$	Metamodel extension: RoleToSubjectAssignment relation (see **Fig. 10**) and OCL Constraint 2
Definition 4.11: $dui : (DU_T \times P_I) \mapsto \mathcal{P}(DU_I)$	Implicitly defined via our metamodel extension and the specification of UML activity models (see **Fig. 10**) and OMG (2011b), Schefer and Strembeck (2011a)
Definition 4.12: $res : DU_I \mapsto S$	OCL Constraint 12
Definition 4.13: $\forall du \in dta(t_1), p_i \in P_I : \forall t_x \in ti(t_1, p_i), du_x \in dui(du, p_i) : es(t_x) = res(du_x)$	OCL Constraint 13
Definition 4.14: $rer : DU_I \mapsto R$	OCL Constraint 14

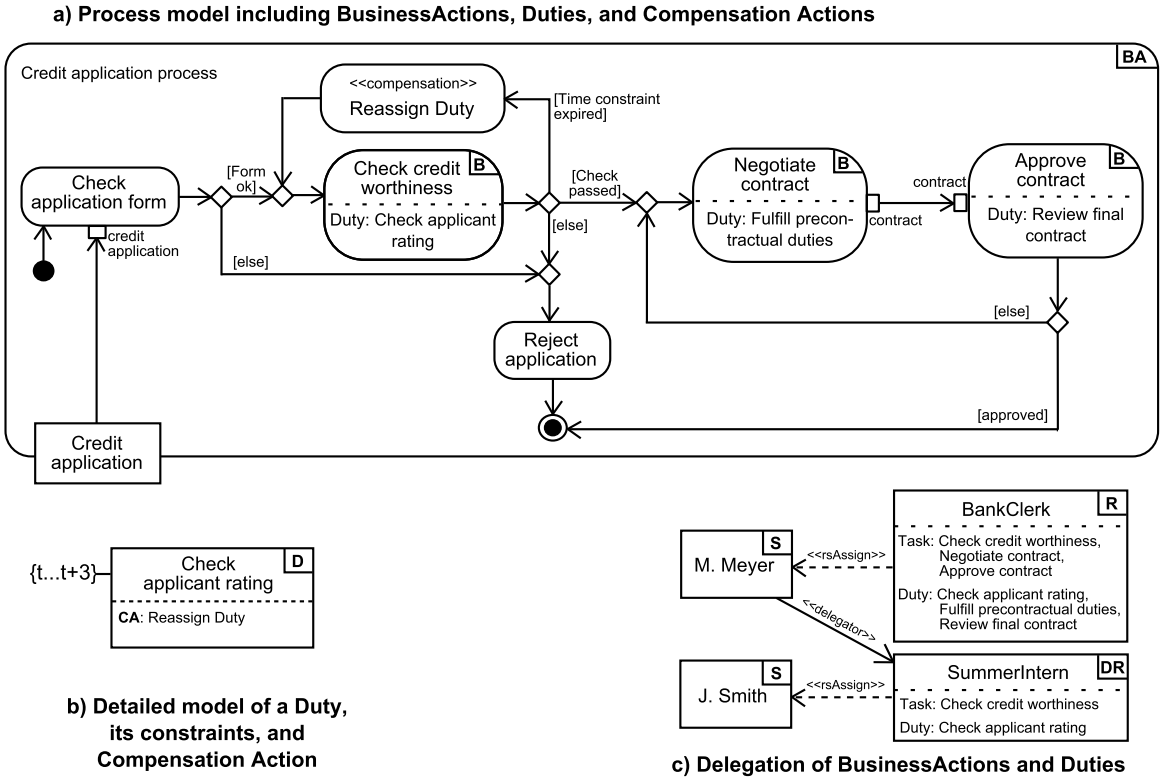

Fig. 12 Extended credit application process

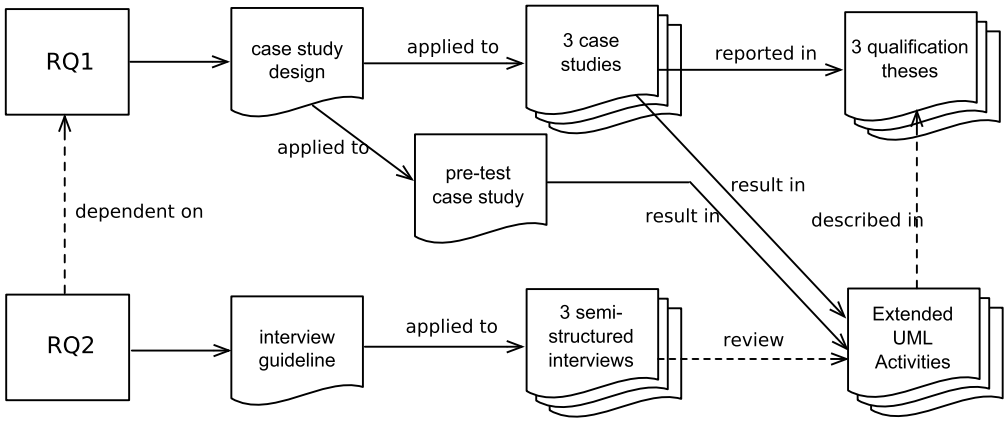

Fig. 13 Multi-method research design

new modeling constructs introduced in Sect. 5.1. The process in **Fig. 12a** includes five actions, three of which are defined as BusinessActions. The BusinessActions are associated with Duties: the BusinessAction *Check credit worthiness* is associated with the Duty *Check applicant rating*, the BusinessAction *Negotiate contract* with the Duty *Fulfill precontractual duties*, and the BusinessAction *Approve contract* with the Duty *Review final contract*. In addition, the Compensation Action *Reassign Duty* is triggered if the Duty *Check applicant rating* is not discharged in time.

