Information and Software Technology 53 (2011) 456–483

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Modeling process-related RBAC models with extended UML activity models

Mark Strembeck [a,*], Jan Mendling [b,1]

[a] Institute of Information Systems, New Media Lab, Vienna University of Economics and Business (WU Vienna), Austria
[b] Institute of Information Systems, Humboldt-Universität zu Berlin, Germany

A R T I C L E   I N F O

Article history:
Received 28 July 2010
Received in revised form 30 November 2010
Accepted 30 November 2010
Available online 13 December 2010

Keywords:
Process modeling
Role-based access control
Role engineering
Systems modeling
UML

A B S T R A C T

Context: Business processes are an important source for the engineering of customized software systems and are constantly gaining attention in the area of software engineering as well as in the area of information and system security. While the need to integrate processes and role-based access control (RBAC) models has been repeatedly identified in research and practice, standard process modeling languages do not provide corresponding language elements.
Objective: In this paper, we are concerned with the definition of an integrated approach for modeling processes and process-related RBAC models – including roles, role hierarchies, statically and dynamically mutual exclusive tasks, as well as binding of duty constraints on tasks.
Method: We specify a formal metamodel for process-related RBAC models. Based on this formal model, we define a domain-specific extension for a standard modeling language.
Results: Our formal metamodel is generic and can be used to extend arbitrary process modeling languages. To demonstrate our approach, we present a corresponding extension for UML2 activity models. The name of our extension is *Business Activities*. Moreover, we implemented a library and runtime engine that can manage Business Activity runtime models and enforce the different policies and constraints in a software system.
Conclusion: The definition of process-related RBAC models at the modeling-level is an important prerequisite for the thorough implementation and enforcement of corresponding policies and constraints in a software system. We identified the need for modeling support of process-related RBAC models from our experience in real-world role engineering projects and case studies. The Business Activities approach presented in this paper is successfully applied in role engineering projects.

1. Introduction

Each business process includes a number of tasks that need to be executed sequentially or in parallel in order to successfully complete the process [8,62,82]. In this context, access permissions grant the right to perform a certain task. Therefore, the human users participating in a workflow must own the permissions that are needed to execute the corresponding tasks.

Access control deals with the elicitation, specification, maintenance, and enforcement of authorization policies in software-based systems [40,71]. In recent years, role-based access control (RBAC) [23,24,70] has developed into a de facto standard for access control in both research and industry. In the context of RBAC, roles are used to model different job positions and scopes of duty within a particular organization or within an information system. These roles are equipped with the permissions that are needed to perform their respective tasks. Human users and other active entities (subjects) are assigned to roles according to their work profile [84,86]. Therefore, RBAC directly supports the principle of least privilege because each user can be assigned to the exact roles, and thus owns the exact number of permissions, that are needed to fulfill his duties. Thoroughly engineered roles also tend to change significantly slower than the assignment of individuals to these roles. Thus, establishing roles as an abstraction mechanism for subjects significantly facilitates the administration of permissions. In addition, the advantages of RBAC on the modeling and technical levels directly translate into lower maintenance costs [26].

In order to model process-related RBAC models, we therefore need an approach that integrates the different concepts in a consolidated modeling language. To actually enforce the process definitions and access control policies in an information system, the corresponding models need to be mapped to the respective software platform. However, while a number of sophisticated

* Corresponding author. Address: Augasse 2-6, A-1090 Vienna, Austria. Tel.: +43 1 31336 4983.
E-mail addresses: mark.strembeck@wu.ac.at (M. Strembeck), jan.mendling@wiwi.hu-berlin.de (J. Mendling).
1 Address: Spandauer Strasse 1, D-10178 Berlin, Germany. Tel.: +49 30 2093 5805.

0950-5849/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2010.11.015

approaches exist that allow for the formal specification and analysis of process-related access control policies and constraints (see, e.g., [10,43,90]), corresponding modeling support for software systems is largely missing. In this paper, we present an integrated approach to model processes and process-related RBAC models. In addition, we implemented a library and runtime engine to straightforwardly map these processes and RBAC models to a software platform.

1.1. Motivation

Role engineering is the process of defining roles, permissions, constraints, and role hierarchies. In scenario-driven role engineering [84,86,88], we use scenario and process models as a primary communication and engineering vehicle. Since its first publication in 2002, scenario-driven role engineering was successfully applied in many projects and is used by a number of consulting firms and international projects (see, e.g., [16]). From our experience in role engineering projects and case studies [39,54,84,86,88] we can say that RBAC as well as mutual exclusion and binding of duty constraints are some of the theoretical concepts that are actually and frequently used in the "real-world". Nevertheless, due to missing modeling support for process-related RBAC models, real-world projects most often try to specify these concepts with ad hoc extensions to modeling languages or via informal textual comments. However, such types of work-arounds result in significant problems for the comprehensibility and maintainability of these ad hoc models, and they make it difficult to translate the respective modeling-level concepts to actual software systems. In addition to our own experience, the demand for an integrated modeling support of business processes, roles, and corresponding constraints, such as separation of duty and binding of duty, has been repeatedly identified in the literature [44,68,69,98]. This experience and real-world problems led us to the approach we present in this paper.

In general, different types of problems arise in the context of processes and process-related RBAC models. First, process modeling languages such as BPMN [59], EPCs [46,72], Petri nets [63,64], or UML activity models [61] do not provide native language constructs to model access control policies. Second, while some approaches exist to integrate process models with access control policies or constraints on different abstraction levels (see, e.g., [9,48,49,100]), there is a problem that the language used for process modeling is most often different from (or not integrated with) the system modeling language that is used to specify the corresponding software system. This may again result in problems because different modeling languages provide different language abstractions that cannot easily be mapped to each other. In particular, such semantic gaps may involve significant efforts when conceptual models from different languages need to be integrated and mapped to a software platform [6,36,101].

However, a complete and correct mapping of process definitions and related access control policies to the corresponding software system is essential in order to assure consistency between the modeling-level specifications on the one hand, and the software system that actually manages corresponding process instances and enforces the respective policies on the other. The demand to ensure that runtime process instances comply with modeling-level processes and policies becomes even more pressing with recent laws and regulations such as the Sarbanes-Oxley Act (SOX), the Health Insurance Portability and Accountability Act (HIPAA), or the Basel II Accord. For example, adequate support for the definition and enforcement of process-related access control policies, including separation of duty constraints, is one important part of SOX compliance [13,18,52]. Moreover, corresponding compliance requirements also arise from security recommendations and standards such as the NIST security handbook [55], the NIST recommended security controls [56], the ISO 27000 standard family [29–31] (formerly ISO 17799), legally binding agreements such as business contracts, or company-specific (internal) rules/regulations.

1.2. Approach synopsis

In model-driven software development (MDSD) [73,74,81] a common approach is to develop a domain-specific language (DSL) that provides relevant domain abstractions as first-class language elements [51,80,89,103]. To ensure compliance between models and software platforms, the models defined in a DSL are mapped to source code artifacts of the software platform via automated model transformations [50,75,102]. In general, a DSL can be a standalone language or it can be embedded in a host language and extend the host language with domain-specific language abstractions.

In this paper, we present *Business Activities* as an integrated approach to enable the specification of process flows as well as process-related RBAC models and constraints. Fig. 1 shows an informal overview of the main conceptual elements included in the Business Activity approach. In particular, we first define a formal generic metamodel for Business Activities (see Section 3). This generic metamodel is based on the consensual core elements of process models and can be used to extend arbitrary process modeling languages. However, in scenario-driven role engineering we use UML2 activity and interaction models as standard means for scenario and process modeling [86]. Moreover, UML2 [61] is the de facto standard for software systems modeling. Therefore, we chose to define an extension to the UML2 standard to demonstrate our approach. In particular, we use our generic metamodel to define a corresponding extension to UML2 activity models (see Section 4).

UML2 activity models provide a process modeling language that allows one to model the control and object flows between different actions. The main element of an activity diagram is an activity. Its behaviour is defined by a decomposition into different actions. An 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 [61]). Fig. 2 shows an example of a credit application process modeled via a standard UML2 activity diagram.

Our UML extension for Business Activities provides domain-specific language abstractions that support the definition of

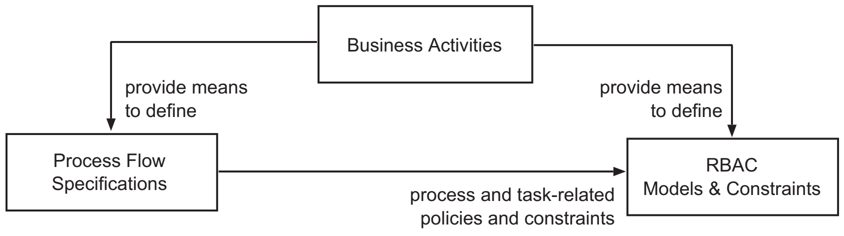

Business Activities
provide means to define
provide means to define
Process Flow Specifications
RBAC Models & Constraints
process and task-related policies and constraints

Fig. 1. Business Activities model processes and process-related RBAC models.

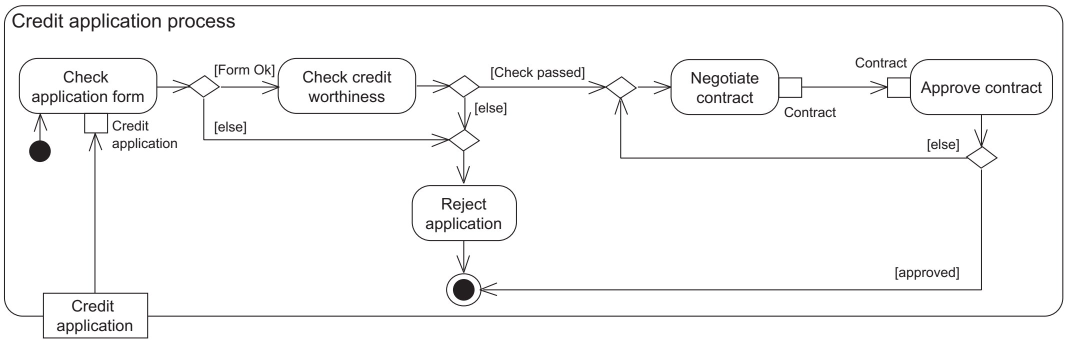


Fig. 2. A simple credit application process modeled as standard UML activity diagram.

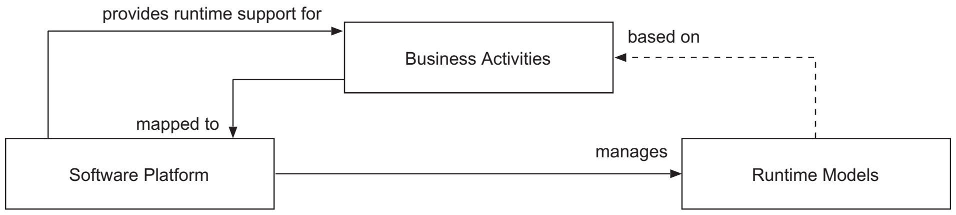


Fig. 3. Platform-independent Business Activities are mapped to platform-specific runtime models.

process-related RBAC models. In addition, the extension directly supports the definition of mutual exclusion and binding of duty constraints (for details see Sections 3 and 4). We use the object constraint language (OCL, [58]) to specify invariants for our UML2 extension. Corresponding software tools can enforce these invariants on the modeling-level as well as in runtime models. Thereby, we can ensure the consistency of our Business Activity models with the respective constraints. Moreover, our UML extension can be applied to supplement other UML-based approaches and can be integrated in UML-based software tools. However, note that our general approach does not depend on UML and may also be applied to extend other process modeling languages.

In addition to our modeling extension, we also implemented a corresponding software platform to ensure compliance of the processes and RBAC policies modeled via Business Activities on the one hand, and the respective runtime models on the other. Our software platform for Business Activities allows for a direct mapping of the different modeling level artifacts to corresponding source code structures (see Fig. 3).

The remainder of this paper is structured as follows. Section 2 gives an overview of access control in a business process context with a special emphasis on separation of duty and binding of duty. Next, Section 3 presents a formal and generic model for Business Activities. Subsequently, Section 4 presents our UML extension for Business Activities to demonstrate how the generic metamodel can be used to extend a standard modeling language, and Section 5 gives examples for Business Activity models. In Section 6, we discuss insights from real-world projects, and describe how our Business Activities help resolve different issues we identified in these projects. Section 7 gives an overview of our software platform for Business Activities (which includes a corresponding library and runtime engine), as well as the mapping of Business Activity models to runtime models. Next, Section 8 discusses related work, and Section 9 concludes the paper.

2. Access control in a business process context

When performing an IT-supported workflow, human users and proactive/autonomous software agents have to fulfill certain tasks to execute the workflow. Each action in a workflow (like changing a document or sending a message) is typically associated with a certain access operation (e.g. to a document or a messaging service, respectively). Thus, an active entity participating in a workflow (be it a human user or a software agent) must be authorized to perform the actions that are needed to complete its tasks in the workflow [10,27,57,91,92,95]. The business processes of an organization are therefore an ideal source to identify and define a tailored set of access control policies for this organization, as well as their information system [54,85,86,88].

In the context of business process modeling, separation of duty constraints and binding of duty constraints are important means to assist the specification of business processes and to control corresponding workflows [10,11,90,95].

2.1. Separation of duty

Separation of duty (SOD) constraints enforce conflict of interest policies [3,14,22,42,83]. 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. Thus, the definition of mutual exclusive entities is a well-known mechanism to enforce separation of duty. *Mutual exclusive* tasks result from the division of powerful rights or responsibilities to prevent fraud and abuse. An example is the common practice to separate the tasks of the "controller" role and the "chief buyer" role in medium-sized and large companies.

SOD constraints can be subdivided in static separation of duty constraints and dynamic separation of duty constraints. In essence, *static separation of duty* constraints specify that two mutual exclusive entities must *never* be *assigned* to the same subject *simultaneously*. *Dynamic separation of duty* constraints define that two mutual exclusive entities must *never* be *activated* by the same subject *simultaneously*. This means that two dynamically mutual exclusive entities may be assigned to the same subject. The corresponding subject, however, is only allowed to activate at most one of its dynamically mutual exclusive entities at the same time.

In this context, a *task-based separation of duty constraint* is a SOD constraint that considers task order and task history in a particular process instance to decide if a certain subject or role is allowed to perform a certain task [11,28,90,92,97]. Again, task-based SOD constraints can be static or dynamic. A *static task-based SOD constraint* can be enforced by defining that two statically mutual exclusive (SME) tasks must never be assigned to the same role and must never be performed by the same subject. This constraint is global with respect to *all process instances* in the corresponding information system. For example, a company may choose to define two SME tasks "Order supplies" and "Approve payment" to prevent fraud and abuse.

In contrast, a *dynamic task-based SOD constraint* refers to individual process instances and can be enforced by defining that two dynamically mutual exclusive (DME) tasks must never be performed by the same subject in the *same process instance*. In other words: two DME tasks can be assigned to the same role. However, to complete a process instance which includes two DME tasks, one needs at least two different subjects. This means, although a subject might possess a role which includes all permissions to perform two DME tasks, a DME constraint enforces that the same subject does not perform both tasks in the same process instance.

For example, in the credit application process from Fig. 2 a bank may assign the tasks "Negotiate contract" and "Approve contract" to the "Bank clerk" role and define these tasks as dynamically mutual exclusive. Each subject owning the "Bank clerk" role may then perform both tasks. Nevertheless, because of the DME constraint on these tasks, we always need at least two bank clerks to complete an instance of the credit application process.

2.2. Binding of duty

Binding of Duty (BOD) constraints define a connection between two (or more) tasks so that a subject (or role) who performed one of these tasks must also perform the corresponding related task(s). In other words, in a given process instance two "bound tasks" must always be performed by the same subject (or role), e.g. because of specific knowledge the subject acquires while performing the first of two bound tasks, for reasons of corporate or organization-internal processing standards, or to simplify interaction with other process stakeholders [90,95].

Moreover, binding of duty can be subdivided in subject-based and role-based constraints. A *subject-based BOD constraint* then defines that the *same individual* who performed the first task must also perform the bound task(s). In contrast to that, a *role-based BOD constraint* defines that bound tasks must be performed by members of the *same role*, but not necessarily by the same individual. Throughout the paper, we will use the terms *subject-binding* and *role-binding* as synonyms for subject-based BOD constraints and role-based BOD constraints respectively.

Consider an example where a role-binding is defined on the tasks "Check credit worthiness" and "Negotiate contract" shown in Fig. 2. If an instance of the "Check credit worthiness" task is performed by a subject owning the "Bank clerk" role, the corresponding "Negotiate contract" task must also be performed by a member of the "Bank clerk" role. This means that the second task can be performed by any subject owning the "Bank clerk" role, and this subject *may* be the same individual who performed the "Check credit worthiness" task.

In contrast to that, if we would define a subject-binding on the "Check credit worthiness" and "Negotiate contract" tasks, the second task *must* always be performed by the same individual who performed the first task.

3. A formal and generic metamodel for Business Activities

In this section, we present a formal generic metamodel for processes and process-related RBAC models. The metamodel is based on the consensual set of basic modeling elements for process models [46]. In principle, it can be integrated with any process modeling language that defines control flows between tasks. Furthermore, such a formalization is of central importance to discuss correctness properties because process-related RBAC models can easily become complex. As a first step, we formalize access control concepts for Business Activities via a so-called Business Activity RBAC Model in Section 3.1. This formalization helps us define formal correctness criteria for process-related RBAC models. We provide a set of static correctness criteria which ensure the integrity of the model when defining mutual exclusion and binding constraints for a set of Business Activities. In addition, we define dynamic correctness criteria that take the runtime allocation of tasks to subjects into account. Section 3.2 integrates the Business Activity RBAC Model with common routing elements of process modeling languages via a so-called Business Activity Flow Model. We formalize its behavioural semantics as a labeled transition system [15,53], which provides the foundation for checking satisfiability. Because access control constraints relate to both process instance and process type information, we have to include both in our metamodel. This approach is similar to the metamodel formalization of a workflow system in general as introduced by [41] (which does not cover access control in detail, however). In this way, our formal metamodel also provides an approach for the modeling, implementation, and enforcement of SOD and BOD constraints in a business process context (as identified by the workflow resource patterns [68] for instance).

3.1. Generic metamodel for Business Activity RBAC models

We first define the essential concepts of a Business Activity RBAC model including subjects, roles, tasks, and processes. Tasks and processes have to be defined both on an instance and a type level. This is because mutual exclusion and binding constraints are specified at the level of task types, while a number of respective correctness criteria have to be discussed in relation to particular executions of a process, i.e. in the context of task and process instances.