Figure 12b presents the Duty *Check applicant rating* which is connected to the BusinessAction *Check credit worthiness*. It is associated with a DutyTimeConstraint and a Compensation Action. The DutyTimeConstraint expresses that the Duty *Check applicant rating* needs to be completed within three time units (e.g., days) after the corresponding BusinessAction has been started. Otherwise, the Compensation Action *Reassign Duty* is executed.

The responsibility for the Duties is illustrated in **Fig. 12c** showing the Role BankClerk which is assigned to the three BusinessActions and the associated Duties defined in the credit application process. Thus, a Subject assigned to the BankClerk role is responsible for performing these Duties and related BusinessActions. In this example, the Subject *M. Meyer* is assigned to the BankClerk role and therefore also needs to discharge the associated Duties. M. Meyer decides to delegate her Duty *Check applicant rating* to her summer intern *J. Smith*. For this purpose, she creates a permanent DelegationRole *SummerIntern* and assigns the Duty to the DelegationRole. Subsequently, she assigns the Subject *J. Smith* to her DelegationRole *SummerIntern*. J. Smith is now authorized and responsible for discharging the Duty *Check applicant rating* when performing the BusinessAction *Check credit worthiness*, until either the Duty is revoked from the DelegationRole or he loses his assignment to the DelegationRole.

6 Case Study on Modeling Process-Related RBAC Delegation Models

To evaluate our approach with regard to its practical applicability, we conducted a case study applying our UML extension on real-world processes. Our case study is based on a collection of real-world process models we retrieved from a large Austrian school center. The selection consists of about 30 processes, which were collected by members of the school during a process management initiative. The control flow of some processes was graphically visualized depicting the sequence of tasks and corresponding authorized/responsible persons. However, these processes were visualized using a non-standard ad hoc graphical notation. Furthermore, most of the processes were described in a detailed textual/tabular listing of activities with varying level of granularity. The process descriptions included references to legal requirements (e.g., certain paragraphs in the Austrian law concerning teaching in schools) and other internal or external regulations.

In the case study presented in this paper, we remodeled the processes that included information on delegation scenarios via our UML extension (see Sect. 5). This case study is part of a larger qualitative multi-method study presented in Schefer-Wenzl et al. (2013). In particular, we adopted a sequential multi-method research design with two subsequent research phases and two different research instruments (see **Fig. 13**). The two guiding research questions were: Which are the barriers to adopting our UML extensions by domain modelers having a basic background in UML activity modeling (RQ1)? Which are the barriers to using the process models based on our UML extensions for non-technical, non-security stakeholders in modeled organizations (RQ2)?

As for RQ1, we designed interpretative case studies because we wanted to address RQ1 using non-trivial process engineering tasks. RQ2 would then be covered by subsequent semi-structured interviews which would allow us to collect data concerning the communicability as perceived by important stakeholders. In addition, the interviews would permit clarifying critical model details for the respondents to improve the quality of the answers. For further details on the whole study, please refer to Schefer-Wenzl et al. (2013).

In **Fig. 14**, an example process from the case study on the UML extension presented in this paper is illustrated. The complete set of processes modeled in this case study is documented in Vondal (2012). **Figure 14a** depicts a BusinessActicity that models the process of organizing the school's open day. This process is part of a larger set of processes dealing with the organization of the open day. All new modeling elements introduced in the Delegations extension are used in this example process. The process depicted in **Fig. 14a** includes six BusinessActions. Two of these BusinessActions are associated with duties. Moreover, the process defines two subject-binding constraints between "Schedule date" and "Announce