Fig. 4 shows essential relations of subjects (users), roles, tasks, and processes via a class diagram [61]. Note that Fig. 4 includes "exclusion" and "binding" associations for tasks only. While mutual exclusion constraints can also be defined on the level of roles [22,70], the definition of mutual exclusive roles may result in a number of computational problems [42]. Moreover, the context in a workflow system is given through process instances and corresponding task instances. In this paper, we therefore focus on mutual exclusion and binding constraints defined on the task level (i.e. on the permission level). In Fig. 4, the exclusion relations between tasks are represented via the `staticExclusion` and `dynamicExclusion` associations defined on `TaskType`. Likewise, the binding relations between tasks are represented through the `subjectBinding` and `roleBinding` associations on `TaskType`.

A role that is associated with two mutual exclusive tasks then inherits the corresponding SME or DME constraint from these tasks (see below). 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 1 first specifies the essential elements of the metamodel and their basic interrelations.

Definition 1 (**Business Activity RBAC Model**). A Business Activity RBAC Model $BRM = (E,Q,D)$ where $E = S \cup R \cup P_T \cup P_I \cup T_T \cup T_I$ refers to pairwise disjoint sets of the metamodel, $Q = rh \cup rsa \cup es \cup er \cup tra \cup ti$ to mappings that establish relationships, and $D = sb \cup rb \cup sme \cup dme$ to binding and mutual exclusion constraints, such that

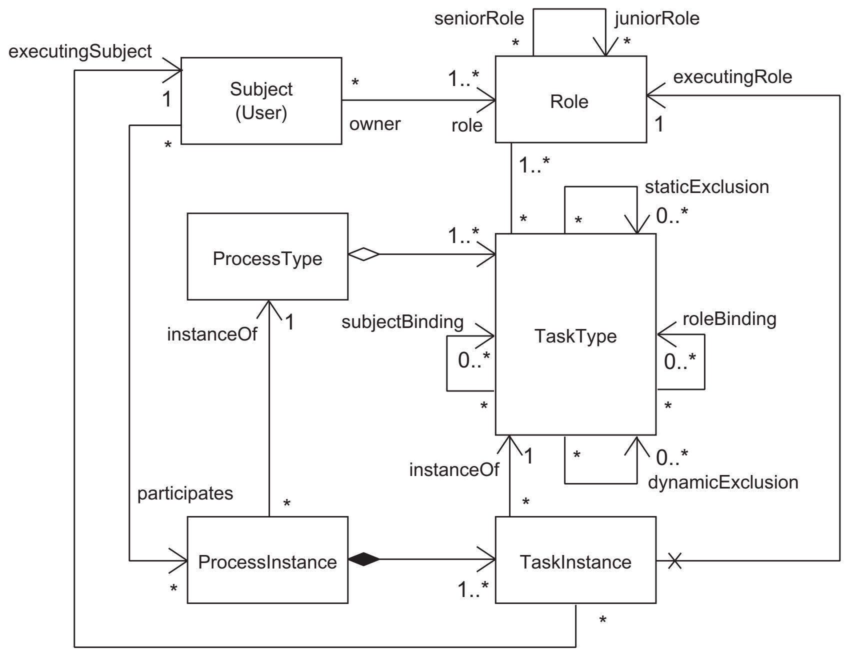


Fig. 4. Conceputal overview: main elements of Business Activity RBAC models.

- For the sets of the metamodel:
  - 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*.

Below, we iteratively define the partial mappings of the Business Activity RBAC Model and provide corresponding formalizations ($\mathscr{P}$ refers to the power set):

1. Roles can be arranged in a role hierarchy [24,70], where senior-roles inherit the permissions from their junior-roles. For example, in a banking context we may have a junior-role "Bank Clerk" and a corresponding senior-role "Bank Manager". The "Bank Manager" role then inherits all permissions from the "Bank Clerk" role. Moreover, because the "Bank Clerk" role may itself have one or more junior-roles (such as a role "Bank Intern"), the "Bank Manager" does inherit the permissions from its direct as well as his transitive junior-roles. Formally: *The mapping* $rh : R \mapsto \mathscr{P}(R)$ *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. Task types are assigned to roles to define the permissions of the corresponding roles. The task-to-role assignment relation is a many-to-many relation, so that each role can own the permissions to execute several different task types and each task type may be assigned to several roles. Moreover, in a role hierarchy each role owns the permissions that are directly assigned to this role, as well as the permissions inherited from its junior-roles. Consider an example where the junior-role "Bank Intern" has the (directly assigned) permission to perform the task "Check application form", and its senior-role "Bank Clerk" has the (directly assigned) permission to perform the "Negotiate contract" task (see Fig. 2). In this example, the "Bank Clerk" role inherits the permission to perform the "Check application form" task from its junior-role "Bank Intern". Formally: *The mapping* $tra : R \mapsto \mathscr{P}(T_T)$ *is called* **task-to-role assignment**. *For* $tra(r) = T_r$ *we call* $r \in R$ role *and* $T_r \subseteq T_T$ *is called the set of* tasks assigned to r. *The mapping* $tra^{-1} : T_T \mapsto \mathscr{P}(R)$ *returns the set of roles a task is assigned to* (*the set of roles owning a task*). *This assignment implies a mapping* **task ownership** $town : R \mapsto \mathscr{P}(T_T)$, *such that for each role* $r \in R$ *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* $town^{-1} : T_T \mapsto \mathscr{P}(R)$ *returns the set of roles a task is assigned to (directly or transitively via a role hierarchy*).
3. Roles are assigned to subjects, and through their roles the subjects acquire the rights to execute corresponding tasks. The role-to-subject assignment relation is a many-to-many relation, so that each subject may own several roles and each role can be assigned to different subjects. For example, in case the "Bank Clerk" role is assigned to two subjects called Alice and Bob, both can perform all tasks assigned to the "Bank Clerk" role. Formally: *The mapping* $rsa : S \mapsto \mathscr{P}(R)$ *is called* **role-to-subject assignment**. *For* $rsa(s) = R_s$ *we call* $s \in S$ *subject* and $R_s \subseteq R$ *the set of* roles assigned to this subject (*the set of roles owned by* s). *The mapping* $rsa^{-1} : R \mapsto \mathscr{P}(S)$ *returns all subjects assigned to a role (the set of subjects owning a role*). *This assignment implies a mapping* **role ownership** $rown : S \mapsto \mathscr{P}(R)$, *such that for each subject* $s \in S$ *all direct and inherited roles are included, i.e.* $rown(s) = \bigcup_{r \in rsa(s)} rh^*(r) \cup rsa(s)$. *The mapping* $rown^{-1} : R \mapsto \mathscr{P}(S)$ *returns all subjects assigned to a role (directly or transitively via a role hierarchy).*
4. Each process type consists of an arbitrary number of task types, and each task type can be associated with an arbitrary number of process types. Thus the process-type to task-type relation is a many-to-many relation. For example, the credit application process from Fig. 2 consists of five task types. Each of these five task types may also be associated with

other process types. The "Approve contract" task type could also be associated with a "Purchase real estate" process type, for instance. Formally: *The mapping* $ptd : P_T \mapsto \mathscr{P}(T_T)$ *is called* **process type definition**. *For* $ptd(p_T) = T_{p_T}$ *we call* $p_T \in P_T$ *process type and* $T_{p_T} \subseteq T_T$ *the set of task types associated with* p_T.

5. For each process type we can create an arbitrary number of corresponding process instances. For example, each actual credit application is handled via an own instance of the credit application process from Fig. 2. Formally: *The mapping* $pi : P_T \mapsto \mathscr{P}(P_I)$ *is called* **process instantiation**. *For* $pi(p_T) = P_i$ *we call* $p_T \in P_T$ *process type and* $P_i \subseteq P_I$ *the set of process instances instantiated from process type* p_T.
6. For each task type we can create an arbitrary number of respective task instances. For example, for each instance of the credit application process (see Fig. 2) we also have an own instance of the "Check credit worthiness" task type. Thus, if we have two instances of the credit application process, e.g. one for applicant Claudia and one for applicant Diane, we also have two instances of the "Check credit worthiness" task for each of the two process instances. Formally: *The mapping* $ti : (T_T \times P_I) \mapsto \mathscr{P}(T_I)$ *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. Because role-to-subject assignment is a many-to-many relation (see Definition 1.3), more than one subject may be able to execute instances of a certain task type. For example, if we have two subjects, Alice and Bob, who are assigned to the "Bank Clerk" role, each of them may execute instances of the "Check credit worthiness" task type (which we assume is assigned to this role). However, for each instance of the respective task type only one of the potential candidates actually executes the corresponding task instance. For example, if we have two credit applications filed by Claudia and Diane, respectively, Alice may execute the "Check credit worthiness" task instance for Claudia, while Bob executes the corresponding task instance for Diane. The subject executing a particular task instance is called the executing-subject of this instance. Formally: *The mapping* $es{:}T_I \mapsto S$ *is called* **executing-subject** *mapping. For* $es(t) = s$ *we call* $s \in S$ *the* executing subject and $t \in T_I$ is called *executed task instance*.
8. Via the role hierarchy, different roles may posses the permission to perform a certain task type (see Definitions 1.1 and 1.2). For example, if the "Check application form" task type is assigned to the "Bank Intern" role, this permission is also inherited by the corresponding "Bank Clerk" senior-role. The role that is used to actually execute a certain task instance is called the executing-role of this instance. Formally: *The mapping* $er{:}T_I \mapsto R$ *is called* **executing-role** *mapping. For* $er(t) = r$ *we call* $r \in R$ *the* executing role *and* $t \in T_I$ *is called* executed task instance.
9. To enforce subject-based BOD constraints, we define subject-binding relations between task types (for a discussion of subject-based BOD see Section 2). Formally: *The mapping* $sb : T_T \mapsto \mathscr{P}(T_T)$ *is called* **subject-binding**. *For* $sb(t_1) = T_{sb}$ *we call* $t_1 \in T_T$ *the* subject binding task and $T_{sb} \subseteq T_T$ the set of *subject bound tasks*.
10. To enforce role-based BOD constraints, we define role-binding relations between task types (for a discussion of role-based BOD see Section 2). Formally: *The mapping* $rb : T_T \mapsto \mathscr{P}(T_T)$ *is called* **role-binding**. *For* $rb(t_1) = T_{rb}$ *we call* $t_1 \in T_T$ *the* role binding task and $T_{rb} \subseteq T_T$ *the set of* role bound tasks.
11. To enforce static task-based SOD constraints, we define static mutual exclusion (SME) relations between task types (for a discussion of static SOD see Section 2). Formally: *The mapping* $sme : T_T \mapsto \mathscr{P}(T_T)$ *is called* **static mutual exclusion**. *For* $sme(t_1) = T_{sme}$ *with* $T_{sme} \subseteq T_T$ *we call each pair* $t_1 \in T_T$ *and* $t_x \in T_{sme}$ statically mutual exclusive tasks.
12. To enforce dynamic task-based SOD constraints, we define dynamic mutual exclusion (DME) relations between task types (for a discussion of dynamic SOD see Section 2). Formally: *The mapping* $dme : T_T \mapsto \mathscr{P}(T_T)$ *is called* **dynamic mutual exclusion**. *For* $dme(t_1) = T_{dme}$ *with* $T_{dme} \subseteq T_T$ *we call each pair* $t_1 \in T_T$ *and* $t_x \in T_{dme}$ dynamically mutual exclusive tasks.

For Business Activity RBAC Models there are two different types of correctness that need to be considered: static correctness and dynamic correctness. *Static correctness* refers to the logical consistency of the elements and relationships in the Business Activity RBAC Model. It is static in the sense that it refers to the design-time of the model, i.e. it refers to process types and task types. *Dynamic correctness* relates to the compliance of process instances with the mutual exclusion and binding constraints at runtime. Thus, it is dynamic in the sense that it refers to the runtime execution of a particular process. Definition 2 provides the rules for static correctness.

Definition 2. Let *BRM* = (*E*,*Q*,*D*) be a Business Activity RBAC Model. *BRM* is said to be statically correct if the following requirements hold:

1. Mutual exclusion constraints between tasks define that a subject who owns (in case of a SME relation) or executes (in case of a DME relation) one of two mutual exclusive tasks must not own or execute the other task (see Section 2). Therefore: *Tasks cannot be mutual exclusive to themselves*:

$$\forall t_2 \in sme(t_1) : t_1 \neq t_2 \text{ and } \forall t_2 \in dme(t_1) : t_1 \neq t_2$$

2. As indicated by their name, mutual exclusion constraints are bidirectional. If, for example, the task "Negotiate contract" is defined as (dynamically or statically) mutual exclusive to the task "Approve contract" (see Fig. 2), the "Approve contract" task is also mutual exclusive to "Negotiate contract". Therefore we define the: *Mutuality of mutual exclusion constraints*:

$$\forall t_2 \in sme(t_1) : t_1 \in sme(t_2) \text{ and } \forall t_2 \in dme(t_1) : t_1 \in dme(t_2)$$

3. Binding constraints between tasks define that a subject or role who executes one bound task must also execute the other task (see Section 2). For example, if a binding is defined between the "Check credit worthiness" task and the "Negotiate contract" task (see Fig. 2), the subject or role executing an instance of the first task must also execute the corresponding instance of the second task (see also Definitions 1.7, 1.9, 1.10). Therefore: *Tasks cannot be bound to themselves*:

$$\forall t_2 \in sb(t_1) : t_1 \neq t_2 \text{ and } \forall t_2 \in rb(t_1) : t_1 \neq t_2$$

4. Similar to mutual exclusion constraints, binding constraints are bidirectional. If, for example, the task "Check credit worthiness" is bound to the task "Negotiate contract" (see Fig. 2), the "Negotiate contract" task is also bound to "Check credit worthiness". Therefore we define the: *Mutuality of binding constraints*:

$$\forall t_2 \in sb(t_1) : t_1 \in sb(t_2) \text{ and } \forall t_2 \in rb(t_1) : t_1 \in rb(t_2)$$

5. A SME constraint defines that two task types must never be assigned to the same subject, while a DME constraint defines that the instances of two task types must never be executed by the same subject (see Section 2). Thus, because a SME constraint completely prevents the assignment, and thereby implicitly the execution, of two mutual exclusive tasks through

the same subject, it is global with respect to all process instances. Thus, it is not sensible to define a SME and a DME relation between the same two tasks. Therefore: Tasks are either statically or dynamically mutual exclusive:

$$\forall t_2 \in sme(t_1) : t_2 \notin dme(t_1) \text{ and } \forall t_2 \in dme(t_1) : t_2 \notin sme(t_1)$$

6. SME constraints *conflict with* all types of *binding constraints* (subject-binding as well as role-binding). This is because a binding constraint defines that (in the context of the same process instance) the instances of two bound task types *must* be performed by the same subject respectively the same role, while a SME constraint defines that the instances of two statically mutual exclusive task types *must not* be performed by the same subject respectively the same role (see Section 2). It is impossible to fulfill both constraints at the same time. Therefore, it is *not* allowed to define a SME constraint *and* a binding constraint between the same two task types t_1 and t_2: Either SME constraint or binding constraint:

$$\forall t_2 \in sme(t_1) : t_2 \notin sb(t_1) \wedge t_2 \notin rb(t_1) \text{ and}$$
$$\forall t_2 \in sb(t_1) \cup rb(t_1) : t_2 \notin sme(t_1)$$

7. DME constraints and subject-binding constraints *conflict*. This is because a subject-binding constraint defines that (in the context of the same process instance) the instances of two bound task types *must* be performed by the same subject (the same individual). In contrast, a DME constraint defines that (in the context of the same process instance) the instances of two task types *must not* be performed by the same subject. It is not possible to fulfill both constraints at the same time. Therefore, it is *not* possible to specify a DME constraint *and* a subject-binding constraint between the same two task types t_1 and t_2:*Either DME constraint or subject-binding constraint*:

$$\forall t_2 \in dme(t_1) : t_2 \notin sb(t_1) \text{ and } \forall t_2 \in sb(t_1) : t_2 \notin dme(t_1)$$

Note that it *is* possible, however, to simultaneously define a role-binding constraint *and* a DME constraint on two tasks. This is because a DME constraint defines that (in the context of the same process instance) a subject *must not* own the instances of two dynamically mutual exclusive task types (see above). A role-binding constraint yet only defines that (in the context of the same process instance) the instances of two bound task types *must* be performed by the same *role*, not by the same subject/individual. This can, for instance, be achieved in the credit application process of Fig. 2 if there is a role binding and a DME constraint between the tasks "Check creditworthiness" and "Approve contract". This can be interpreted as a peer review (different subjects owning the same role). Therefore, DME constraints and role-binding constraints do *not* conflict.

8. Because each role owns the tasks that are directly assigned to this role as well as the tasks assigned to its junior-roles (see also Definitions 1.1 and 1.2), we have to make sure that no role can own two SME tasks, neither directly nor via a role hierarchy. Therefore, we must ensure the: *Consistency of task ownership and SME*:

$$\forall t_2 \in sme(t_1) : town^{-1}(t_2) \cap town^{-1}(t_1) = \emptyset$$

9. Subjects acquire the rights to perform certain task types via role-to-subject assignment relations (see Definition 1.3). However, it must never happen that a subject acquires the rights to perform two SME task types (see Section 2). This is because the role-to-subject assignment relations of a certain subject also define indirect/transitive relations between all task types assigned to this subject (see Definitions 1.1, 1.2, and 1.3). Therefore, the assignment of two SME task types to a certain subject must be prevented directly at design-time (in contrast to runtime checks which are performed on task instances). Otherwise, the transitive relation between two SME task types would violate the SME constraints, which would, again, result in an inconsistent and invalid model. Based on the relations defined at design-time (in particular, the task-to-role assignment and role-to-subject assignment relations), we can then perform the runtime allocation of task instances to subjects.[2] Therefore, we must ensure the: *Consistency of role ownership and SME*:

$$\forall t_2 \in sme(t_1), r_2 \in town^{-1}(t_2), r_1 \in town^{-1}(t_1) :$$
$$rown^{-1}(r_2) \cap rown^{-1}(r_1) = \emptyset$$

In addition to static correctness, Definition 3 provides the rules for dynamic correctness of a Business Activity RBAC Model, i.e. the rules that can only be checked in relation to existing task and process *instances*.