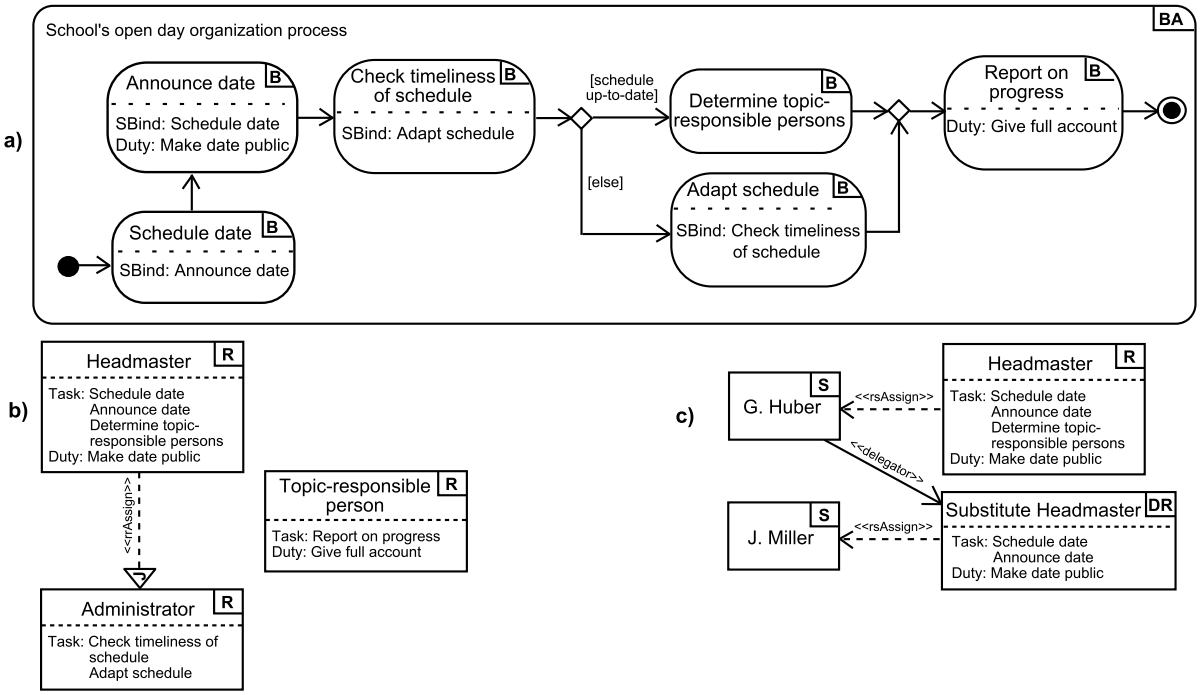

Fig. 14 Example process in an Austrian school

Table 3 Questions from the semi-structured interviews

Q1	Do the process models provide added value for the school? If yes, in how far can the school/members of the school benefit from the extended process models?
Q2	How will the extended process models potentially be used in the school?
Q3	What do you think about our approach of integrating process models and related security aspects? Advantages/Disadvantages?
Q4	Do you have difficulties in understanding different parts of the processes? If yes, which parts are easy to understand and which parts are difficult or not comprehensible?
Q5	Do you have any suggestions on how the graphical representation of the processes can be improved?

date" as well as between "Check timeliness of schedule" and "Adapt schedule". In **Fig. 14**b, roles and corresponding task and duty assignments are shown. We identified three roles for the open day organization process. For example, the headmaster of this school is permitted to perform five of the tasks in this process, two of these tasks are inherited from a junior-role.

Figure 14c illustrates which tasks and duties of the open day organization process a headmaster is allowed to delegate to his/her substitute headmaster. Note that these tasks and duties need to be defined as delegable before we can delegate them (see Sects. 2 and 5). The headmaster may delegate two of the tasks to his/her delegation role substitute headmaster. Moreover, when delegating a task being associated to a duty, this duty also has to be delegated (see Sects. 2 and 5). Subsequently, all delegatees being assigned to the delegation role substitute headmaster are authorized to perform the delegated tasks and duties.

After remodeling the processes via our UML extension, we evaluated the remodeled process diagrams of the case study by performing semi-structured interviews with three members of the school, including the headmaster, one teacher, and one member of the administrative staff. This approach was chosen because interviews are one of the most important methods in case study research (see, e.g., Runeson and Höst 2009). Moreover, for qualitative case studies it is recommended to choose subjects from different parts of the organization to involve different roles in the interviews (Corbin and Strauss 2008).

The interview was carefully designed using the guidelines from Hove and Anda (2005). It consisted of five main open-ended questions. Each interview varied between 20 and 25 minutes in length. The answers were recorded by using field notes which were then subsequently analyzed by the interviewer. **Table 3** details the main questions asked in the interviews.

In the interviews, two advantages of the visually modeled processes were communicated: First, the headmaster emphasized that new employees who are not familiar with school procedures would now have a comprehensive and easy-to-understand, diagram-based documentation of key processes and related delegation concerns at hand. This would have the potential of facilitating work tasks and communication with other school members during the first weeks after joining the school. This opinion may also support the frequently cited conjecture that models employing a process flow metaphor are suitable communication instruments for non-technical

domain experts (see, e.g., Dumas et al. 2012). In addition, before the case study was performed, only a few processes were depicted using an ad hoc (i.e., non-standard) visual notation. Most processes were described via textual documents in varying degrees of detail. The state of the organization's process descriptions was therefore inconsistent and inhomogeneous. Moreover, the interview partners noted that the access-control enriched process models would improve the general awareness among the school members of how closely security requirements are related to key organizational processes. All three members of the school stated that the process models are easy to comprehend (e.g., task and role labels, basic sequencing of tasks, relations between duties and tasks).

The case study design was aligned to evaluating our modeling framework. As a consequence, the study design presents limitations to the generalizability of our findings. An important limitation results from the scope of a single organization. The observations might therefore be specific to the domain of Austrian secondary schooling. However, within this domain, we aimed at a broad coverage of domain areas: the process models cover topics ranging from the school's process management to the emergency evacuation procedures. Nevertheless, future work must investigate whether the findings hold for different branches and different types of organizations.

In likewise manner, there are threats to the observations from the three interviews. To begin with, they cannot be generalized beyond the narrow educational domain because the interview partners are all embedded into a single institution. There is also the risk of an interviewer bias because the interviewer is also author of the evaluated UML extensions. This double role might have affected the open-ended conversation of the interviews. To minimize this risk, the interviewer, however, tried to observe rather than steer the conversation and encouraged the interviewees to talk.

7 Platform Support

In order to provide runtime support for the enforcement of process-related RBAC delegation models at the PSM layer, we implemented a corresponding extension to the Business Activity library and runtime engine. In this Section, we provide an overview of our platform support for process-related RBAC delegation models (available for download at BAL 2012). First, we present the Business Activity library and runtime engine.

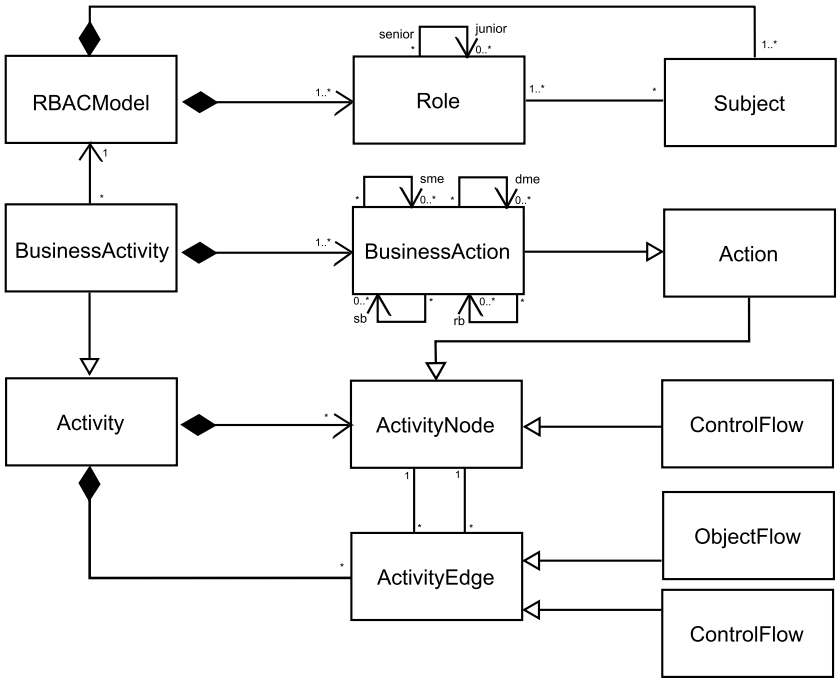

Fig. 15 Class model of the Business Activity library and runtime engine (Strembeck and Mendling 2011)

The Business Activity library and runtime engine is a software platform that can manage process-related RBAC runtime models and enforce access control policies as well as several kinds of entailment constraints (see Strembeck and Mendling 2011). It supports all artifacts of process-related RBAC models and provides functions for managing corresponding runtime instances. Moreover, it automatically enforces all invariants defined via OCL constraints (see Sect. 5.2). **Figure 15** shows an excerpt of the essential class relations of the Business Activity library and runtime engine.

The Delegation package extends the Business Activity library and runtime engine with support for process-related RBAC delegation models as defined in the previous Sections. **Figure 16** shows the essential class relations of the Delegation extension package. The Business Activity library and runtime engine as well as the Delegation extension package are implemented via the programming language eXtended Object Tcl (XOTcl, see, e.g., Neumann and Sobernig 2009, 2011; Neumann and Zdun 2000). XOTcl is an object-oriented extension of the scripting language Tcl (Ousterhout 1990) and is publicly available from Neumann and Zdun (2012). XOTcl is a C-library that can be dynamically loaded into Tcl-compatible environments and is embeddable into C programs. Amongst others, XOTcl provides a mixin mechanism (see Zdun et al. 2007). XOTcl mixin classes are a dynamic message interception technique. They allow to flexibly define extension classes in addition to the inheritance hierarchy.

XOTcl supports per-object mixins as well as per-class mixins. Per-object mixins are classes that are applied as mixins for an individual instance of a class, while per-class mixins are classes that are applied as mixins for a class (see Zdun et al. 2007 for details). Both XOTcl mixin constructs are used in the Delegation extensions package to dynamically activate or deactivate certain behavior for a class or object (see **Fig. 16**).

The Business Activity library and runtime engine in combination with the Delegation extension package ensures the compliance of processes modeled via the BusinessActivitiesDelegation extension and user-defined delegation policies. Thereby, it supports a straightforward mapping of modeling level RBAC delegation models to the corresponding runtime models.

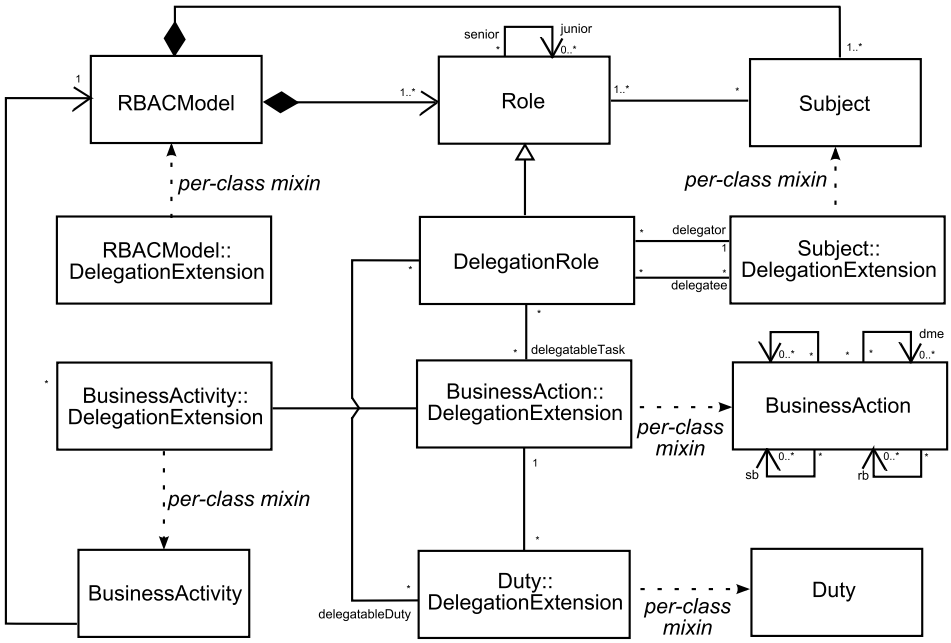

Fig. 16 Class model of the Delegation package

8 Related Work

In general, we distinguish three types of related work for this paper. First, we have approaches that primarily aim to integrate delegation aspects into role-based access control models. Second, a few approaches focus on the delegation of duties/obligations. Third, a number of different delegation approaches for business process/workflow environments exist. Many of the access control- and business process-related approaches are complementary to our work and are well-suited to be combined with our process-related RBAC delegation models.