Definition 3. Let *BRM* = (E,Q,D) be a Business Activity RBAC Model and P_I its set of process instances. *BRM* 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 (see Section 2):

$$\forall t_2 \in sme(t_1), pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) :$$
$$es(t_x) \neq es(t_y)$$

We include this rule for the sake of completeness only, as the rule must always hold due to the consistency rule for role ownership and SME (see Definition 2.9).

2. In the same process instance, the executing subject of DME tasks must be different (see Section 2):

$$\forall t_2 \in dme(t_1), pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) :$$
$$es(t_x) \neq es(t_y)$$

3. In the same process instance, role bound tasks must have the same executing role (see Section 2):

$$\forall t_2 \in rb(t_1), pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) :$$
$$er(t_x) = er(t_y)$$

4. In the same process instance, subject bound tasks must have the same executing subject (see Section 2):

$$\forall t_2 \in sb(t_1), pi \in P_I : \forall t_x \in ti(t_2, pi), t_y \in ti(t_1, pi) :$$
$$es(t_x) = es(t_y)$$

3.2. Generic metamodel for Business Activity process flows

As we have now defined the structure of a Business Activity RBAC Model, we can formalize the generic flow metamodel. Fig. 5 shows the basic elements of Business Activity Process Flows and the main relations between these elements. Again, while this graphical model only gives an overview, we now provide a formal specification of the process flow model. This formalization defines a process flow model as a graph with specific types of nodes for actions and control elements, as well as arcs capturing the flow of control. The behavioural semantics are formalized as a labeled transition system [15,53]. Before a task can be completed, we have to check whether it is possible to find a subject who owns a corresponding role, and who is allowed to execute the task without vio-

[2] When we instantiate a task type, we check which subjects are allowed to execute the respective task type according to their roles. Next, we choose one of the subjects that can potentially be allocated to this task instance and actually allocate the task instance to the respective subject (see also Section 7). This subject becomes the executing-subject of the corresponding task instance. In contrast to SME constraints, DME constraints must be checked at runtime, i.e. in context of a specific process instance (see also [87], Section 2, and Definition 3.2).

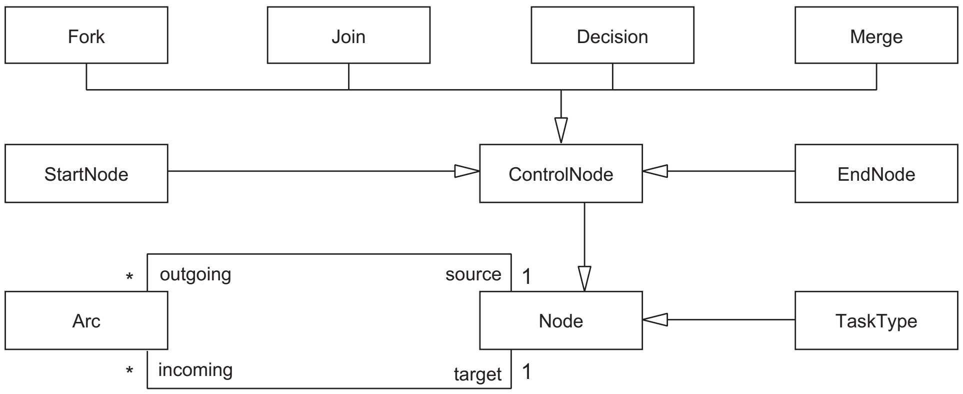


Fig. 5. Conceptual overview: main elements of Business Activity Process Flows.

lating mutual exclusion and binding constraints [87]. Therefore, our formalization has to explicitly take the execution history of a process instance into account – which is in contrast to most formalizations of process modeling languages which only cover the control flow.

Definition 4 (**Business Activity Process Flow Model**). A Process Flow Model $PFM = (N,A)$ where $N = T_T \cup C_F \cup C_J \cup C_D \cup C_M \cup \{start,end\}$ refers to pairwise disjoint sets and $A \subseteq N \times N$ such that

- An element of N is called node and an element of A is called arc.
- An element of T_T is called task type.
- An element of $C = C_F \cup C_J \cup C_D \cup C_M$ is called control node. An element of C_F is called fork, an element of C_J join, an element of C_D decision, and an element of C_M merge.
- *start* is called start node and *end* is called end node.
- All nodes $n \in N$ are on a path from *start* to *end*.

We will now define the dynamics of a process flow model based on its interplay with a Business Activity RBAC model. In this context, we need to explicitly refer to the execution history $h(p)$ of a process instance p. If the reference to a particular process instance is clear, we will simply write h instead of $h(p)$. This history is recorded in the Business Activity RBAC Model *BRM* since it has to reflect which subject has executed which task instance. Therefore, we now define the history of a process instance based on *BRM*.

Definition 5. Let $BRM = (E,Q,D)$ be a Business Activity RBAC Model and P_I its set of process instances. For a particular process instance $p \in P_I$, an execution event $exec(p) \in (T_I \times T_T \times R \times S)$ is a record of a particular task execution where T_I refers to the set of task instances, T_T to the set of corresponding task types, R to the set of executing roles, and S to the set of executing subjects. The execution history $h(p)$ of a process instance p is defined as a mapping $h : P_I \mapsto \mathscr{P}(\{(t_i,t_t,r,s) | t_i \in T_I, t_t \in T_T, r \in R, s \in S\})$, which maps $h(p)$ to a set of execution events $exec(p)$.

For a Business Activity Process Flow Model we define a notion of state, based on the distribution of control tokens on the arcs and the execution history. This resembles the state concept of [94]. Yet, our notion of history is richer in the sense that it covers also historic role assignments and subject assignments. We use letter m to refer to a state and letter M to refer the set of states, respectively. These letters are commonly used in Petri net formalizations, and are derived from the term "marking", which is used as a synonym for state in the Petri net context.

Definition 6 (**State of a Process Instance**). Let *PFM* be a Business Activity Process Flow Model and *BRM* a Business Activity RBAC Model such that the tasks of the first match the latter, that is $T_T^{PFM} = T_T^{BRM}$, and such that $a \in A$ are the arcs of *PFM*. The state of a process instance $p \in P_I$ is defined as a pair $m = (d,h)$ where h refers to the execution history of p and where d is an element of $D : A \mapsto \mathscr{N}$, which is a distribution of tokens on the arcs of the model with $\mathscr{N}$ being the natural numbers. The initial state $m_i = (d_i,h)$ of p is a state m such that

- $h = \emptyset$
- For each $a = (n_1,n_2)$ with $n_1 = start$:$d(a) = 1$.
- For each $a = (n_1,n_2)$ with $n_1 \neq start$:$d(a) = 0$.

Based on these definitions we can specify the behavioural semantics of a Business Activity RBAC Model. Similar to Petri nets, we define its behaviour using a notion of state and firing rules that can be used to construct the reachability graph. Yet, in our definition we do not use a Petri net translation but rather directly express state transitions in terms of a labeled transition system. This approach is typically used when formalizing complex behaviour in workflow languages that is difficult to express directly in terms of Petri net constructs. It is used, for instance, in the formalization of YAWL and of EPCs [2,37,47]. Formalizations via a mapping to Petri nets have been defined for EPCs [1], BPMN [19], and activity diagrams [21]. For an overview see [45].

For the definition, notationwise, we refer to the set of incoming and outgoing arcs of a node as follows: for each *node* $n \in N$, we define the set of incoming arcs $n_{in} = \{(x,n) | x \in N \wedge (x,n) \in A\}$, and the set of outgoing arcs $n_{out} = \{(n,y) | y \in N \wedge (n,y) \in A\}$. The result is a reachability graph definition in terms of a labeled transition system [46]. The transition relations of "fork" and "join" control nodes are visualized in the first row of Fig. 6: if each of the incoming arcs has a token, then a transition can be applied to a subsequent state where each outgoing arc receives a token. The "decision" and "merge" control nodes (the diamond shape symbols in Fig. 6) define a transition for every token that is coming through one of the input arcs to be propagated to exactly one of the output arcs. Most interestingly is the transition for a task node, depicted via the round-cornered rectangles at the bottom row of Fig. 6. It has to consider the availability of subjects with suitable roles, such that, given the execution history, none of the mutual exclusion and binding constraints are violated (see also Definition 3). Then, the state change can be applied and a new execution entry is added to the history.

A consequence of this specification is that if there is no subject-role combination that is allowed to execute a task at a particular

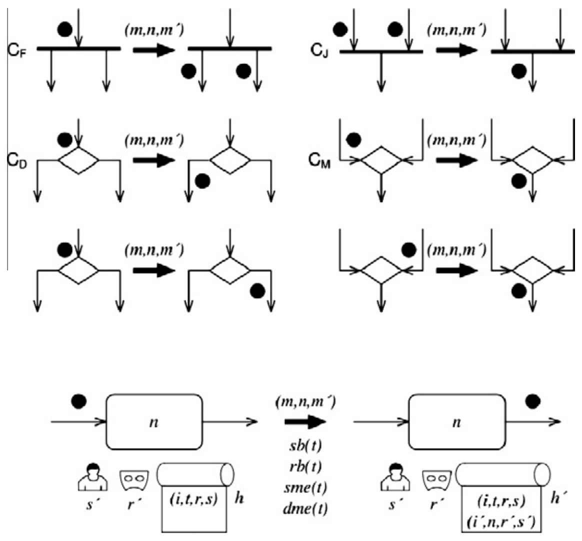


Fig. 6. Transition relations in a Business Activity Process Flow Model.

point in time, then the process instance will deadlock. This is an analysis feature that allows us to trace back the satisfiability problem to standard techniques for deadlock detection.

Definition 7 (**Reachability Graph**). Let $PFM = (N,A)$ be a Business Activity Process Flow Model with $N = T_T \cup C_F \cup C_J \cup C_D \cup C_M \cup \{start, end\}, BRM$ a Business Activity RBAC Model including T_I, $p \in P_I$ a process instance, and h the execution history of p. Then the Reachability Graph $RG = (M,TR)$ with $TR \subseteq M \times N \times M$ must comply with the following constraints:

1. The initial state is in M, i.e. $m_i \in M$ with $T_I = \emptyset$.
2. If $m = (d,h) \in M$ and $n \in C_F \cup C_J$ and for all $a \in n_{in}$:$d(a) > 0$ and
   $m' = (d',h')$ exists such that
   for all $a \in A \backslash (n_{in} \cup n_{out})$:$d'(a) = d(a)$,
   for all $a \in n_{in}$:$d'(a) = d(a) - 1$, and
   for all $a \in n_{out}$:$d'(a) = d(a) + 1$,
   then $m' \in M$ and $(m,n,m') \in TR$.
3. If $m = (d,h) \in M$ and $n \in C_D \cup C_M$ and for all $a \in n_{in}$:$d(a) > 0$ and $m' = (d',h')$ exists such that
   for all $a \in A \backslash (n_{in} \cup n_{out})$:$d'(a) = d(a)$,
   there exists an $a \in n_{in}$:$d'(a) = d(a) - 1$, and
   there exists an $a \in n_{out}$:$d'(a) = d(a) + 1$,
   then $m' \in M$ and $(m,n,m') \in TR$.
4. If $m = (d,h) \in M$ and $n \in T_T$ and for all $a \in n_{in}$:$d(a) > 0$ and there exists $i' \in ti(n,p), r' \in R, s' \in S$ with $n \in tra(r')$ and $r' \in rsa(s')$ such that for all $(i,t,r,s) \in h$ holds that:
   - if $n \in sb(t)$ then for all $ti \in ti(t,p)$:$s = es(ti) = es(i') = s'$, and
   - if $n \in rb(t)$ then for all $ti \in ti(t,p)$:$r = er(ti) = er(i') = r'$, and
   - if $n \in sme(t)$ then for all $ti \in ti(t,p)$:$s = es(ti) \neq es(i') = s'$, and
   - if $n \in dme(t)$ then for all $ti \in ti(t,p)$:$s = es(ti) \neq es(i') = s'$; and
     $m' = (d',h')$ exists such that
     for all $a \in A \backslash (n_{in} \cup n_{out})$:$d'(a) = d(a)$,
     there exists an $a \in n_{in}$:$d'(a) = d(a) - 1$, and
     there exists an $a \in n_{out}$:$d'(a) = d(a) + 1$, and
     $h' = h \cup \{(i', n, r', s')\}, T_I' = T_I \cup \{i'\}$ such that
     $i' \in ti(n,p)$, $es(i') = s'$, and $er(i') = r'$,
     and $m' \in M$ and $(m,n,m') \in TR$.

Proposition 1 (**Dynamic Correctness BRM**). *All BRM that correspond to states M of the reachability graph are dynamically correct.*

Proof. by induction:

Base Case: In the initial state holds $T_I = \emptyset$ such that all four properties of Definition 3 are fulfilled.

Induction: For transitions 7.2 and 7.3, BRM' does not change and therefore remains correct if BRM was also correct. For transition 7.4, BRM' remains correct by definition since (i',n,r',s') does not violate any constraints. □

Note that Proposition 1 has implications for the so-called satisfiability problem. This is the question whether it is guaranteed that there is always a role-subject pair that allows the process to proceed [17]. However, in this paper we are concerned with modeling support for process-related RBAC models and do not address the satisfiability problem. This is because, reachability graph analysis is already an exponential problem in the general case even without considering mutual exclusion and binding constraints, but different optimization strategies can be employed [93]. What we gain though is the capability of simulating process execution including dynamically correct role and subject allocation (see also [87]). In this way, the formalization serves as the generic foundation for a thorough workflow and RBAC specification for process modeling languages.

4. UML extension for Business Activities

In this section, we extend UML activity models so that they provide modeling support for Business Activities as defined in Section 3. Therefore, we define a new package *BusinessActivities* as an extension to the UML2 metamodel. Fig. 7 shows the metamodel of the BusinessActivities package, including all new modeling constructs defined in this package. In particular, we introduce BusinessActivity, BusinessAction, Subject, Role, RoleToSubjectAssignment, and RoleToRoleAssignment as new modeling elements.

A *BusinessActivity* is defined as a subclass of Activity (from the BasicActivities, CompleteActivities, FundamentalActivities, and StructuredActivities packages, see [61]). A *BusinessAction* is defined as a subclass of Action (from the CompleteActivities, FundamentalActivities, and StructuredActivities packages, see [61]) and as a subclass of Classifier (from Kernel, Dependencies, PowerTypes). Defining BusinessAction as a Classifier yields a number of advantages for our purposes. For example, it allows modelers to define specialized subtypes of BusinessAction with own specialized behavioural or structural features. Moreover, BusinessActions can be instantiated and, in contrast to ordinary UML Actions, the same instance can be used/executed multiple times in an activity (see also [61]). Among other things, this means that each instance of a BusinessAction may have its own state and history, for example including attributes to capture how often the action has been executed, which subjects and roles executed the action, etc. (see Sections 4.1 and 4.2).

With respect to the concepts defined in Section 3, *BusinessActivities* are applied to specify *process types* and *BusinessActions* define *task types*. In other words, BusinessActivities and BusinessActions define abstract type descriptions. These abstract processes and tasks can be instantiated as often as required. For example, we can define the general control and object flow of a credit application process in a bank via BusinessActions. The corresponding workflow and the included tasks can then be executed an arbitrary number of times to process incoming credit applications.

In the remainder of this section, we define OCL invariants for Business Activities (Section 4.1) and present a visual notation for our extension (Section 4.2). Subsequently, we illustrate its application with examples (Section 5).

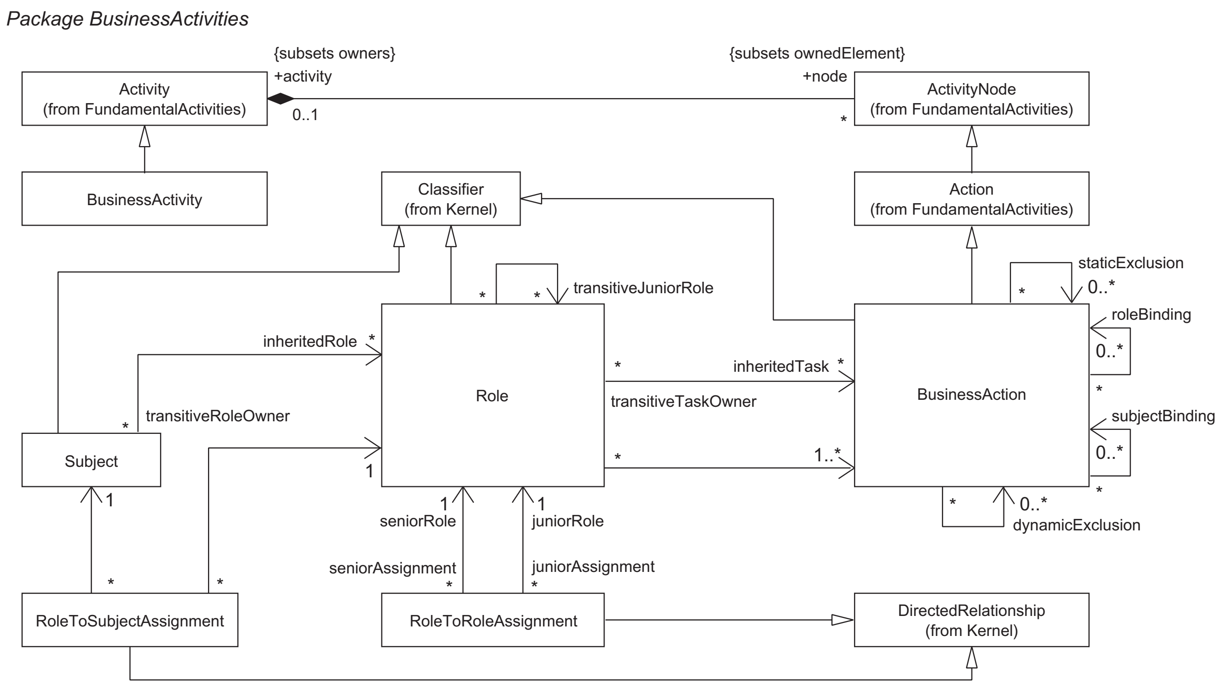


Fig. 7. UML metamodel extension for Business Activities.

4.1. Invariants for Business Activities

In accordance with Section 3, we now specify OCL invariants [58] on BusinessActivities and BusinessActions that ensure the correct semantics of models defined with our UML extension (see also [61]). In particular, these OCL invariants define the semantics of the corresponding graphical Business Activity models (see Section 4.2) and therefore make sure that the graphical models are correct and complete with respect to the formal specification from Section 3. However, for the sake of readability, this section only shows four OCL invariants as examples. The complete list of OCL invariants for the Business Activity extension is found in Appendix A.