Table 4 shows an overview of related work on modeling delegation of roles, tasks, and duties in an access control or business process context. With respect to the concepts and artifacts specified in Sects. 2, 3, 4, and 5, we use a $\surd$ if a related approach provides similar and/or comparable support for a certain concept, and a Δ if a related approach provides at least partial support for a particular aspect.

In recent years, there has been much work on various aspects of role- and permission-based delegation. In Barka and Sandhu (2000b), RBDM, a framework for characterizing role-based delegation models is presented which distinguishes, for instance, between permanent or temporary, partial or total, and single- or multi-step delegation. All of these concepts are also integrated in our delegation model presented in this paper. A formal model and some extensions for RBDM are presented in Barka and Sandhu (2000a). RDM2000 (Zhang et al. 2003a) is an extension of RBDM supporting role-based and multi-step delegation. Furthermore, a rule-based declarative language is proposed to specify and enforce policies. Similar to our approach, separation of duty constraints are considered and corresponding tool support is provided.

In Zhang et al. (2003b), a permission-based delegation model (PBDM) is presented which allows for delegation of roles and permissions. Delegation roles are defined to delegate permissions to a user. Most of the concepts introduced in Zhang et al. (2003b) are also integrated in our delegation model. Yet, support for entailment constraints in Zhang et al. (2003b) is limited to static separation of duty constraints, while we also consider binding constraints. An approach similar to Zhang et al. (2003b) is presented in Hasebe et al. (2010), where a capability-based delegation model (CRBAC) based on RBAC96 (Sandhu et al. 1996) is introduced to support cross-domain delegation of roles and permissions in terms of capability transfer. Compared to our work, none of these approaches supports the delegation of duties. Some delegation-related conflicts are only detected in Zhang et al. (2003b), without providing corresponding resolutions. Recently, an approach for the model-based specification of role-based delegation and revocation policies via UML was introduced in Sohr et al. (2012). In contrast to our approach, Sohr et al. (2012) do not integrate their concepts into a business process context. Moreover, they do not address the delegation of duties, and do not consider binding constraints. In addition, their modeling approach is not based on a formal metamodel. Instead, standard UML class and object diagrams are used for graphically visualizing delegation policies. Corresponding tool support as well as conflict detection and resolution handling is not provided in Sohr et al. (2012).

Duties or obligations may also be subject to delegation. Yet, the delegation of duties has received little attention in literature so far, although it has been identified as important phenomenon, e.g., in Cole et al. (2001), where different ways of delegating obligations are discussed. In Schaad and Moffett (2002), the delegation of obligations is addressed, mainly motivating the reasons for delegating obligations and stressing the need for balancing authorizations and obligations. Recently, a basic delegation model for obligations has been introduced in Ghorbel-Talbi et al. (2010, 2011). In this approach, different kinds of duty-level and role-level delegations are considered, also taking contextual information into account. However, in comparison to our work none of these approaches considers the delegation of duties in a business process context. Moreover, entailment constraints, correspond-

Table 4 Comparison of related work

	Delegation of roles	Delegation of tasks	Delegation of duties	Entailment constraints	Conflict detection	Conflict resolution	Formal metamodel (CIM layer)	Modeling support (PIM layer)	Tool support (PSM layer)
Role-based delegation models									
Barka and Sandhu (2000a, 2000b)	√						Δ		
Zhang et al. (2003a)	√			Δ			Δ		Δ
Shang and Wang (2008); Zhang et al. (2003b)	√			Δ	Δ		Δ		
Hasebe et al. (2010)	√						Δ		
Sohr et al. (2012)	√			Δ				Δ	
Delegation models for obligations									
Cole et al. (2001)			√						
Schaad and Moffett (2002)			√				Δ		
Ghorbel-Talbi et al. (2010, 2011)	√		√				Δ		
Delegation in business processes									
Gaaloul and Charoy (2009); Gaaloul et al. (2011, 2010)	Δ	√		√			√		Δ
Atluri and Warner (2005)		√		Δ	Δ		Δ		
Wainer et al. (2007)	√	√		Δ			Δ		Δ
Crampton and Khambhammettu (2008c)	√	√		Δ			Δ		
Crampton and Khambhammettu (2008a)	√	√		Δ	Δ		Δ		
Process-related RBAC delegation models (our approach)	√	√	√	√	√	√	√	√	√

ing modeling/tool support, or the detection and resolution of related conflicts is not further analyzed.

Delegation in a business process/workflow context has also received considerable attention. In Atluri and Warner (2005), the notion of delegation is extended to allow for conditional delegation. Different types of constraints, such as separation of duty constraints, are addressed in the context of delegation. Moreover, three types of conflicts as well as a runtime allocation algorithm comparable to Algorithm 4 presented in Sect. 4 are presented. A formal model for role-based and task-based delegation in workflows using the notions of case and organizational unit is described in Wainer et al. (2007). Compared to our work, Wainer et al. (2007) does not distinguish between subject-based and role-based binding constraints. Moreover, the detection and resolution of delegation-related conflicts is not discussed in Wainer et al. (2007). Similar approaches are also presented in Crampton and Khambhammettu (2008a, 2008c) without providing related modeling support and only limited support for conflict detection. The effects of some delegation operations on three workflow execution models are described in Crampton and Khambhammettu (2008c).

Only few contributions exist which consider authorization constraints and related conflicts in the context of delegation. Gaaloul and Charoy (2009) and Gaaloul et al. (2011, 2010) present a formal approach for integrating task delegation into the RBAC model which also considers separation of duty and binding of duty constraints. Compared to Gaaloul and Charoy (2009) and Gaaloul et al. (2011, 2010), our approach also considers the delegation of duties and provides a corresponding extension to the UML to enable the graphical visualization of process-related delegation concepts. In Shang and Wang (2008), an extension to PBDM is presented to integrate authorization constraints in permission-based delegation. In contrast to our work, Shang and Wang (2008) only focuses on static separation of duty constraints and shortly addresses related conflicts. Moreover, only role-based constraints are analyzed, while we consider task-based constraints. In Crampton and Khambhammettu (2008a), the satisfiability problem of workflows in the context of constrained delegation is addressed. Crampton also provides an algorithm that determines whether to permit a delegation request. However, the algorithm does not distinguish between different conflict types and does not provide corresponding resolutions in order to permit the delegation. Furthermore, this approach only considers task- and role-based delegation, while we also allow for the delegation of duties. Schaad addresses delegation conflicts in Schaad (2001). In contrast to our work, only conflicts between separation of duty constraints and delegation activities in the RBAC96 model are considered. Moreover, the conflicts are detected after conducting the delegation, while our algorithms detect conflicts before the delegation is performed. Thus, in our approach, conflicts are detected and

resolved before causing an inconsistent RBAC configuration.

To the best of our knowledge, this work represents the first attempt to systematically check for conflicts before delegating tasks, duties, and roles in a business process context at design- and runtime. In contrast to other approaches, we also consider mutual-exclusion and binding constraints and provide resolution strategies to resolve each conflict type (see **Table 4**).

9 Conclusion

In this paper, we presented an approach to support the integrated modeling of delegation concepts and business processes. Our approach is based on a formal CIM-layer metamodel for process-related RBAC delegation models. Moreover, we presented generic algorithms and resolution strategies for conflicts detected in the context of the delegation of tasks, duties, and roles. A special focus is on the problem of mutual-exclusion and binding constraints in an RBAC delegation context. Note that in our approach, conflicts are detected and resolved before causing an inconsistent RBAC configuration. Thereby, we ensure the continuous consistency of corresponding process-related RBAC delegation models.

At the PIM layer, we provide UML modeling support for the integrated modeling of business processes and corresponding delegation policies via extended UML Activity diagrams. Moreover, to support the controlled delegation of roles, tasks, and duties at the PSM layer we implemented our approach as a delegation extension for the BusinessActivity library and runtime engine, which is available for download at BAL (2012). We also performed a case study and conducted interviews to evaluate the practical applicability of our integrated modeling approach on real-world processes. In our future work, we plan to conduct further industrial case studies to analyze, for instance, potential issues regarding the complexity and comprehensibility of the graphical syntax of our modeling extension. Moreover, we will investigate how other security-related concepts can be integrated with the delegation extension. For instance, we intend to integrate our extension with other security extensions, such as the Secure Object Flows (SOF) extension introduced in Hoisl et al. (2014). Furthermore, we plan to use our generic CIM layer model to extend other process modeling languages (such as BPMN) with a delegation extension and analyze potential differences between different host languages with respect to these security extensions.

References

Atluri V, Warner J (2005) Supporting conditional delegation in secure workflow management systems. In: Proceedings of the 10th ACM symposium on access control models and technologies (SACMAT), pp 49–58

BAL (2012) Business activity library and runtime engine. http://wi.wu.ac.at/home/mark/BusinessActivities/library.html. Accessed 2012-09-24

Barka E, Sandhu R (2000a) A role-based delegation model and some extensions. In: Proceedings of the 23rd national information systems security conference (NISSEC)

Barka E, Sandhu R (2000b) Framework for role-based delegation models. In: Proceedings of the 16th annual computer security applications conference (ACSAC)

Basin D, Doser J, Lodderstedt T (2006) Model driven security: from UML models to access control Infrastructure. ACM Transactions on Software Engineering and Methodology 15(1):39–91

Botha RA, Eloff JH (2001) Separation of duties for access control enforcement in workflow environments. IBM Systems Journal 40(3):666–682

Casati F, Castano S, Fugini M (2001) Managing workflow authorization constraints through active database technology. Information Systems Frontiers 3(3):319–338

Cole J, Derrick J, Milosevic Z, Raymond K (2001) Author obliged to submit paper before 4 July: policies in an enterprise specification. In: Proceedings of the International workshop on policies for distributed systems and networks (POLICY), pp 1–17

Corbin J, Strauss A (2008) Basics of qualitative research: techniques and procedures for developing grounded theory. Sage, Thousand Oaks

Crampton J, Khambhammettu H (2008a) Delegation and satisfiability in workflow systems. In: Proceedings of the 13th ACM symposium on access control models and technologies (SACMAT), pp 31–40

Crampton J, Khambhammettu H (2008b) Delegation in role-based access control. International Journal of Information Security 7(2):123–136

Crampton J, Khambhammettu H (2008c) On delegation and workflow execution models. In: Proceedings of the 2008 ACM symposium on applied computing (SAC)

Dumas M, Rosa ML, Mendling J, Maesaku R, Hajo AR, Semenenko N (2012) Understanding business process models: the costs and benefits of structuredness. In: Proceedings of the 24th International conference on advanced information systems engineering (CAiSE)

Ferraiolo D, Barkley J, Kuhn D (1999) A role-based access control model and reference implementation within a corporate intranet. ACM Transactions on Information and System Security (TISSEC) 2(1)