OCL Constraint 1. *Each role may have direct and indirect/transitive junior-roles. In other words, if a role r has junior-roles and these junior-roles have junior-roles themselves, r inherits the indirect junior-roles as transitive junior-roles.*

```
context Role inv:
self.seniorAssignment->forAll(sa|
 sa.juniorRole.seniorAssignment->forAll(jrsa|
  self.transitiveJuniorRole->exists(tjr|
  tjr.name=jrsa.juniorRole.name)
  and
  jrsa.juniorRole.transitiveJuniorRole->forAll(
  jrtjr| self.transitiveJuniorRole->exists(tjr|
    tjr.name=jrtjr.name)
)))
```

OCL Constraint 9. *To assign subjects to an instance of a BusinessAction, and to determine the subject that executes a particular BusinessAction instance included in a particular BusinessActivity instance, we require that each BusinessAction defines an attribute called "executingSubject" (and thereby each instance owns a respective slot). Moreover, the executingSubject attribute must refer to a subject that is (via one of its roles) actually allowed to execute this BusinessAction.*

```
context BusinessAction inv:
self.instanceSpecification->forAll(i|
 i.slot->exists(s|
   s.definingFeature.name=executingSubject
 and
  (self.role->exists(r|
   r.roleToSubjectAssignment->exists(rsa|
     rsa.subject.name=s.value)
   or
   r.transitiveRoleOwner->exists(tro|
     tro.name=s.value))
)))
```

OCL Constraint 17. *In a role-hierarchy, mutual exclusion constraints are subject to inheritance (see Section 3). Therefore, two SME BusinessActions must never be assigned to the same role, neither directly nor transitively.*

```
context Role
inv:
self.businessAction->forAll(ba1,ba2|
  ba1.staticExclusion->select(sme|
    sme.name=ba2.name)->isEmpty()
)
inv:
  self.businessAction->forAll(ba|
    self.inheritedTask->forAll(it|
      ba.staticExclusion->select(sme|
      sme.name=it.name)->isEmpty()
)))
```

OCL Constraint 19. *To enforce SME constraints on BusinessActions, we specify that the instances of two SME BusinessActions must never have the same executing subject (see also Constraint 9 which specifies the requirement that each BusinessAction defines an attribute called "executingSubject"). For details on the InstanceSpecification element see [61].*

Table 1
Consistency of the generic metamodel and the UML2 extension for Business Activities.

Generic definitions	Covered through
Definition 1.1: $rh : R \mapsto \mathscr{P}(R)$ and $r \in R$: $rh^{*}(r) \cap \{r\} = \emptyset$	Metamodel extension: RoleToRoleAssignment (see Fig. 7), and OCL constraints 1 and 2
Definition 1.2: $tra : R \mapsto \mathscr{P}(T_T)$ and $town : R \mapsto \mathscr{P}(T_T)$	Metamodel extension: Association between Role and BusinessAction (see Fig. 7), and OCL constraint 3
Definition 1.3: $rsa : S \mapsto \mathscr{P}(R)$ and $rown : S \mapsto \mathscr{P}(R)$	Metamodel extension: RoleToSubjectAssignment (see Fig. 7), and OCL constraint 4
Definition 1.4: $ptd : P_T \mapsto \mathscr{P}(T_T)$	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7 and [61]); OCL constraints 5, 6, 7, and 8
Definition 1.5: $pi : P_T \mapsto \mathscr{P}(P_I)$	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7 and [61]); OCL constraints 5, 6, 7, and 8
Definition 1.6: $ti : (T_T \times P_I) \mapsto \mathscr{P}(T_I)$	OCL constraints 5, 6, 7, and 8
Definition 1.7: es: $T_I \mapsto S$	OCL constraint 9
Definition 1.8: er: $T_I \mapsto R$	OCL constraint 10
Definition 1.9: $sb : T_T \mapsto \mathscr{P}(T_T)$	Metamodel extension: subjectBinding association; recursive association on BusinessAction (see Fig. 7)
Definition 1.10: $rb : T_T \mapsto \mathscr{P}(T_T)$	Metamodel extension: roleBinding association, recursive association on BusinessAction (see Fig. 7)
Definition 1.11: $sme : T_T \mapsto \mathscr{P}(T_T)$	Metamodel extension: staticExclusion association, recursive association on BusinessAction (see Fig. 7)
Definition 1.12: $dme : T_T \mapsto \mathscr{P}(T_T)$	Metamodel extension: dynamicExclusion association, recursive association on BusinessAction (see Fig. 7)
Definition 2.1: $\forall t_2 \in sme(t_1)$: $t_1 \neq t_2$ and $\forall t_2 \in dme(t_1)$: $t_1 \neq t_2$	OCL constraint 11
Definition 2.2: $\forall t_2 \in sme(t_1)$: $t_1 \in sme(t_2)$ and $\forall t_2 \in dme(t_1)$: $t_1 \in dme(t_2)$	OCL constraint 12
Definition 2.3: $\forall t_2 \in sb(t_1)$: $t_1 \neq t_2$ and $\forall t_2 \in rb(t_1)$: $t_1 \neq t_2$	OCL constraint 13
Definition 2.4: $\forall t_2 \in sb(t_1)$: $t_1 \in sb(t_2)$ and $\forall t_2 \in rb(t_1)$: $t_1 \in rb(t_2)$	OCL constraint 14
Definition 2.5: $\forall t_2 \in sme(t_1)$: $t_2 \notin dme(t_1)$ and $\forall t_2 \in dme(t_1)$: $t_2 \notin sme(t_1)$	OCL constraint 15
Definition 2.6: $\forall t_2 \in sme(t_1)$: $t_2 \notin sb(t_1) \wedge t_2 \notin rb(t_1)$ and $\forall t_2 \in sb(t_1) \cup rb(t_1)$: $t_2 \notin sme(t_1)$	OCL constraint 16
Definition 2.7: $\forall t_2 \in dme(t_1)$: $t_2 \notin sb(t_1)$and $\forall t_2 \in sb(t_1)$: $t_2 \notin dme(t_1)$	OCL constraint 16
Definition 2.8: $\forall t_2 \in sme(t_1)$: $town^{-1}(t_2) \cap town^{-1}(t_1) = \emptyset$	OCL constraint 17
Definition 2.9: $\forall t_2 \in sme(t_1)$, $r_2 \in town^{-1}(t_2)$, $r_1 \in town^{-1}(t_1)$: $rown^{-1}(r_2) \cap rown^{-1}(r_1) = \emptyset$	OCL constraint 18
Definition 3.1: $\forall t_2 \in sme(t_1)$, $pi \in P_I$: $\forall t_x \in ti(t_2,pi)$, $t_y \in ti(t_1,pi)$: $es(t_x) \cap es(t_y) = \emptyset$	OCL constraint 19
Definition 3.2: $\forall t_2 \in dme(t_1)$, $pi \in P_I$: $\forall t_x \in ti(t_2,pi)$, $t_y \in ti(t_1,pi)$: $es(t_x) \cap es(t_y) = \emptyset$	OCL constraint 20
Definition 3.3: $\forall t_2 \in rb(t_1)$, $pi \in P_I$: $\forall t_x \in ti(t_2,pi)$, $t_y \in ti(t_1,pi)$: $er(t_x) = er(t_y)$	OCL constraint 21
Definition 3.4: $\forall t_2 \in sb(t_1)$, $pi \in P_I$: $\forall t_x \in ti(t_2,pi)$, $t_y \in ti(t_1,pi)$: $es(t_x) = es(t_y)$	OCL constraint 22
Definition 4: $PFM = (N,A)$	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7 and [61])
Definition 5: $h : P_I \mapsto \mathscr{P}(\{(t_i, t_t, r, s) \mid t_i \in T_I, t_t \in T_T, r \in R, s \in S\})$	Instantiation of metamodel extensions (see [61] and Fig. 7); OCL constraints 5, 6, 7, 8, 9, and 10
Definition 6: $m = (d,h)$	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7 and [61]); OCL constraints 5, 6, 7, 8, 9, and 10
Definition 7.1. 1: initial state	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7 and [61])

Table 1 (*continued*)

Generic definitions	Covered through
Definition 7. 2: token passing for fork and join nodes	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7 and [61])
Definition 7. 3: token passing for decision and merge nodes	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7 and [61])
Definition 7. 4: token passing and history for process and task types (BusinessActivities and BusinessActions)	Implicitly defined via our metamodel extension and the specification of UML activity models (see Fig. 7 and [61]); OCL constraints 5, 6, 7, 8, 9, 10, 19, 20, 21, and 22

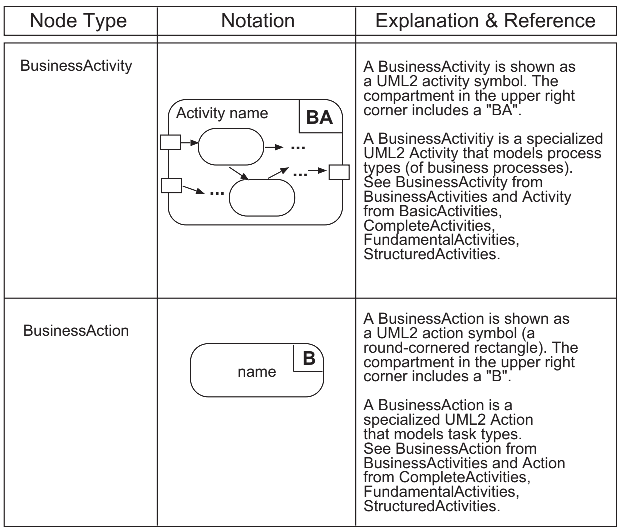
```
context BusinessAction inv:
self.staticExclusion->forAll(sme|
 self.instanceSpecification->forAll(i|
  sme.instanceSpecification->forAll(j|
  i.slot->forAll(is|
   j.slot->forAll(js|
    if is.definingFeature.name=executingSubject
       and
       js.definingFeature.name=executingSubject
    then
       not (is.value=js.value)
    endif
)))))
```

In this way, all definitions from Section 3 can be specified via corresponding OCL invariants. Table 1 gives an overview of how each of the generic definitions from Section 3 is mapped to our UML extension for Business Activities (see also Appendix A). Note that UML2 activity models include all types of control nodes needed for our extension (see Section 3). In addition, activity models have a token semantics and the sequencing of actions is controlled via control flow edges and object flow edges which carry control tokens and object tokens respectively (for details see [61]). In this respect, the abilities of UML2 activity diagrams go beyond the features that are required by our generic metamodel (see Section 3) which would only require control tokens. The consistency requirements induced by the OCL invariants can be then enforced in a software system to ensure the correctness of Business Activity models (see also Section 7).

4.2. Notation elements for Business Activities

Fig. 8 shows the notation elements for BusinessActivities and BusinessActions and Fig. 9 depicts two different presentation options for BusinessActions.[3] Fig. 9a shows two BusinessActions `Action1` and `Action2` that include a mutual exclusion constraint. To indicate mutual exclusion or binding constraints in the graphical representation, we add the names of corresponding BusinessActions to the graphical representation. The constraints are separated from the name of the corresponding BusinessAction via a dashed line (see Fig. 9). In particular, we use the following prefixes to define the different types of constraints:

- `SME`: indicates static mutual exclusion relations to other BusinessActions.
- `DME`: indicates dynamic mutual exclusion relations to other BusinessActions.
- `RBind`: indicates role-binding relations to other BusinessActions.
- `SBind`: indicates subject-binding relations to other BusinessActions.



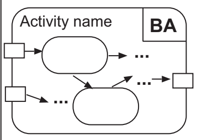
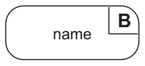
Node Type	Notation	Explanation & Reference
BusinessActivity		A BusinessActivity is shown as a UML2 activity symbol. The compartment in the upper right corner includes a "BA". A BusinessActivitiy is a specialized UML2 Activity that models process types (of business processes). See BusinessActivity from BusinessActivities and Activity from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities.
BusinessAction		A BusinessAction is shown as a UML2 action symbol (a round-cornered rectangle). The compartment in the upper right corner includes a "B". A BusinessAction is a specialized UML2 Action that models task types. See BusinessAction from BusinessActivities and Action from CompleteActivities, FundamentalActivities, StructuredActivities.

Fig. 8. Notation elements for BusinessActivities and BusinessActions.

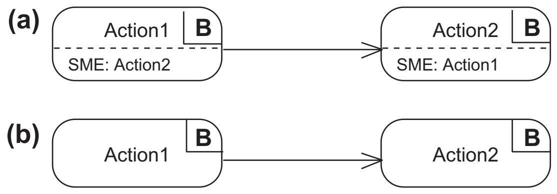


Fig. 9. Presentation options for BusinessActions.

However, it is important to understand that the graphical symbols presented in Fig. 9 are only *presentation options* to visualize the relations that are formally defined through our UML metamodel extension and the corresponding OCL constraints (see Section 4 and Appendix A). This means that a mutual exclusion constraint or a binding constraint that is defined on two BusinessActions exists *independent* of its visualization in the graphical model. Fig. 9b elides the constraint section shown in Fig. 9a. For example, a modeling tool can allow modelers to switch between both presentation options.

[3] Note that the use of different presentation options is common for UML2 modeling elements (for details see [61]).

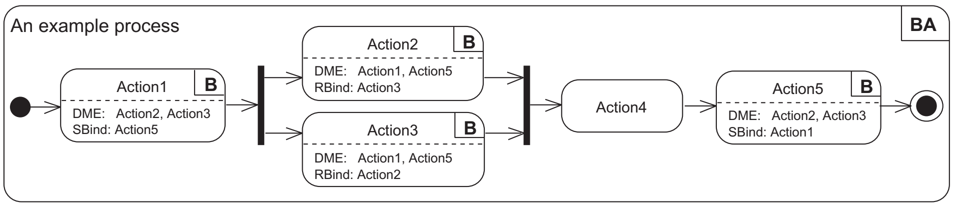


Fig. 10. An example process modeled as BusinessActivity.

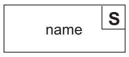
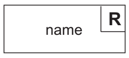
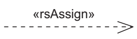
Node Type	Notation	Explanation & Reference
Subject		A Subject is shown as a rectangle. The compartment in the in the upper right corner includes a "S". See Subject from BusinessActivities.
Role		A Role is shown as a rectangle. The compartment in the in the upper right corner includes a "R". See Role from BusinessActivities.
RoleToSubjectAssignment		A RoleToSubjectAssignment relation is shown as a dashed arrow with the «rsAssign» keyword attached to it. See RoleToSubjectAssignment from BusinessActivities.
RoleToRoleAssignment		A RoleToRoleAssignment relation is shown as a dashed arrow with the «rrAssign» keyword attached to it. The arrowhead is a triangle including an "J" to indicate the end of the relation that points to the junior-role. See RoleToRoleAssignment from BusinessActivities.

Fig. 11. Notation elements for Subjects and Roles.

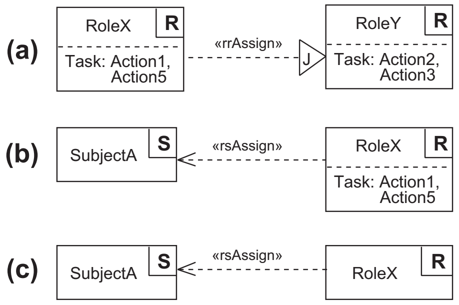


Fig. 12. Presentation options for Roles.

Fig. 10 shows an example process modeled as a BusinessActivity. In particular, this process includes five actions, four of which are BusinessActions. We recall that a BusinessActivity is a specialized UML Activity (see Fig. 7) and therefore may include any modeling element that can be included in UML activities, such as ordinary UML actions or fork, join, and decision nodes. In the example, `Action1` and `Action5` define a DME constraint to `Action2` and `Action3`, indicated through the `DME` entries in the corresponding BusinessAction symbols. This means that the example in Fig. 10 defines that in each instance of the example process the subject performing `Action1` and `Action5` must be distinct from the subject performing `Action2` and `Action3`.

In addition, `Action1` defines a subject binding to `Action5`, which means that in each instance of the example process `Action1` and `Action5` must be performed by the same subject. Finally, there is a role-binding between `Action2` and `Action3`, which means that both actions must be executed by a member of the same role.

As defined above, all these relations are mutual. Thus, each mutual exclusion or binding relation is shown in each of the corresponding BusinessActions (cf. Fig. 10).

Fig. 11 depicts the notation elements for Subjects, Roles, SubjectToRoleAssignment, and RoleToRoleAssignment relations. Fig. 12 shows different presentation options for Roles. Fig. 12a shows two roles RoleX and RoleY that are connected through a RoleToRoleAssignment relation. In the example, the arrowhead of this RoleToRoleAssignment relation points to RoleY, indicating RoleY as junior-role of RoleX (cf. Fig. 11).

Moreover, we add the names of BusinessActions that are assigned to a particular role to the role symbol, see Fig. 12a and b. In the same style we apply for BusinessActions, this task list is separated from the respective role name via a dashed line. Again, remember that this is just a presentation option to visualize the relations of roles and associated BusinessActions. The association of a particular role to its BusinessActions, however, exists independent of this visualization in the graphical model. Fig. 12c shows the example from Fig. 12b with elided task associations.

Fig. 13 shows an example of a role-hierarchy. In particular, we see seven roles (RoleA to RoleG) in two role-hierarchies. The first hierarchy consists of only two roles, where RoleE is defined as senior-role of RoleA. The second hierarchy consists of five roles and RoleF is defined as direct senior-role of RoleB and RoleC, while RoleG is defined as direct senior-role of RoleD and RoleF. Thereby RoleG is the most powerful role in this role-hierarchy. That is, in addition to its own BusinessActions and the BusinessActions associated with its direct junior-roles, RoleG transitively inherits the privilege to perform all BusinessActions associated with RoleB and RoleC (cf. Sections 3 and 4).

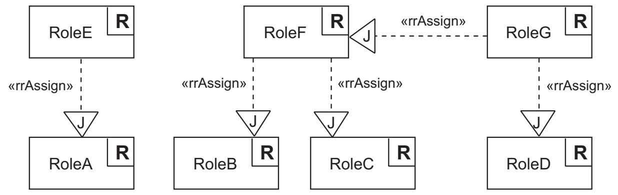


Fig. 13. Example of a role hierarchy.

5. Business Activity examples

In this section, we show three examples of how Business Activities are used to define process-related RBAC models (including mutual exclusion constraints and binding constraints). Because each Business Activity model must conform to the OCL invariants defined in Section 4.1 and Appendix A, the graphical representation can be reduced to the essential modeling-level information without bothering the process modeler with such formal constraints.

This means that the OCL invariants guarantee that the processes and process-related RBAC models defined with our Business Activity extension are correct and conflict-free with regard to the formal definition given in Section 3 (see Table 1). To actually ensure the consistency of the OCL constraints for Business Activities in a software system, the respective constraints must be checked and

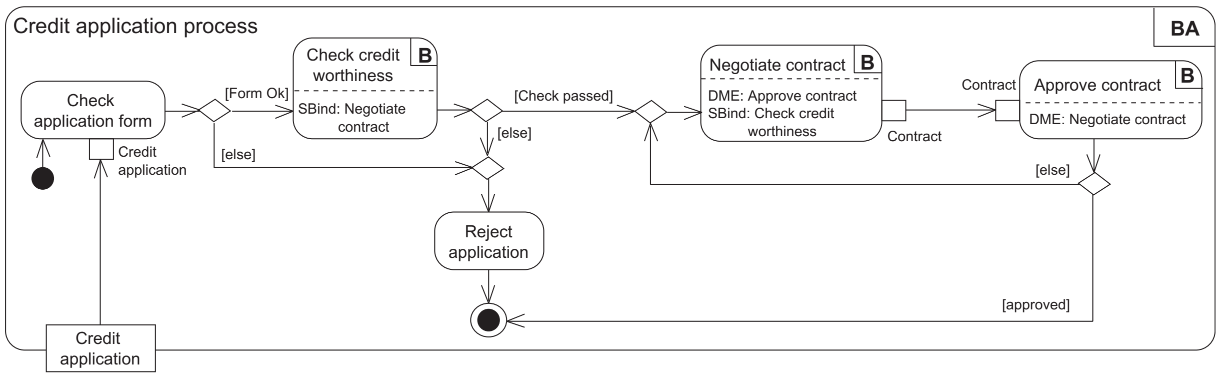


Fig. 14. A credit application process modeled as a BusinessActivity.

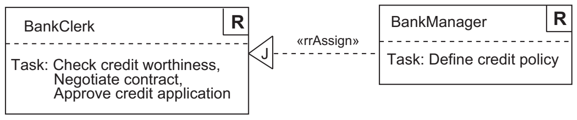

BankClerk
R
Task: Check credit worthiness, Negotiate contract, Approve credit application
«rrAssign»
BankManager
R
Task: Define credit policy

Fig. 15. Two roles for the credit application example.

enforced at the implementation level. In other words, the OCL invariants make sure of the consistency and correctness of the models if these constraints can be enforced by software tools. For this reason, we implemented a Business Activity library and runtime engine that automatically enforces all invariants defined in Appendix A, and ensures the design-time as well as runtime consistency of Business Activities in an actual software system (see Section 7).

Aside from the small-scale examples presented below, Section 6 describes our experience with applying Business Activities to processes and process-related RBAC models from actual projects.

5.1. A credit application process

Fig. 14 depicts a BusinessActivity that models the credit application process from Fig. 2. The process includes five actions, three of which are BusinessActions. The process defines a DME constraint between the "Negotiate contract" and "Approve contract" actions. This constraint is defined to enforce the four-eyes-principle for the corresponding tasks, to ensure that each credit application is processed by two different bank clerks. Moreover, we define a subject-binding between the "Check credit worthiness" and the "Negotiate contract" tasks to ensure that these tasks are performed by the same subject. The subject binding is defined so that the subject who is already familiar with a particular application also negotiates the contract with the corresponding customer.

Fig. 15 shows two roles modeled for the credit application process. The BankClerk role is permitted to perform the "Check credit worthiness", "Negotiate contract", and the "Approve credit application" actions. Moreover, we have a BankManager role which is defined as senior-role of BankClerk and thereby inherits the corresponding permissions. Additionally, a BankManager is allowed to conduct the "Define credit policy" action. This task (not included in the process depicted in Fig. 15) allows a bank manager to define the basic parameters and conditions that each credit application and each contract must comply with.

5.2. A paper review process

Fig. 16 shows an example of a review process for research papers submitted to a scientific conference. The process includes four BusinessActions and one ordinary action. For each paper submitted to the conference, an instance of this process is executed. The first BusinessAction is the "Submit paper" action. It is defined as dynamically mutual exclusive to the "Paper review" and the "Make decision" actions. In other words, we define that a subject cannot submit a paper and write a review, or make an acceptance decision, for this very paper. However, because DME constraints refer to process instances (see Sections 3 and 4) it is possible that a particular subject submits a paper, and acts as an author for this paper, while acting as a reviewer for another paper (in another instance of the paper review process).

Fig. 17 shows the three roles required for the review process. Note that in this example, the roles do not have inheritance relations (i.e. RoleToRoleAssignment relations). Such standalone roles are defined for environments where we have clearly separated functional areas that not hierarchically structured.

5.3. A radiological image reading process

Fig. 18 shows a reading process for radiological images modeled as a BusinessActivity. At first, a radiological examination is conducted to produce radiological images (e.g. via a CT scan). Next, the "Image reading" action is conducted to determine if the image quality is appropriate, and if so, to produce a radiological finding. Subsequently, the "Write report" action is performed to write a radiological report for the images. Finally, the report has to be validated. In case the report includes errors or is incomplete, it must be revised before it is resubmitted for validation.

We define a subject binding between the "Image reading" action and the "Write report" action (see Fig. 18). This is sensible to ensure that the same radiologist who assessed the images in the "Image reading" action also writes the radiological report on these images. Moreover, to enforce the four-eye-principle on radiological reports, we define a DME constraint between the "Write report" and the "Report validation" actions. This means, for any radiological report these actions must always be conducted by two different subjects. This is an essential quality and safety measure to guard against mistakes and malpractices.

Fig. 19 shows the roles required for the image reading process. The Radiologist role is permitted to perform the "Radiological examination", "Image reading", and "Write report" actions. The SeniorRadiologist role is defined as senior-role of Radiologist and thereby inherits the permission to perform the three tasks. In addition, members of the SeniorRadiologist role are permitted to perform the "Report validation" action. This means that at least one SeniorRadiologist must participate in each instance of the image reading process. However, due to the DME constraints on the "Write report" and "Report validation" tasks, no subject can complete the process by herself. In other words, even if a member of

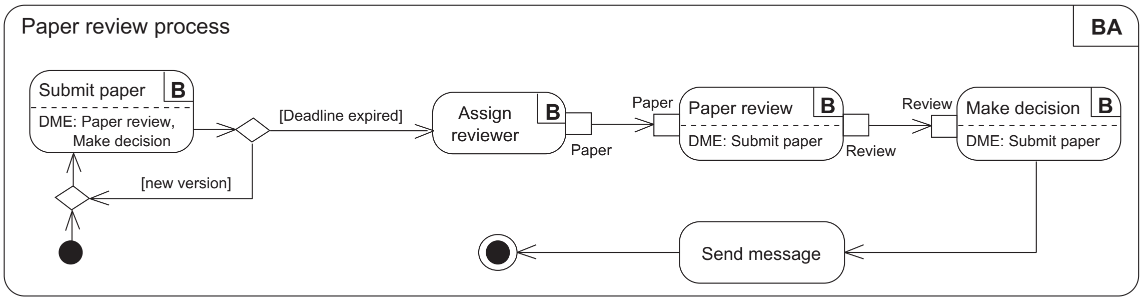


Fig. 16. A paper review process modeled as a BusinessActivity.

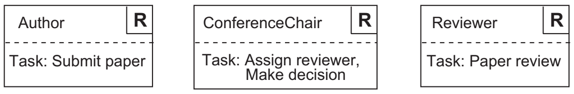


Fig. 17. Roles for the paper review example.

the SeniorRadiologist role performs the "Image reading" and "Write report" actions, the DME constraint requires that another subject (acting as SeniorRadiologist) performs the "Report validation" action (see Fig. 18).

6. Using Business Activities for real-world process models

Scenario-driven role engineering is a systematic approach for the elicitation, specification, and maintenance of RBAC models. We are continuously conducting role engineering projects and gained much experience in this area (see, e.g., [39,54,84,86]). For example, in 2008 we conducted a role engineering project with the Austrian Federal Ministry of Finance,[4] in 2009 we conducted a corresponding case study with the German branch of ABB,[5] in 2009/10 we were involved in a rights management project with Ernst & Young,[6] and currently we are conducting a role engineering project with the Vienna City Municipality.[7] In each of these (as well as in other) projects we identified the urgent need for modeling support of process-related RBAC models (see also Section 1.1). While we have non-disclosure agreements with our project partners, and are therefore not allowed to discuss business secrets from these projects, we can discuss our experience and describe how our Business Activities help to resolve different issues we identified in these projects.

In one of our projects, most processes/procedures were defined via purely textual descriptions. Some of these descriptions were written as prose/running text while others were written as lists of enumerated text descriptions. In addition, few processes were specified via proprietary or informal graphical models. Mutual exclusion and binding constraints were either specified as part of the textual process descriptions, or they were defined in a supplemental textual document (such as "task 1 and task 3 should be conducted by the same individual", or "task 1 and task 6 must be conducted by different individuals"). For example, the description of a technical support workflow at a "Helpdesk" workplace included a number of binding constraints (role-binding as well as subject-binding constraints). In particular, the binding constraints defined that certain tasks in a certain instance of the support workflow must be conducted by the same Helpdesk employee or a member of the same role. However, due to the textual and informal nature of the process descriptions, the support workflow was difficult to follow and to learn. We modeled the process as a Business Activity which included all process and constraint information in a single graphical model. Thereby, the Business Activity significantly enhanced the understandability of the process and provided a consistent and complete graphical description.

In another project, the processes were modeled via standard UML activity models. In addition, for each process a table was defined where the rows and columns consisted of the process' task names, respectively. The cells of these tables then included the mutual exclusion and binding constraints between different tasks. However, while these tables were already complex and required some effort to be matched with the corresponding process description, this complexity further increased for larger process models because each task in the process model resulted in an additional row and an additional column in the respective constraint table. In this context, our Business Activities provide a lightweight and easy to use modeling approach for the integrated specification of process descriptions and corresponding mutual exclusion and binding constraints. Thus, they consolidate process descriptions and corresponding constraint tables and thereby render the tables obsolete.

In a third project, the respective organization's business processes were defined using many different formats, including textual descriptions and different types of graphical models. Additional information, such as mutual exclusion and binding constraints, were defined using informal textual notes that were attached to the respective process descriptions. In addition, some processes were not documented at all but only existed as part of the "organizational knowledge" of certain employees. Moreover, because the respective organization did not have stringent maintenance procedures for their business processes, many process descriptions were outdated, at least partially. In this situation, Business Activities are used as a means to define a complete and coherent set of process descriptions – including corresponding roles and constraints. Moreover, because Business Activities are a native UML extension they allow for a consistency check with software systems that are modeled in UML, and facilitate the impact and effort estimation for the adaption/extension of an existing software system. In other words, in case an existing software system is modeled using UML, we can identify the components or subsystems that are participating in the execution of a certain business process (modeled via the Business Activities extension). Thereby, we can also identify the components or sub-systems that are potentially affected when integrating/implementing the tailored RBAC model which is defined via the Business Activities extension. Once we know which components or sub-systems are potentially affected, we can estimate the effort and the impact that would result from the implementation/integration of the tailored RBAC model.

[4] http://english.bmf.gv.at/.
[5] http://www.abb.com/.
[6] http://www.ey.com/.
[7] http://www.wien.gv.at/ma14/.

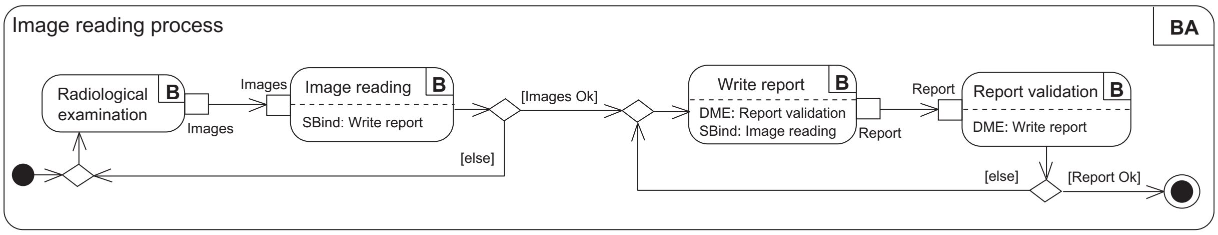


Fig. 18. A reading process for radiological images modeled as a BusinessActivity.

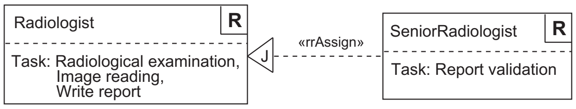


Fig. 19. Roles for the image reading example.

From our experience in role engineering projects and case studies, we can conclude that Business Activities are a suitable means to model process-related RBAC models. We applied our extension to produce project-specific Business Activity models. This modeling was conducted in cooperation with domain experts from the respective projects. In particular, we discussed the Business Activity models with domain experts and used their feedback for an iterative refinement of the models. In addition, the domain experts did also suggest to keep the graphical notation as lightweight as possible. This was one of the reasons why we defined the notation elements for BusinessActivity and BusinessAction as variants of the original Activity and Action symbols including a BA and a B in the upper right corner respectively (see Section 4.2). The domain experts found the BusinessActivity models to be well-suited for their purposes and the notation straightforward to learn.

Business Activities enable a consistent and complete description of processes and can be integrated with other types of UML models. Because the OCL invariants "in the background" enforce the consistency of Business Activity models, we can provide a lightweight graphical syntax that is reduced to the essential modeling-level information – without bothering the process modeler with such formal constraints. Thereby Business Activity models can serve as a primary communication vehicle and help bridge the gap between experts from the application domain (such as bank managers, power plant engineers, or physicians) on the one hand, as well as software engineers and security engineers on the other.

7. Platform support for Business Activities

In this section, we give an overview of our platform support for Business Activities. Section 7.1 first presents the Business Activity library and runtime engine. Subsequently, Section 7.2 gives an overview of the mapping of modeling-level Business Activities to runtime models.

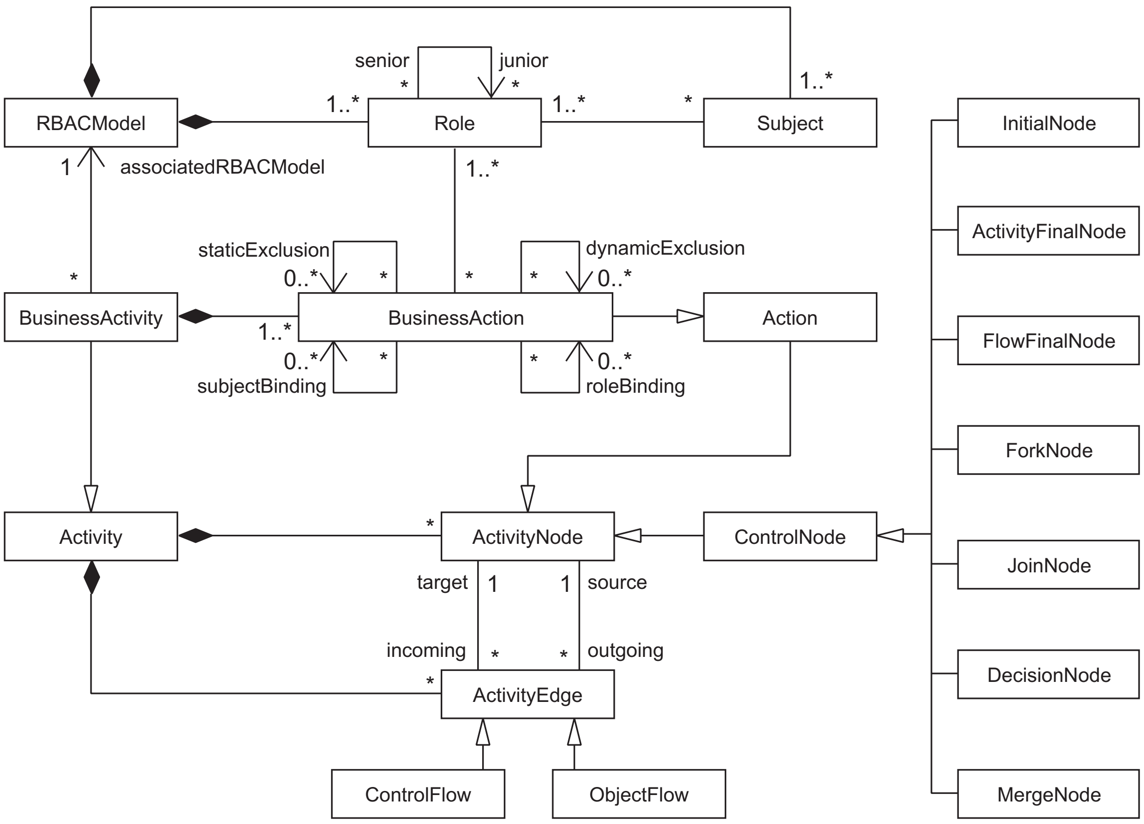


Fig. 20. Class model of the Business Activity library and runtime engine (excerpt).