Ferraiolo DF, Kuhn DR, Chandramouli R (2007) Role-based access control, 2nd edn. Artech House, Norwood

Abstract

Sigrid Schefer-Wenzl, Mark Strembeck

Modeling Support for Role-Based Delegation in Process-Aware Information Systems

In the paper, an integrated approach for the modeling and enforcement of delegation policies in process-aware information systems is presented. In particular, a delegation extension for process-related role-based access control (RBAC) models is specified. The extension is generic in the sense that it can be used to extend process-aware information systems or process modeling languages with support for process-related RBAC delegation models. Moreover, the detection of delegation-related conflicts is discussed and a set of pre-defined resolution strategies for each potential conflict is provided. Thereby, the design-time and runtime consistency of corresponding RBAC delegation models can be ensured. Based on a formal metamodel, UML2 modeling support for the delegation of roles, tasks, and duties is provided. A corresponding case study evaluates the practical applicability of the approach with real-world business processes. Moreover, the approach is implemented as an extension to the BusinessActivity library and runtime engine.

Keywords: Access control, Business processes, Delegation, Duties, RBAC, Security

Gaaloul K, Charoy F (2009) Task delegation based access control models for workflow systems. In: Proceedings of the 9th IFIP conference on e-business, e-services, and e-society (I3E)
Gaaloul K, Zahoor E, Charoy F, Godart C (2010) Dynamic authorisation policies for event-based task delegation. In: Proceedings of the 22nd International conference on advanced information systems engineering (CAiSE)
Gaaloul K, Proper E, Charoy F (2011) An extended RBAC model for task delegation in workflow systems. In: Proceedings of the workshops on business informatics research
Georgiadis CK, Mavridis I, Pangalos G, Thomas RK (2001) Flexible team-based access control using contexts. In: Proceedings of the 6th ACM symposium on access control models and technologies (SACMAT), pp 21–27
Ghorbel-Talbi MB, Cuppens F, Cuppens-Boulahia N (2010) Negotiating and delegating obligations. In: Proceedings of the International conference on management of emergent digital ecosystems (MEDES)
Ghorbel-Talbi MB, Cuppens F, Cuppens-Boulahia N, Metayer DL, Piolle G (2011) Delegation of obligations and responsibility. In: Proceedings of the International information security and privacy conference (SEC)
Hasebe K, Mabuchi M, Matsushita A (2010) Capability-based delegation model in RBAC. In: Proceedings of the 15th ACM symposium on access control models and technologies (SACMAT), pp 109–118
Hoisl B, Sobernig S, Strembeck M (2014) Modeling and enforcing secure object flows in process-driven SOAs: an integrated model-driven approach. Software and Systems Modeling 2:513–548
Hove SE, Anda B (2005) Experiences from conducting semi-structured interviews in empirical software engineering research. In: Proceedings of the 11th IEEE International software metrics symposium (METRICS)
Joshi JBD, Bertino E (2006) Fine-grained role-based delegation in presence of the hybrid role hierarchy. In: Proceedings of the 11th ACM symposium on access control models and technologies (SACMAT), pp 81–90
Jürjens J (2005) Sound methods and effective tools for model-based security engineering with UML. In: Proceedings of the 27th International conference on software engineering (ICSE)
Mouratidis H, Jürjens J (2010) From goal-driven security requirements engineering to secure design. International Journal of Intelligent Systems 25(8):813–840
Neumann G, Sobernig S (2009) XOTcl 2.0 – a ten-year retrospective and outlook. In: Proceedings of the sixteenth annual Tcl/Tk conference
Neumann G, Sobernig S (2011) An overview of the next scripting toolkit. In: Proceedings of the 18th annual Tcl/Tk conference
Neumann G, Zdun U (2000) XOTcl, an object-oriented scripting language. In: Proceedings of Tcl2k: the 7th USENIX Tcl/Tk conference
Neumann G, Zdun U (2012) XOTcl homepage. http://www.xotcl.org/. Accessed 2012-09-10
Oh S, Park S (2003) Task-role-based access control model. Information Systems 28(6): 533–562
OMG (2011a) Meta object facility (MOF) core specification. Version 2.4.1, formal/2011-08-07. The Object Management Group. http://www.omg.org/spec/MOF. Accessed 2012-02-27
OMG (2011b) Unified modeling language (OMG UML): superstructure. Version 2.4.1, formal/2011-08-06. The Object Management Group. http://www.omg.org/spec/UML
OMG (2014) Object constraint language specification. Version 2.4, formal/2014-02-03. The Object Management Group. http://www.omg.org/spec/OCL. Accessed 2014-04-25
Ousterhout J (1990) Tcl: an embeddable command language. In: Proceedings of the winter USENIX conference
Ravichandran A, Yoon J (2006) Trust management with delegation in grouped peer-to-peer communities. In: Proceedings of the 11th ACM symposium on access control models and technologies (SACMAT), pp 71–80
Recker J, Indulska M, Rosemann M, Green P (2006) How good is BPMN really? Insights from theory and practice. In: 14th European conference on information systems
Rodriguez A, de Guzman IGR (2007) Obtaining use case and security use cases from secure business process through the MDA approach. In: Proceedings of the international workshop on security in information systems (WOSIS)
Rodriguez A, Fernandez-Medina E, Piattini M (2006) Towards a UML 2.0 extension for the modeling of security requirements in business processes. In: Proceedings of the international conference on trust and privacy in digital business (TrustBus)
Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14(2):131–164
Russell N, Hofstede AHMT, Edmond D (2005) Workflow resource patterns: identification, representation and tool support. In: Proceedings of the 17th conference on advanced information systems engineering (CAiSE'05). Lecture notes in computer science, vol 3520. Springer, Heidelberg, pp 216–232
Sandhu R, Coyne E, Feinstein H, Youman C (1996) Role-based access control models. IEEE Computer 29(2):38–47
Schaad A (2001) Detecting conflicts in a role-based delegation model. In: Proceedings of the 17th annual computer security applications conference (ACSAC), pp 117–126
Schaad A, Moffett JD (2002) Delegation of obligations. In: Proceedings of the 3rd International workshop on policies for distributed systems and networks (POLICY)
Schefer S, Strembeck M (2011a) Modeling process-related duties with extended UML activity and interaction diagrams. Electronic Communications of the EASST, 37
Schefer S, Strembeck M (2011b) Modeling support for delegating roles, tasks, and duties in a process-related RBAC context. In: International workshop on information systems security engineering (WISSE). Lecture notes in business information processing. Springer, Heidelberg
Schefer S, Strembeck M, Mendling J, Baumgrass A (2011) Detecting and resolving conflicts of mutual-exclusion and binding constraints in a business process context. In: Proceedings of the 19th International conference on cooperative information systems (CoopIS). Lecture notes in computer science, vol 7044. Springer, Heidelberg
Schefer-Wenzl S, Strembeck M, Baumgrass A (2012) An approach for consistent delegation in process-aware information systems. In: Proceedings of the 15th International conference on business information systems (BIS). Lecture notes in business information processing, vol 117. Springer, Heidelberg
Schefer-Wenzl S, Sobernig S, Strembeck M (2013) Evaluating a UML-based modeling framework for process-related security properties: a qualitative multi-method study. In: Proceedings of the 21st European conference on information systems (ECIS), Utrecht
Schmidt DC (2006) Model-driven engineering – guest editor's introduction. IEEE Computer 39(2):25–31
Selic B (2003) The pragmatics of model-driven development. IEEE Software 20(5):19–25
Shang Q, Wang X (2008) Constraints for permission-based delegations. In: Proceedings of the 8th IEEE International conference on computer and information technology workshops (CITWORKSHOPS), pp 216–223
Sloman MS (1994) Policy driven management for distributed systems. Journal of Network and Systems Management 2(4):333–360
Sohr K, Kuhlmann M, Gogolla M, Hu H, Ahn GJ (2012) Comprehensive two-level analysis of role-based delegation and revocation policies with UML and OCL. Information and Software Technology 54(12):1396–1417
Stahl T, Völter M (2006) Model-driven software development. Wiley, New York
Strembeck M (2005) Embedding policy rules for software-based systems in a requirements context. In: Proceedings of the 6th IEEE International workshop on policies for distributed systems and networks (POLICY)
Strembeck M (2010) Scenario-driven role engineering. IEEE Security & Privacy 8(1):28–35
Strembeck M, Mendling J (2010) Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In: Proceedings of the 18th International conference on cooperative information systems (CoopIS). Lecture notes in computer science, vol 6426. Springer, Heidelberg
Strembeck M, Mendling J (2011) Modeling process-related RBAC models with extended UML activity models. Information and Software Technology 53(5):456–483
Tan K, Crampton J, Gunter CA (2004) The consistency of task-based authorization constraints in workflow systems. In: Proceedings of the 17th IEEE workshop on computer security foundations
Thomas RK, Sandhu RS (1997) Task-based authorization controls (TBAC): a family of models for active and enterprise-oriented authorization management. In: Proceedings of the IFIP TC11 WG11.3 11th International conference on database security XI: status and prospects, pp 166–181
Vondal F (2012) Modellierung von Delegation in prozessbezogenen RBAC-Modellen – Eine Fallstudie. Bachelor thesis, WU Vienna
Wainer J, Barthelmess P, Kumar A (2003) W-RBAC – a workflow security model incorporating controlled overriding of constraints. International Journal of Cooperative Information Systems 12(4):455
Wainer J, Kumar A, Barthelmess P (2007) DW-RBAC: a formal security model of delegation and revocation in workflow systems. Information Systems 32(3):365–384
Warner J, Atluri V (2006) Inter-instance authorization constraints for secure workflow management. In: Proceedings of the 11th