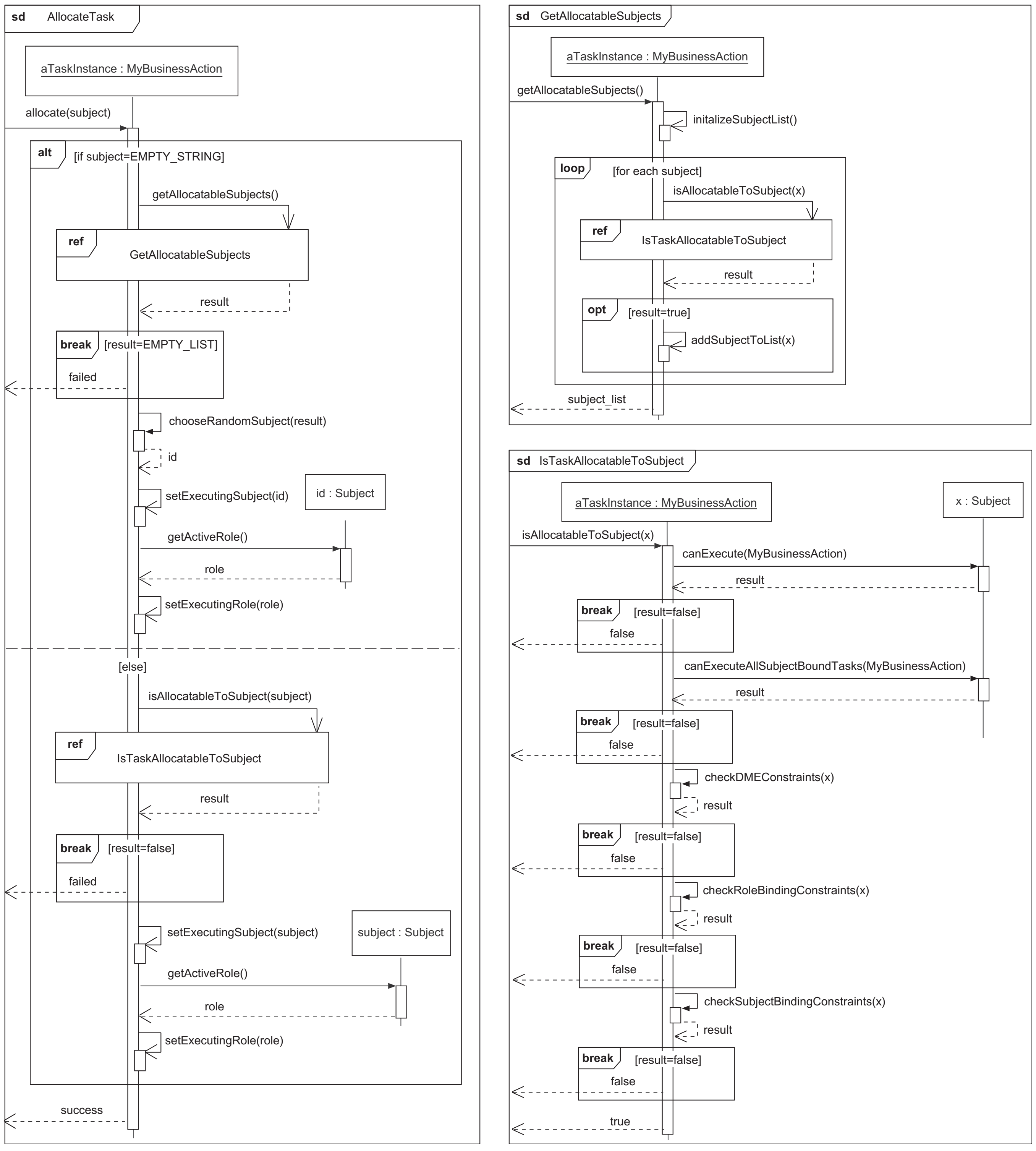

sd AllocateTask
aTaskInstance : MyBusinessAction
allocate(subject)
alt [if subject=EMPTY_STRING]
getAllocatableSubjects()
ref GetAllocatableSubjects
result
break [result=EMPTY_LIST]
failed
chooseRandomSubject(result)
id
setExecutingSubject(id)
id : Subject
getActiveRole()
role
setExecutingRole(role)
[else]
isAllocatableToSubject(subject)
ref IsTaskAllocatableToSubject
result
break [result=false]
failed
setExecutingSubject(subject)
subject : Subject
getActiveRole()
role
setExecutingRole(role)
success
sd GetAllocatableSubjects
aTaskInstance : MyBusinessAction
getAllocatableSubjects()
initalizeSubjectList()
loop [for each subject]
isAllocatableToSubject(x)
ref IsTaskAllocatableToSubject
result
opt [result=true]
addSubjectToList(x)
subject_list
sd IsTaskAllocatableToSubject
aTaskInstance : MyBusinessAction
x : Subject
isAllocatableToSubject(x)
canExecute(MyBusinessAction)
result
break [result=false]
false
canExecuteAllSubjectBoundTasks(MyBusinessAction)
result
break [result=false]
false
checkDMEConstraints(x)
result
break [result=false]
false
checkRoleBindingConstraints(x)
result
break [result=false]
false
checkSubjectBindingConstraints(x)
result
break [result=false]
false
true

Fig. 21. Business Activity runtime engine: allocating tasks to subjects.

7.1. The Business Activity library and runtime engine

In addition to the modeling level support introduced in Section 4, we implemented a software platform that can manage corresponding runtime models and enforce the different policies and constraints at the application level. Fig. 20 shows the essential class relations of our library and runtime engine for Business Activities. For the sake of simplicity, we only show an excerpt of the class model including the classes and class relations which are most relevant for the purposes of this paper. Our software platform provides *complete support* for all modeling level elements defined in Section 4 and automatically enforces all invariants defined via OCL constraints. Furthermore, it ensures the compliance of processes modeled via Business Activities and user-defined mutual exclusion and binding constraints. Thus, it was developed to support a straightforward mapping of modeling level Business Activities to corresponding runtime models (see Section 7.2).

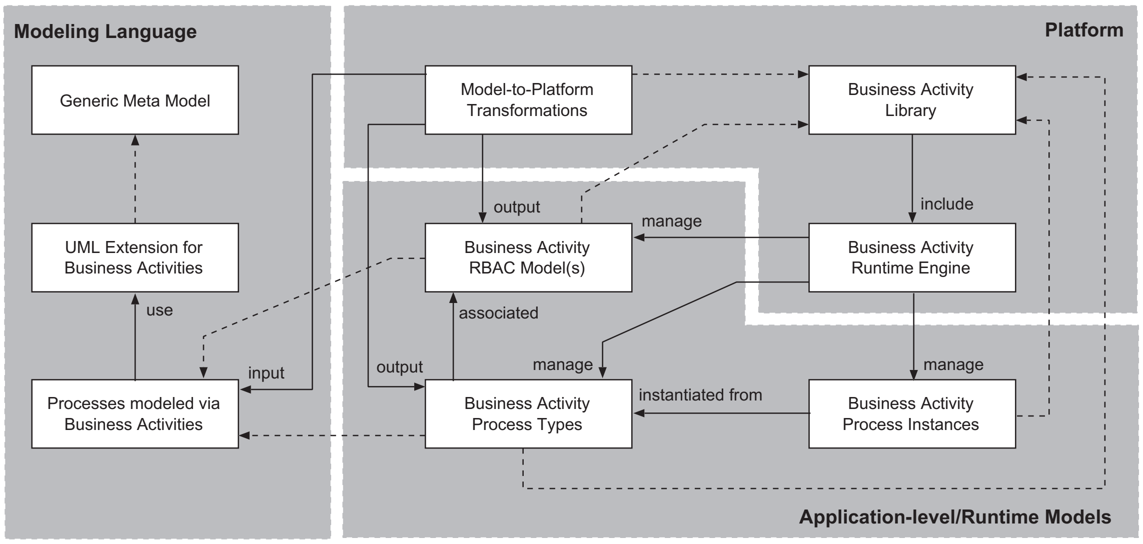

Modeling Language
Platform
Application-level/Runtime Models
Generic Meta Model
UML Extension for Business Activities
Processes modeled via Business Activities
Model-to-Platform Transformations
Business Activity Library
Business Activity RBAC Model(s)
Business Activity Runtime Engine
Business Activity Process Types
Business Activity Process Instances
use
input
output
output
associated
manage
manage
manage
include
instantiated from

Fig. 22. Modeling and runtime support for Business Activities: conceptual overview.

In addition to platform support for the different modeling level artifacts, the Business Activity library and runtime engine also provide functions to facilitate the management of corresponding runtime models. One important feature for the management of Business Activity runtime models is task allocation, i.e. the assignment of a certain task instance to a certain subject. In particular, the allocation of task instances included in a specific process instance must be consistent with the associated RBAC model as well as the mutual exclusion and binding constraints defined for the corresponding process type (see [87], as well as Sections 3 and 4).

Fig. 21 shows UML interaction models that describe our implementation of the task allocation concern in detail. When an `allocate` call reaches a particular task instance (i.e. an instance of a particular BusinessAction) the corresponding task instance first checks if the subject parameter is empty or if it includes a subject name. If the subject parameter is empty (i.e. it includes the empty string), it calls the `getAllocatableSubjects` method to determine the list of all subjects that could potentially be allocated to this task. The `getAllocatableSubjects` method, again, checks if a subject can execute the corresponding task type (according to its role), and if mutual exclusion or binding constraints exist which prevent an allocation to a particular subject. Subsequently, the method returns the list of allocatable subjects. If the subject list is empty (i.e. none of the subjects can be allocated to the task instance), the allocation procedure is stopped and a "failed" message is returned (see the "break" operator in the AllocateTask model of Fig. 21). In case the subject list is not empty, however, the `allocate` method randomly chooses a subject from the list and sets the executing-subject property (i.e. the owner of the task instance) and the executing-role property accordingly and thereby allocates the task instance to this subject (see also Sections 3 and 4).

In case the subject parameter of the initial `allocate` invocation is not empty, the `allocate` method first checks if this particular subject is allocatable to the corresponding task instance (see the "[else]" guarded operand in the AllocateTask model of Fig. 21). If the subject is not allocatable to this task instance, the procedure is stopped and a "failed" message is returned. Otherwise, the method allocates the subject by setting the executing-subject and executing-role properties accordingly (see Fig. 21).

Thus, before task allocation, a subject just owns a role which defines the subject's (abstract) capability of performing tasks (BusinessActions) of certain types. However, the subject neither has a concrete authorization nor an obligation to execute a certain task *instance*. Then, on task allocation, a certain subject is assigned to a particular task instance.

Following this approach we directly enforce the principle of least privilege and we assure that, at any time, each subject has the exact set of permissions it needs to perform its tasks. This means no subject s_i can access a task instance that is allocated to another subject s_j, neither accidently nor on purpose. Thus, when using our runtime engine one just has to call the `allocate` method to allocate a task instance to a particular user. This can be done manually (e.g. by some supervisor) as well as automatically (e.g. as part of an automated workflow escalation management). Depending on the result of the allocation procedure, the `allocate` method either returns a "success" message or a message indicating why the allocation failed (see UML interaction models in Fig. 21).

7.2. Mapping Business Activities to runtime models

Fig. 22 shows an informal overview of the main conceptual entities residing on different abstraction levels (namely modeling language, software platform, and runtime models). The metamodel defined in Section 3 is generic and can be used to extend modeling languages with corresponding (native) language abstractions. The UML extension presented in Section 4 enables the modeling of Business Activities in a widely used standard modeling language. The modeling level part of our approach is shown on the left-hand side of Fig. 22.

Actual Business Activities defined with our UML extension are platform-independent and can (in principle) be mapped to arbitrary software platforms. However, if the corresponding target platform does not provide all features that are needed to support Business Activities, it is likely that the intended target platform must be extended first [81,89]. To enable a continuous support for Business Activities, we implemented a corresponding software platform that supports all facets of the approach (see Section 7.1). This part of our approach is shown in the upper right compartment of Fig. 22. In particular, we use model transformations [50,75,81,102] to map UML2 Business Activities to runtime models that are managed by our library and runtime engine (see Section 7.1).[8]

[8] Typically, transformations are conducted by a special purpose generator component. The generator component receives models as input and produces platform-specific generated artifacts as output [81,89]. In order to do this correctly, the generator "knows" the source metamodel and the target metamodel, as well as the target platform. However, adding the generator entity to Fig. 22, would significantly complicate Fig. 22 without providing any added value (from a conceptual point of view).

Table 2
Modeling process-related RBAC models: related work.

	Business Processes	Roles	Role hierarchies	SME tasks	DME tasks	Task-based role binding	Task-based subject binding	Runtime support
Approaches to provide modeling support for process-related RBAC models								
UML-based representation of RBAC [4,76,78]		○	○	○				○
Towards UML-based role engineering [20]		○						
Using UML to visualize RBAC constraints [65]		√	○	○	○			
UML2 profile for security requirements [66]	√	√						
BPMN extension for security requirements [67]	√	√						
Task-based constraints for BPMN [100]	√	√			√		√	
Model-driven approaches using RBAC as an illustrative example								
CRBAC-a model-driven approach [5]	○	√	√					√
Model-driven security [7]	√	√	√					√
AC spec. using MOF and UML profiles [25]		√	○	○				○
UML specification of access control policies [38]		√	○					
Business Activities (our approach)	√	√	√	√	√	√	√	√

The Business Activity runtime engine then manages the corresponding runtime models of user-defined process types and associated RBAC models (see lower right compartment of Fig. 22). Via the runtime engine's application programming interface (API), runtime Business Activities can then be integrated with other software components. For the sake of simplicity, other UML models and other application logic are not included in Fig. 22.

8. Related work

In general, we distinguish two types of related work for this paper. First, we have approaches that primarily aim to provide modeling support for process-related RBAC models. We see such approaches as related work in the narrower sense. Second, a number of different approaches exist that use methods and techniques from model-driven development to include security in the software development process. Some of these approaches use examples from the domain of (role-based) access control for demonstration purposes. However, in these other approaches the main focus is on different aspects of model-driven development, rather than on the specification of a modeling approach for business processes and process-related RBAC models. Thus, we see such approaches as related work in a broader sense. Moreover, many of the model-driven approaches are complementary to our work and are well-suited to be combined with our Business Activities.

Table 2 shows an overview of related work on modeling of process-related RBAC models. With respect to the concepts specified in Sections 3 and 4, we use a √ if a related approach provides similar or comparable support for a certain concept, and a ○ if a related approach provides at least partial support for a particular concept.

Many related approaches use pure UML or UML profiles to provide a certain level of modeling support for (role-based) access control. For example, in UML Activity diagrams the swimlane notation is used to visualize the (hierarchical) partitioning of ActivityPartition elements. Because ActivityPartitions often correspond to certain actors or organizational units, they can be used to *visualize* a certain aspect of mutual exclusion. However, using UML ActivityPartitions to visualize mutual exclusion would rely on an informal interpretation of a graphical convention. Thus, because the ActivityPartition element [61, page 340] is not expressive enough to represent mutual exclusion constraints, such a visualization of mutual exclusion constraints would entirely rely on the human interpretation of a graphical model or some informal textual comment attached to a model.

UML profiles provide a mechanism for the extension of existing UML metaclasses to adapt them for non-standard purposes. However, UML profiles are not a first-class extension mechanism [61, page 654]. They extend *existing* metaclasses of the UML metamodel and the extension defined through a profile must be consistent with the semantics of the extended (original) UML metaclasses. For example, if we extend the UML metaclass "Class" via a profile, the extended Class will still be a Class. In particular, it is forbidden to insert new metaclasses in the UML metaclass hierarchy via profiles [61, page 654]. Thus, extensions that are based on UML profiles all reach their limit if the extension goes beyond a certain level of complexity. In particular, profiles are not suited for more complex extensions that require the definition of specialized modeling elements with specialized semantics.

For this reason, more complex extensions are defined via UML metamodel extensions [60,61,81]. An extension of the UML metamodel allows to define *new* and specifically tailored UML elements (defined via new metaclasses), and allows to define a customized notation, syntax, and semantics for the new modeling elements. Because an extension for process-related RBAC models (including mutual exclusion and binding constraints) clearly goes beyond the purpose of UML profiles, we chose to demonstrate our approach via a UML metamodel extension (see Section 4). Moreover, because our extension is a native UML extension it can directly be combined with other native UML elements, such as the swimlane notation, if needed or if desired.

In [20], Epstein and Sandhu introduced an early approach to model different RBAC aspects. However, their approach is described informally and provides only very basic support for the modeling of RBAC artifacts. In [76], Shin and Ahn suggest how UML models can be used to represent different aspects of RBAC. They do not introduce an extension but try to model different RBAC elements via standard UML models. However, the (expected) result is that UML does not provide special purpose modeling constructs for RBAC and thereby provides only limited support for defining RBAC models. Moreover, in [76] process-related RBAC models are not addressed. In [4], Ahn and Shin use OCL to define different types of role-based constraints. Nevertheless, these constraints are based on examples (e.g. for defining two conflicting manager roles). In particular, the constraints presented in [4] are not generic and do not define the semantics of RBAC-specific modeling constructs, but are defined for a certain (application-specific) class model. In [77,78], Sohr, Ahn et al. present an enhanced version of the approach including a tool to check OCL constraints. However, their main focus is on the definition and checking of application-specific RBAC constraints using OCL rather than on modeling process-related RBAC models (see Sections 3 and 4). Therefore, Table 2 indicates that this approach offers partial support for the different concepts. In [79], Sohr et al. describe how they applied their approach in a Web services context. A similar approach for specifying RBAC constraints via OCL is described in [96].

Ray et al. [65] use UML to visualize RBAC constraints. In particular, they use UML object diagrams to define so called template diagrams. These template diagrams visualize invalid configurations of a particular RBAC model. As an alternative, they propose to use OCL templates to define RBAC constraints.

Rodriguez et al. [67] present an BPMN extension for modeling security requirements in business processes. In particular, they aim to support the definition of non-repudiation, attack harm detection, integrity, privacy, and access control requirements. However, as their focus is on security requirements, they do not provide means for the definition of RBAC models, mutual exclusion constraints, or binding constraints. In [66], Rodriguez et al. present a UML profile that provides the same elements introduced in [67] only this time for UML2 activity diagrams.

In [100], Wolter and Schaad present an approach for the modeling of authorization constraints in BPMN. They provide a formal definition for task-based constraints that allows one to specify the minimum number of users that must be involved in the execution of a particular task set (i.e. in the execution of two or more tasks), and the maximum number of tasks out of this set that can executed by the same user. Their approach supports the definition of roles, as well as DME tasks and task-based subject binding. However, the approach does not provide a direct support for SME tasks as defined in Section 3. Nevertheless, support for SME tasks could be added with moderate efforts. Based on the formal definition of task constraints, Wolter and Schaad describe how to define dynamically mutual exclusive tasks and task-based subject binding in BPMN. However, their extension to BPMN is only defined informally.

Basin et al. [7] present a sophisticated approach called model-driven security. In particular, they first define domain-specific UML profiles for security modeling languages (e.g. an RBAC profile) and design modeling languages (e.g. a profile for component models). Subsequently, these UML profiles are integrated to provide a bridge between a security modeling language and a design modeling language. Based on the integrated ("security-enabled") design language, transformations are used to generate corresponding source code artifacts for a particular software platform. The generated artifacts then enforce the application-specific security policies defined on the modeling level. Basin et al. demonstrate the approach with an UML profile for RBAC which they call SecureUML. This profile also includes an element called AuthorizationConstraint which can be used to define constraints on permissions. However, the SecureUML profile does not include semantics for the definition of SME or DME tasks. The focus of [7] is on integrating security aspects with a model-driven development approach rather than modeling of business processes and process-related RBAC models. In fact, the model-driven security approach of [7] and our Business Activities are well-suited to be combined in a complementary fashion.