ACM symposium on access control models and technologies (SACMAT), pp 190–199
Weske M (2012) Business process management: concepts, languages, architectures, 2nd edn. Springer, Heidelberg
Wolter C, Schaad A, Meinel C (2008) A transformation approach for security enhanced business processes. In: Proceedings of the IASTED International conference on software engineering
Wolter C, Menzel M, Schaad A, Miseldine P, Meinel C (2009) Model-driven business process security requirement specification. Journal of Systems Architecture 55(4):211–223
Zdun U, Strembeck M, Neumann G (2007) Object-based and class-based composition of transitive mixins. Information and Software Technology 49(8):871–891
Zhang L, Ahn GJ, Chu BT (2003a) A rule-based framework for role-based delegation and revocation. ACM Transations on Information System Security 6(3):404–441
Zhang X, Oh S, Sandhu R (2003b) PBDM: a flexible delegation model in RBAC. In: Proceedings of the 8th ACM symposium on access control models and technologies (SACMAT), pp 149–157
Zhao G, Chadwick D, Otenko S (2007) Obligations for role based access control. In: Proceedings of the 21st International conference on advanced information networking and applications workshops (AINAW), pp 424–431
zur Muehlen M, Indulska M (2010) Modeling languages for business processes and business rules: a representational analysis. Information Systems 35(4):379–390