Another model-driven approach is presented by Alam et al. [5]. They propose an RBAC extension called constraint-based RBAC (CRBAC) which is supposed to be used in the service-oriented architecture (SOA) context. A UML profile is defined to provide modeling support for CRBAC. In addition, they use a special purpose (OCL-inspired) language to define permission-to-role assignments. The models and permission-to-role assignments are then used to generate corresponding XACML documents. The XACML policies are then used for access control purposes in a web services framework. In [12], Breu et al. present a closely related approach that uses elements of UML use case diagrams and OCL to define roles and permissions.

In [38] Koch and Parisi-Presicce present an approach for the UML-based specification and verification of access control policies. They use standard class models to define a so called type diagram for a particular access control model (e.g. for RBAC). Subsequently, UML object diagrams are used to specify certain rules and constraints for the different entities included in the respective type diagram. After the class and object diagrams are defined, graph transformations are applied to verify the resulting access control specification. The approach is demonstrated using an RBAC variant, called view-based access control, where permissions are associated with views and views are assigned to roles. Nevertheless, in [38] the example type diagram as well as the corresponding constraints are only specified informally by means of examples. The semantics of different access control model elements (e.g. roles, views, permissions) or relations between these elements are not defined in detail. Furthermore, the focus of [38] is on the verification of UML-based models via graph transformations, rather than on modeling support for process-related RBAC models and corresponding constraints. In [25], a similar approach is presented which gives an overview of how model-driven techniques can be applied to map the (platform-independent) view-based access control model to a specific software platform, and graph transformations are used to verify the consistency of the different models. The approaches presented in [25,38] can be combined with our approach in a complementary fashion.

Another related approach is UMLsec [32,33]. In essence, it provides a UML profile for the definition and analysis of security properties for software systems. For example, UMLsec is used to define and verify cryptographic protocols. In addition, it can be used for design and runtime checking of permission-based security [34,35]. However, UMLsec does not deal with process-related RBAC models and aims at a lower abstraction layer than Business Activities. Therefore, UMLsec is well-suited to be combined with our approach. Business Activities would then be used to model business processes and process-related RBAC models, while UMLsec would be used to specify the fine-grained system-level procedures for permission and constraint checking in a particular software system.

In addition to the approaches described above, other model-driven security approaches exist that do not address the modeling of process-related RBAC models but can be combined with our Business Activities. An example is the approach for model-driven business process security requirement specification presented in [99].

9. Conclusion

In this paper, we presented Business Activities as an integrated approach to model processes and process-related RBAC models. The approach was inspired by our experience gained in numerous role engineering projects and case studies (see Sections 1.1 and 6). We defined a formal generic metamodel for Business Activities and demonstrated our approach via an UML2 extension. Moreover, we implemented a library and runtime engine as a software platform for Business Activities that enables a seamless mapping from modeling-level Business Activities to corresponding runtime models. Thereby, the approach supports the continuous consistency of Business Activities, process-related RBAC models, and constraints that are defined at the design-level on the one hand, and corresponding runtime models which are managed by the software platform on the other. The new notation elements for Business Activities are lightweight and easy to use for modelers who are familiar with UML2 activity models. In our extension, RBAC-specific knowledge (including mutual exclusion and binding constraints) is encoded via OCL invariants. Therefore, each valid Business Activity must conform to the corresponding OCL constraints. The UML2 extension for Business Activities can be integrated with other UML-based approaches or tools. However, our approach is generic and does not rely on UML or any particular software platform. Thus, it can also be applied to extend other modeling languages or other software platforms.

Moreover, Business Activities can also help identify shortcomings in technologies or software platforms that are intended to implement the respective processes and access control policies. In other words: Business Activity models can serve as an integrated description of the processes and access control information that is needed in a certain business context or by a certain organization. In case we want to enforce the corresponding behaviour in an information system, we need to map the different modeling-level artifacts to the corresponding target system (i.e. to functions of the respective software components). This way, we can directly identify which access control policies and constraints can already be enforced in the corresponding software system and which parts of a Business Activity cannot be enforced yet (e.g. because the corresponding runtime system lacks corresponding implementation support).

From our experience in role engineering (see, e.g., [39,54, 84,86,88]) a proper motivation was found to define RBAC models and constraints even if they cannot (yet) be enforced on a technical level. The aim to specify and maintain a comprehensive, and preferably complete, process model that is directly integrated with a tailored RBAC model is probably the most important reason for specifying a complete RBAC model. Such "complete" models provide valuable information for the corresponding process and security engineers. For example, it is then possible to identify which subset of an organization's Business Activities can (already) be enforced by the runtime system and which control objectives cannot be achieved yet. This information can be applied to thoroughly configure the respective system and to avoid security breaches that could result from unavailable information. Furthermore, a "complete" description of processes and RBAC models at the level of system requirements and system design can drive the technical evolution of the corresponding target systems to close the gap between a process-related RBAC model and its enforceable subset.

In our future work, we will investigate how other types of constraints, such as context constraints [88], can be expressed at the modeling-level. Another interesting issue is to define suitable modeling primitives for the delegation of roles as well as the delegation of task instances. This is especially important because mutual exclusion constraints and binding constraints can directly influence the delegability of a certain role or task. Thus, we plan to investigate how modeling-level constructs can express different types of constraints as well as the delegability of roles and tasks in a consistent and comprehensive way. In our future work, we also plan to investigate the integration of our concepts with other workflow management systems. In particular, we aim to address the satisfiability of processes with respect to a process-related RBAC model (including mutual exclusion and binding constraints). In addition to existing work [17], we plan to explicitly consider the impact of specific routing elements that are used in a certain process model, such as exclusive (XOR) and parallel splits. Moreover, we plan to conduct further industrial case studies, in particular to analyze potential issues with regard to the complexity and intelligibility of process-related RBAC models.

Appendix A. Invariants for business actions

This appendix provides the complete set of OCL invariants for the Business Activity extension presented in Section 4. In particular, the OCL invariants are defined in accordance with the generic metamodel specified in Section 3. Table 1 shows how each of the generic definitions from Section 3 is mapped to our the Business Activity UML extension.

OCL Constraint 1 *Each role may have direct and indirect/transitive junior-roles. In other words, if a role r has junior-roles and these junior-roles have junior-roles themselves, r inherits the indirect junior-roles as transitive junior-roles.*

```
context Role inv:
  self.seniorAssignment->forAll(sa|
    sa.juniorRole.seniorAssignment->forAll(jrsa|
     self.transitiveJuniorRole->exists(tjr| tjr.name=jrsa.juniorRole.name)
     and
     jrsa.juniorRole.transitiveJuniorRole->forAll(jrtjr|
      self.transitiveJuniorRole->exists(tjr| tjr.name=jrtjr.name)
)  )   )
```

OCL Constraint 2 *A role-hierarchy must be cycle-free. In other words, a role cannot be a junior-role of itself, neither directly nor transitively.*

```
context Role inv:
  self.seniorAssignment->forAll(sa|
    not (self.name=sa.juniorRole.name))
  and
  self.transitiveJuniorRole->forAll(tjr|
    not (self.name=tjr.name))
```

OCL Constraint 3 *Each role inherits the BusinessActions assigned to its junior-roles (i.e. BusinessActions assigned to junior-roles are indirectly/ transitively assigned to the corresponding senior-roles)*

```
context Role
inv:
self.seniorAssignment->forAll(sa|
  sa.juniorRole.businessAction->forall(ba|
    self.inheritedTask->exists(it| it.name=ba.name)
  )
  and
  sa.juniorRole.inheritedTask->forAll(jrit|
    self.inheritedTask->exists(it| it.name=jrit.name)
) )

inv:
self.inheritedTask->forAll(it|
 self.seniorAssignment->exists(sa|
   sa.juniorRole.businessAction->exists(ba| ba.name=it.name)
   or
   sa.juniorRole.inheritedTask->exists(jrit| jrit.name=it.name)
) )
```

OCL Constraint 4 *Aside from the roles directly assigned to a subject, each subject owns the junior-roles of its directly assigned roles (i.e. junior-roles are indirectly/transitively assigned to a subject).*

```
context Subject
inv:
self.roleToSubjectAssignment->forAll(rsa|
 rsa.role.seniorAssignment->forAll(sa|
  self.inheritedRole->exists(ir| ir.name=sa.juniorRole.name))
 and
 rsa.role.transitiveJuniorRole->forAll(tjr|
  self.inheritedRole->exists(ir| ir.name=tjr.name))
)

inv:
self.inheritedRole->forAll(ir|
 self.roleToSubjectAssignment->exists(rsa|
  rsa.role.seniorAssignment->exists(sa| sa.juniorRole.name=ir.name)
  or
  rsa.role.transitiveJuniorRole->exists(tjr| tjr.name=ir.name)
))
```

OCL Constraint 5 *Being a specialized UML2 activity, a BusinessActivity may contain any standard UML2 activity node* [61]. *In addition to standard activity nodes, a BusinessActivity includes one or more BusinessActions – and BusinessActions are always included in a BusinessActivity.*

```
context Activity inv:
 if self.node->exists(n|
        n.oclIsKindOf(BusinessAction))
 then self.oclIsKindOf(BusinessActivity)
 endif
```

OCL Constraint 6 *In order to unambiguously identify different instances of the same BusinessActivity, we require that each BusinessActivity defines an attribute called "processID".*

```
context BusinessActivity inv:
  self.instanceSpecification->forAll(i|
    i.slot->exists(s|
      s.definingFeature.name=processID
) )
```

OCL Constraint 7 *Each BusinessAction defines an attribute "owningProcessInstance" (and thereby each BusinessAction instance owns a respective slot) to ensure that at runtime each instance of a BusinessAction can be associated with the corresponding BusinessActivity instance.*

```
context BusinessAction inv:
  self.instanceSpecification->forAll(i|
    i.slot->exists(s|
      s.definingFeature. name=owningProcessInstance
  ) )
```

(*continued on next page*)

OCL Constraint 8 *Each instance of a BusinessAction must be included in an instance of the corresponding BusinessActivity.*

```
context BusinessAction inv:
  self.instanceSpecification->forAll(i|
    self.activity.instanceSpecification->exists(a|
      i.slot->select(si|
        si.definingFeature. name=owningProcessInstance
          a.slot->select(sa|
            sa.definingFeature.name=processID
            and
            si.value=sa.value
))))
```

OCL Constraint 9 *To assign subjects to an instance of a BusinessAction, and to determine the subject that executes a particular BusinessAction instance included in a particular BusinessActivity instance, we require that each BusinessAction defines an attribute called "executingSubject" (and thereby each instance owns a respective slot). Moreover, the executingSubject attribute must refer to a subject that is (via one of its roles) actually allowed to execute this BusinessAction.*

```
context BusinessAction inv:
  self.instanceSpecification->forAll(i|
    i.slot->exists(s|
      s.definingFeature.name=executingSubject
    and
      (self.role->exists(r|
      r.roleToSubjectAssignment->exists(rsa| rsa.subject.name=s.value)
      or
      r.transitiveRoleOwner->exists(tro| tro.name=s.value))
  )))
```

OCL Constraint 10 *In order to define role-binding constraints, we must be able to identify the executing role of a particular BusinessAction. Thus, we require that each BusinessAction defines an attribute called "executingRole", and therefore each BusinessAction instance (i.e. each run-time instance of a task modeled via a BusinessAction) owns a corresponding slot. Moreover, the executingRole attribute must refer to a role that is associated with the corresponding BusinessAction (either directly or transitively via the role-hierarchy).*

```
context BusinessAction inv:
  self.instanceSpecification->forAll(i|
    i.slot->exists(s|
      s.definingFeature.name=executingRole
      and
        (self.role->exists(r| r.name=s.value)
        or
        self.transitiveTaskOwner->exists(tto| tto.name=s.value))
))
```

OCL Constraint 11 *A BusinessAction cannot be mutually exclusive to itself.*

```
context BusinessAction
inv: self.staticExclusion->select(sme|
      sme.name=self.name)->isEmpty()

inv: self.dynamicExclusion->select(dme|
      dme.name=self.name)->isEmpty()
```

OCL Constraint 12 *DME as well as SME constraints on BusinessActions are always mutual–to reflect the corresponding mutual exclusion relation.*

```
context BusinessActivity
inv:
self.node->select(oclIsKindOf(BusinessAction))->forAll (a1,a2|
    if a1.staticExclusion->exists(ax| ax.name=a2.name)
    then a2.staticExclusion->exists(ay| ay.name=a1.name)
  endif)
```

```
inv:
self.node->select(oclIsKindOf(BusinessAction))->forAll (a1,a2|
   if a1.dynamicExclusion->exists(ax| ax.name=a2.name)
   then a2.dynamicExclusion->exists(ay| ay.name=a1.name)
  endif)
```

OCL Constraint 13 *A BusinessAction cannot be bound to itself.*

```
context BusinessAction
inv: self.subjectBinding->select(sb|
        sb.name=self.name)->isEmpty()

inv: self.roleBinding->select(rb|
        rb.name=self.name)->isEmpty()
```

OCL Constraint 14 *Binding constraints on BusinessActions are always mutual – to reflect the corresponding binding relation.*

```
context BusinessActivity
inv:
self.node->select(oclIsKindOf(BusinessAction))->forAll (a1,a2|
   if a1.subjectBinding->exists(ax| ax.name=a2.name)
   then a2.subjectBinding->exists(ay| ay.name=a1.name)
endif)

inv:
self.node->select(oclIsKindOf(BusinessAction))->forAll (a1,a2|
   if a1.roleBinding->exists(ax| ax.name=a2.name)
   then a2.roleBinding->exists(ay| ay.name=a1.name)
endif)
```

OCL Constraint 15 *Either a DME or a SME constraint can be defined on two BusinessActions but not both.*

```
context BusinessAction inv:
self.staticExclusion->forAll(sme|
  self.dynamicExclusion->select(dme|
    sme.name=dme.name)->isEmpty()
)
```

OCL Constraint 16 *One may either specify a mutual exclusion constraint or a binding constraint between two BusinessActions, but not both. The only exception from this rule are DME constraints and role binding constraints which do not conflict (see also discussion from* Section 3*).*

```
context BusinessAction
inv: self.staticExclusion->forAll(sme|
       self.subjectBinding->select(sb|
             sb.name=sme.name)->isEmpty())

inv: self.staticExclusion->forAll(sme|
       self.roleBinding->select(rb|
             rb.name=sme.name)->isEmpty())

inv: self.dynamicExclusion->forAll(dme|
       self.subjectBinding->select(sb|
             sb.name=dme.name)->isEmpty())
```

OCL Constraint 17 *In a role-hierarchy, mutual exclusion constraints are subject to inheritance (see Section 3). Therefore, two SME Business-Actions must never be assigned to the same role, neither directly nor transitively.*

```
context Role
inv:
self.businessAction->forAll(ba1,ba2|
 ba1.staticExclusion->select(sme|
  sme.name=ba2.name)->isEmpty())
  )
```

(*continued on next page*)

```
inv:
self.businessAction->forAll(ba|
 self.inheritedTask->forAll(it|
   ba.staticExclusion->select(sme|
    sme.name=it.name)->isEmpty()
)))
```

OCL Constraint 18 A subject must never be assigned to two roles which own SME Business Actions, neither directly nor transitively.

```
context Subject inv:
self.roleToSubjectAssignment->forAll(rsa1,rsa2|
  rsa1.role.businessAction->forAll(ba1|
    rsa2.role.businessAction->forAll(ba2|
      rsa1.role.inheritedTask->forAll(it1|
        rsa2.role.inheritedTask->forAll(it2|
          ba1.staticExclusion->select(sme| sme.name=ba2.name)->isEmpty()
          and
          ba1.staticExclusion->select(sme| sme.name=it2.name)->isEmpty()
          and
          it1.staticExclusion->select(sme| sme.name=it2.name)->isEmpty()
          and
          ba2.staticExclusion->select(sme| sme.name=it1.name)->isEmpty()
)))))
```

OCL Constraint 19 *To enforce SME constraints on BusinessActions, we specify that the instances of two SME BusinessActions must never have the same executing subject.*

```
context BusinessAction inv:
  self.staticExclusion->forAll(sme|
   self.instanceSpecification->forAll(i|
     sme.instanceSpecification->forAll(j|
      i.slot->forAll(is|
       j.slot->forAll(js|
        if is.definingFeature.name=executingSubject
           and
           js.definingFeature.name=executingSubject
        then
            not (is.value=js.value)
         endif
)))))
```

OCL Constraint 20 *To enforce DME constraints on BusinessActions, we define that for each BusinessActivity the instances of two DME BusinessActions which are included in this BusinessActivity must be executed by two distinct subjects.*

```
context BusinessAction inv:
  self.dynamicExclusion->forAll(dme|
    self.instanceSpecification->forAll(i|
      dme.instanceSpecification->forAll(j|
        i.slot->select(si1|
          si1.definingFeature.name=owningProcessInstance
          j.slot->select(sj1|
            sj1.definingFeature.name=owningProcessInstance
            i.slot->select(si2|
              si2.definingFeature.name=executingSubject
              j.slot->select(sj2|
                sj2.definingFeature.name=executingSubject
                if (si1.value=sj1.value)
                then
                    not (si2.value=sj2.value)
                endif
))))))
```

OCL Constraint 21 *In case a role-binding is defined on two BusinessActions, instances of these actions must always be associated with the same executing role.*

```
context BusinessAction inv:
  self.roleBinding->forAll(rbt|
    self.instanceSpecification->forAll(i|
      rbt.instanceSpecification->forAll(j|
        i.slot->select(si1|
          si1.definingFeature.name=owningProcessInstance
          j.slot->select(sj1|
            sj1.definingFeature.name=owningProcessInstance
            i.slot->select(si2|
              si2.definingFeature.name=executingRole
              j.slot->select(sj2|
                sj2.definingFeature.name=executingRole
                if (si1.value=sj1.value)
                then (si2.value=sj2.value)
                endif
)))))))
```

OCL Constraint 22 *If a subject-binding is defined on two BusinessActions, instances of these actions must always be associated with the same executing subject.*

```
context BusinessAction inv:
  self.subjectBinding->forAll(sbt|
    self.instanceSpecification->forAll(i|
      sbt.instanceSpecification->forAll(j|
        i.slot->select(si1|
          si1.definingFeature.name=owningProcessInstance
          j.slot->select(sj1|
            sj1.definingFeature.name=owningProcessInstance
            i.slot->select(si2|
              si2.definingFeature.name=executingSubject
              j.slot->select(sj2|
                sj2.definingFeature.name=executingSubject
                if (si1.value=sj1.value)
                then (si2.value=sj2.value)
                endif
)))))))
```

References

[1] W.M.P. van der Aalst, Formalization and verification of event-driven process chains, Information and Software Technology 41 (10) (1999).

[2] W.M.P. van der Aalst, A.H.M. ter Hofstede, YAWL: yet another workflow language, Information Systems 30 (4) (2005).

[3] G.J. Ahn, R. Sandhu, Role-based authorization constraints specification, ACM Transactions on Information and System Security (TISSEC) 3 (4) (2000).

[4] G.J. Ahn, M.E. Shin, Role-based authorization constraints specification using object constraint language, in: Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), June 2001.

[5] M. Alam, M. Hafner, R. Breu, Constraint based role based access control in the SECTET-framework – a model-driven approach, Journal of Computer Security 16 (2) (2008).

[6] B. Axenath, E. Kindler, V. Rubin, AMFIBIA: a meta-model for the integration of business process modelling aspects, in: F. Leymann, W. Reisig, S.R. Thatte, W. van der Aalst (Eds.), The Role of Business Processes in Service Oriented Architectures, No. 06291 in Dagstuhl Seminar Proceedings, 2006.

[7] D. Basin, J. Doser, T. Lodderstedt, Model driven security: from UML models to access control infrastructures, ACM Transactions on Software Engineering and Methodology (TOSEM) 15 (1) (2006).

[8] J. Becker, M. Rosemann, C. von Uthmann, Guidelines of business process modeling, in: Business Process Management, Models, Techniques, and Empirical Studies, Lecture Notes in Computer Science (LNCS), vol. 1806, Springer Verlag, 2000.

[9] E. Bertino, J. Crampton, F. Paci, Access control and authorization constraints for WS-BPEL, in: Proceedings of the IEEE International Conference on Web Services (ICWS), September 2006.

[10] E. Bertino, E. Ferrari, V. Atluri, The specification and enforcement of authorization constraints in workflow management systems, ACM Transactions on Information and System Security (TISSEC) 2 (1) (1999).

[11] R.A. Botha, J.H.P. Eloff, Separation of duties for access control enforcement in workflow environments, IBM Systems Journal 40 (3) (2001).

[12] R. Breu, G. Popp, M. Alam, Model based development of access policies, International Journal on Software Tools for Technology Transfer 9 (5–6) (2007).

[13] J.C. Cannon, M. Byers, Compliance deconstructed, ACM Queue 4 (7) (2006).

[14] D.D. Clark, D.R. Wilson, A comparison of commercial and military computer security policies, in: Proceedings of the IEEE Symposium on Security and Privacy, April 1987.

[15] J. Cortadella, M. Kishinevsky, L. Lavagno, A. Yakovlev, Deriving petri nets from finite transition systems, IEEE Transactions on Computers 47 (1998).

[16] E.J. Coyne, J.M. Davis, Role Engineering for Enterprise Security Management, Artech House, 2008.

[17] J. Crampton, H. Khambhammettu, Delegation and satisfiability in workflow systems, in: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT), June 2008.

[18] M. Damianides, How does SOX change IT?, Journal of Corporate Accounting & Finance 15 (6) (2004)

[19] R.M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Information & Software Technology 50 (12) (2008).

[20] P. Epstein, R. Sandhu, Towards a UML based approach to role engineering, in: Proceedings of the 4th ACM Workshop on Role-Based Access Control, October 1999.

[21] R. Eshuis, R. Wieringa, Tool support for verifying UML activity diagrams, IEEE Transactions on Software Engineering (TSE) 30 (7) (2004).

[22] D.F. Ferraiolo, J.F. Barkley, D.R. Kuhn, A role-based access control model and reference implementation within a corporate Intranet, ACM Transactions on Information and System Security (TISSEC) 2 (1) (1999).
[23] D.F. Ferraiolo, D.R. Kuhn, Role-based access controls, in: Proceedings of the 15th National Computer Security Conference, October 1992.
[24] D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli, Role-Based Access Control, second ed., Artech House, 2007.
[25] T. Fink, M. Koch, K. Pauls, An MDA approach to access control specifications using MOF and UML profiles, in: Proceedings of the First International Workshop on Views on Designing Complex Architectures (VODCA), Electronic Notes in Theoretical Computer Science, vol. 142, January 2006, pp. 161–179.
[26] M.P. Gallaher, A.C. O'Connor, B. Kropp, The Economic Impact of Role-Based Access Control, National Institute of Standards & Technology (NIST), Planning Report 02-1, March 2002.
[27] C.K. Georgiadis, I. Mavridis, G. Pangalos, R.K. Thomas, Flexible team-based access control using contexts, in: Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT), May 2001.
[28] K. Irwin, T. Yu, W.H. Winsborough, Enforcing security properties in task-based systems, in: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT), June 2008.
[29] ISO, Information technology – Security techniques – Code of practice for information security management, ISO/IEC 27002:2005, Stage: 90.92, April 2008, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50297>.
[30] ISO, Information technology – Security techniques – Information security management systems – Requirements, ISO/IEC 27001:2005, Stage: 90.92, October 2008, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=42103>.
[31] ISO, Information technology – Security techniques – Information security management systems – Overview and vocabulary, ISO/IEC 27000:2009, Stage: 60.60, April 2009, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=41933>.
[32] J. Jürjens, Secure Systems Development with UML, Springer Verlag, 2005.
[33] J. Jürjens, Sound methods and effective tools for model-based security engineering with UML, in: Proceedings of the 27th International Conference on Software Engineering (ICSE), May 2005.
[34] J. Jürjens, Model-based run-time checking of security permissions using guarded objects, in: Proceedings of the 8th International Workshop on Runtime Verification, Lecture Notes in Computer Science (LNCS), vol. 5289, Springer Verlag, 2008.
[35] J. Jürjens, M. Lehrhuber, G. Wimmel, Model-based design and analysis of permission-based security, in: Proceedings of the 10th International Conference on Engineering of Complex Computer Systems (ICECCS), June 2005.
[36] S.K. Kim, D. Burger, D.A. Carrington, An MDA approach towards integrating formal and informal modeling languages, in: International Symposium of Formal Methods Europe, Lecture Notes in Computer Science (LNCS), vol. 3582, Springer Verlag, 2005, pp. 448–464.
[37] E. Kindler, On the semantics of EPCs: resolving the vicious circle, Data & Knowledge Engineering (DKE) 56 (1) (2006).
[38] M. Koch, F. Parisi-Presicce, UML specification of access control policies and their formal verification, Software and System Modeling 5 (4) (2006) 429–447.
[39] S. Kunz, S. Evdokimov, B. Fabian, B. Stieger, M. Strembeck, Role-based access control for information federations in the industrial service sector, in: Proceedings of the 18th European Conference on Information Systems (ECIS), June 2010.
[40] C.E. Landwehr, Formal models for computer security, ACM Computing Surveys 13 (3) (1981).
[41] F. Leymann, D. Roller, Production Workflow – Concepts and Techniques, Prentice Hall, 2000.
[42] N. Li, M.V. Tripunitara, Z. Bizri, On mutually exclusive roles and separation-of-duty, ACM Transactions on Information and System Security (TISSEC) 10 (2) (2007).
[43] N. Li, Q. Wang, Beyond separation of duty: an algebra for specifying high-level security policies, Journal of the ACM (JACM) 55 (3) (2008).
[44] B. List, B. Korherr, An evaluation of conceptual business process modelling languages, in: Proceedings of the 21st ACM Symposium on Applied Computing (SAC), April 2006.
[45] N. Lohmann, E. Verbeek, R.M. Dijkman, Petri net transformations for business processes – a survey, Transactions on Petri Nets and Other Models of Concurrency 2 (2009).
[46] J. Mendling, Metrics for Process Models: Empirical Foundations of Verification, Error Prediction and Guidelines for Correctness, Lecture Notes in Business Information Processing (LNBIP), vol. 6, Springer Verlag, 2008.
[47] J. Mendling, W.M.P. van der Aalst, Formalization and verification of EPCs with OR-joins based on state and context, in: J. Krogstie, A.L. Opdahl, G. Sindre (Eds.), Proceedings of the 19th Conference on Advanced Information Systems Engineering (CAiSE), Lecture Notes in Computer Science, vol. 4495, Springer Verlag, Trondheim, Norway, 2007.
[48] J. Mendling, K. Ploesser, M. Strembeck, Specifying separation of duty constraints in BPEL4People processes, in: Proceedings of the 11th International Conference on Business Information Systems (BIS), Lecture Notes in Business Information Processing (LNBIP), vol. 7, Springer-Verlag, 2008.
[49] J. Mendling, M. Strembeck, G. Stermsek, G. Neumann, An approach to extract RBAC models from BPEL4WS processes, in: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), June 2004.
[50] T. Mens, P. Van Gorp, A taxonomy of model transformation, Electronic Notes in Theoretical Computer Science 152 (2006) 125–142.
[51] M. Mernik, J. Heering, A.M. Sloane, When and how to develop domain-specific languages, ACM Computing Surveys 37 (4) (2005) 316–344.
[52] S. Mishra, H.R. Weistroffer, A framework for integrating Sarbanes-Oxley compliance into the systems development process, Communications of the Association for Information Systems (CAIS) 20 (1) (2007).
[53] T. Murata, Petri nets: properties, analysis and applications, Proceedings of the IEEE 77 (4) (1989).
[54] G. Neumann, M. Strembeck, A scenario-driven role engineering process for functional RBAC roles, in: Proceedings of 7th ACM Symposium on Access Control Models and Technologies (SACMAT), June 2002.
[55] NIST, An Introduction to Computer Security: The NIST Handbook, National Institute of Standards& Technology (NIST), Special Publication 800-12, October 1995, <http://csrc.nist.gov/publications/nistpubs/800-12/handbook.pdf>.
[56] NIST, Recommended Security Controls for Federal Information Systems and Organizations, National Institute of Standards & Technology (NIST), Special Publication 800-53, Revision 3, August 2009, <http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final-errata.pdf>.
[57] S. Oh, S. Park, Task-role-based access control model, Information Systems 28 (6) (2003).
[58] OMG, Object Constraint Language Specification, Version 2.0, formal/06-05-01, The Object Management Group, May 2006, <http://www.omg.org/technology/documents/formal/ocl.htm>.
[59] OMG, Business Process Modeling Notation (BPMN), Version 1.2, formal/2009-01-03, The Object Management Group, January 2009, <http://www.omg.org/spec/BPMN/1.2/>.
[60] OMG, Meta Object Facility (MOF): Core Specification, Version 2.0, formal/06-01-01, The Object Management Group, January 2006, <http://www.omg.org/spec/MOF/2.0/>.
[61] OMG, Unified Modeling Language (OMG UML): Superstructure, Version 2.2, formal/2009-02-02, The Object Management Group, February 2009, <http://www.omg.org/technology/documents/formal/uml.htm>.
[62] C. Ouyang, M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, J. Mendling, From business process models to process-oriented software systems, ACM Transactions on Software Engineering and Methodology (TOSEM) 19 (1) (2009).
[63] J.L. Peterson, Petri nets, ACM Computing Surveys (CSUR) 9 (3) (1977) 223–252.
[64] C.A. Petri, Fundamentals of a theory of asynchronous information flow, in: Proceedings of the Information Processing Congress (IFIP Congress), August/September 1962.
[65] I. Ray, N. Li, R. France, D.K. Kim, Using UML to visualize role-based access control constraints, in: Proceedings of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT), June 2004.
[66] A. Rodríguez, E. Fernández-Medina, M. Piattini, Capturing security requirements in business processes through a UML 2.0 activity diagrams profile, in: Workshop Proceedings of Advances in Conceptual Modeling – Theory and Practice (ER Workshops), Lecture Notes in Computer Science (LNCS), vol. 4231, Springer Verlag, 2006, pp. 32–42.
[67] Alfonso Rodríguez, Eduardo Fernández-Medina, Mario Piattini, A BPMN extension for the modeling of security requirements in business processes, IEICE Transactions on Information and Systems 90-D (4) (2007) 745–752.
[68] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, D. Edmond, Workflow resource patterns: identification, representation and tool support, in: O. Pastor, J. Falcão e Cunha (Eds.), Proceedings of the 17th International Conference on Advanced Information Systems Engineering, CAiSE 2005, Porto, Portugal, June 13–17, 2005, Lecture Notes in Computer Science, vol. 3520, Springer, 2005, pp. 216–232.
[69] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, P. Wohed, On the suitability of UML 2.0 activity diagrams for business process modelling, in: Proceedings of the Third Asia-Pacific Conference on Conceptual Modelling (APCCM), January 2006.
[70] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role-based access control models, IEEE Computer 29 (2) (1996).
[71] R.S. Sandhu, P. Samarati, Access control: principles and practice, IEEE Communications 32 (9) (1994).
[72] A.W. Scheer, ARIS – Business Process Modeling, 3rd ed., Springer Verlag, 2000.
[73] Douglas C. Schmidt, Model-driven engineering – guest editor's introduction, Computer 39 (2) (2006).
[74] B. Selic, The pragmatics of model-driven development, IEEE Software 20 (5) (2003).
[75] S. Sendall, W. Kozaczynski, Model transformation: the heart and soul of model-driven software development, IEEE Software 20 (5) (2003).
[76] M.E. Shin, G.J. Ahn, UML-based representation of role-based access control, in: Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), June 2000.
[77] K. Sohr, G.J. Ahn, M. Gogolla, L. Migge, Specification and validation of authorisation constraints using UML and OCL, in: Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS), Lecture Notes in Computer Science (LNCS), vol. 3679, Springer Verlag, 2005.

[78] K. Sohr, M. Drouineaud, G.J. Ahn, M. Gogolla, Analyzing and managing role-based access control policies, IEEE Transactions on Knowledge and Data Engineering 20 (7) (2008) 924–939.

[79] K. Sohr, T. Mustafa, X. Bao, G.J. Ahn, Enforcing role-based access control policies in web services with UML and OCL, in: Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC), December 2008.

[80] D. Spinellis, Notable design patterns for domain-specific languages, Journal of Systems and Software 56 (1) (2001) 91–99.

[81] Thomas Stahl, Markus Völter, Model-Driven Software Development, John Wiley & Sons, 2006.

[82] E.A. Stohr, J.L. Zhao, Workflow automation: overview and research issues, Information Systems Frontiers 3 (3) (2001).

[83] M. Strembeck, Conflict checking of separation of duty constraints in RBAC – implementation experiences, in: Proceedings of the Conference on Software Engineering (SE 2004), February 2004.

[84] M. Strembeck, A role engineering tool for role-based access control, in: Proceedings of the 3rd Symposium on Requirements Engineering for Information Security (SREIS), August 2005.

[85] M. Strembeck, Embedding policy rules for software-based systems in a requirements context, in: Proceedings of the IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY), June 2005.

[86] M. Strembeck, Scenario-driven role engineering, IEEE Security & Privacy 8 (1) (2010).

[87] M. Strembeck, J. Mendling, 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 (LNCS), vol. 6426, Springer Verlag, 2010.

[88] M. Strembeck, G. Neumann, An integrated approach to engineer and enforce context constraints in RBAC environments, ACM Transactions on Information and System Security (TISSEC) 7 (3) (2004).

[89] M. Strembeck, U. Zdun, An approach for the systematic development of domain-specific languages, Software: Practice and Experience (SP&E) 39 (15) (2009).

[90] K. Tan, J. Crampton, C.A. Gunter, The consistency of task-based authorization constraints in workflow systems, in: Proceedings of the 17th IEEE Workshop on Computer Security Foundations (CSFW), June 2004.

[91] R.K. Thomas, Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments, in: Proceedings of the ACM Workshop on Role Based Access Control, 1997.

[92] R.K. Thomas, R.S. Sandhu, Task-based authorization controls (TBAC): a family of models for active and enterprise-oriented authorization management, in: Proceedings of the IFIP WG11.3 Conference on Database Security, August 1997.

[93] A. Valmari, The state explosion problem, in: Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996, Lecture Notes in Computer Science (LNCS), vol. 1491, Springer Verlag, 1998.

[94] K. van Hee, O. Oanea, A. Serebrenik, N. Sidorova, M. Voorhoeve, History-based joins: semantics, soundness and implementation, Data & Knowledge Engineering (DKE) 64 (1) (2008).

[95] J. Wainer, P. Barthelmes, A. Kumar, W-RBAC – a workflow security model incorporating controlled overriding of constraints, International Journal of Cooperative Information Systems (IJCIS) 12 (4) (2003).

[96] H. Wang, Y. Zhang, J. Cao, J. Yang, Specifying role-based access constraints with object constraint language, in: Proceedings of the 6th Asia-Pacific Conference Advanced Web Technologies and Applications, Lecture Notes in Computer Science (LNCS), vol. 3007, Springer Verlag, 2004.

[97] J. Warner, V. Atluri, Inter-instance authorization constraints for secure workflow management, in: Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT), June 2006.

[98] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, N. Russell, On the suitability of BPMN for business process modelling, in: Proceedings of the 4th International Conference on Business Process Management (BPM), Lecture Notes in Computer Science (LNCS), vol. 4102, Springer Verlag, 2006.

[99] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, C. Meinel, Model-driven business process security requirement specification, Journal of Systems Architecture 55 (4) (2009).

[100] C. Wolter, A. Schaad, Modeling of task-based authorization constraints in BPMN, in: G. Alonso, P. Dadam, M. Rosemann (Eds.), 5th International Conference on Business Process Management (BPM), Lecture Notes in Computer Science, vol. 4714, Springer, 2007, pp. 64–79.

[101] U. Zdun, Patterns of component and language integration, in: D. Manolescu, M. Voelter, J. Noble (Eds.), Pattern Languages of Program Design, vol. 5, Addison Wesley, 2006.

[102] U. Zdun, M. Strembeck, Modeling composition in dynamic programming environments with model transformations, in: Proceedings of the 5th International Symposium on Software Composition, Lecture Notes in Computer Science (LNCS), vol. 4089, Springer-Verlag, 2006.

[103] U. Zdun, M. Strembeck, Reusable architectural decisions for DSL design: foundational decisions in DSL projects, in: Proceedings of the 14th European Conference on Pattern Languages of Programs (EuroPLoP), July 2009.