Modeling Support for Delegating Roles, Tasks, and Duties in a Process-Related RBAC Context

Sigrid Schefer and Mark Strembeck

Institute for Information Systems and New Media
Vienna University of Economics and Business (WU Vienna), Austria
{firstname.lastname}@wu.ac.at

Abstract. The definition of access control concepts at the modeling level is an important prerequisite for the thorough implementation and enforcement of corresponding policies and constraints in a software system. In this paper, we present an approach to provide modeling support for the delegation of roles, tasks, and duties in the context of process-related RBAC models. The delegation model elements are integrated into a software engineering and business process context by providing UML2 modeling support for role-, task, and duty-level delegation. The semantics of our UML extension are formally specified via OCL constraints.

Key words: Access Control, Delegation, RBAC, UML

1 Introduction

In recent years, role-based access control (RBAC) [8, 12] has developed into a de facto standard for access control. In the area of workflow modeling, roles are also used as an abstract concept for delegation [6, 17] or for the assignment of duties defined via obligations [13, 15, 21]. Delegation provides a mechanism to increase flexibility in security management. In essence, one subject can delegate its permissions and duties to another subject [13]. Subsequently, the subject receiving the delegation will act on behalf of the delegator. In order to model the delegation of roles, tasks, and duties in a process-related context, we need an approach that integrates the different concepts in a modeling language. However, standard process modeling languages, such as BPMN or UML Activity diagrams [11], do not provide native language constructs to model RBAC elements. Due to missing modeling support for process-related delegation of roles, tasks, and duties, organizations often try to specify delegation processes via informal textual comments. However, such work-arounds easily result in significant problems regarding consistency between process descriptions and actual process executions [18], and they make it difficult to translate the respective modeling-level concepts to actual software systems.

In this paper, we present an approach for the integrated modeling of business processes and the delegation of roles, tasks, and duties. For this purpose, we present a UML metamodel extension to model the delegation of roles, tasks, and duties via extended UML2 Activity diagrams. Moreover, we formally define the semantics of our newly introduced UML elements via OCL constraints. The remainder of this paper is structured as follows. Section 2 introduces our extension for UML Activity diagrams which allows for the process-related modeling of the proposed delegation model elements. Subsequently, Section 3 presents an example business process model including delegation. Section 4 discusses our approach in comparison to related work and Section 5 concludes the paper.

2 UML Extension for Modeling Delegation

An organization's business processes and software systems are often modeled via graphical modeling languages. The Unified Modeling Language (UML) [11] offers a comprehensive and well-defined modeling framework and is the de facto standard for modeling and specifying information systems. Providing modeling support for delegation of roles, tasks, and duties in business processes via a standard notation like UML can bridge the communication gap between software engineers, security experts, and non-technical stakeholders (see, e.g., [9]).

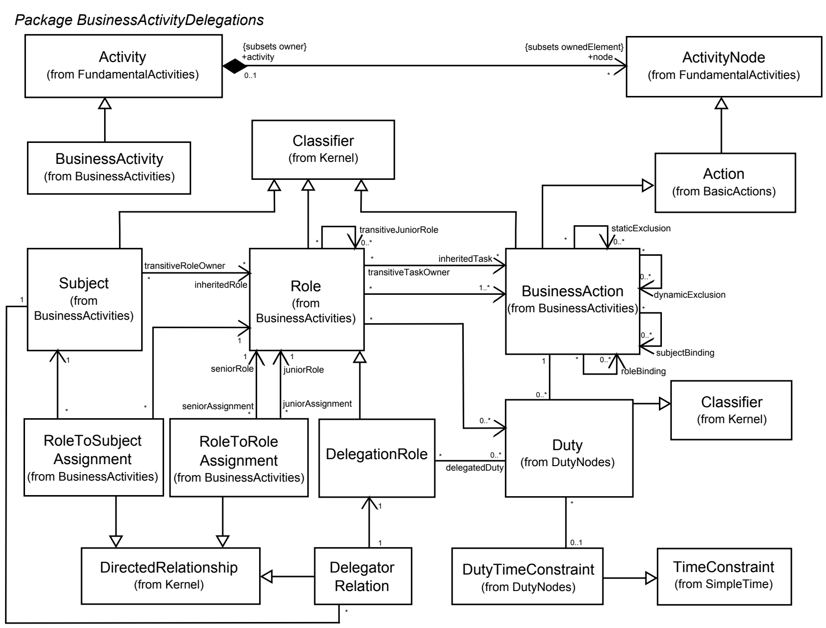

Fig. 1: UML metamodel extension *BusinessActivityDelegations* for Activity diagrams

To achieve the above, we model the delegation of roles, tasks, and duties via *extended UML2 Activity diagrams*. UML2 Activity models provide a process modeling language that allows to model the control and object flows between different actions (for details on UML2 Activity models, see [11]). We introduce a

UML metamodel extension *BusinessActivityDelegations* for modeling delegation of roles, tasks, and duties (see Figure 1). Moreover, we use OCL invariants [10] to define the semantics by encoding delegation-specific constraints.

A *BusinessActivity* is a special UML Activity (see Figure 1). It can include all elements available for ordinary UML Activities in addition to our newly introduced elements. A *BusinessAction* corresponds to a task and comprises all necessary permissions to perform the task (see [16] for further details on BusinessActivities and BusinessActions). A *Duty* is a special UML Classifier (see Figure 1) and is used to model that an action must be performed by a certain Subject [14]. The link between Duties and BusinessActions assures that a Subject being assigned to a Duty also receives all necessary permissions to perform these Duties. *Roles* and *Subjects* are specialised UML Classifiers [16] which are linked to BusinessActions and Duties (see Figure 1). Furthermore, a Duty may be linked to a *DutyTimeConstraint* which is a specialised UML TimeConstraint (from the SimpleTime package, see [11]). If a DutyTimeConstraint has expired, a *Compensation Action* is triggered which is defined as stereotype of the Action metaclass (see [14] for further details).

A *DelegationRole* is a special type of Role which is assigned to a set of delegated Roles, BusinessActions, and/or Duties (see Figure 1). A *DelegatorRelation* is a special UML DirectedRelationship (from the Kernel package, see [11]) and indicates that a certain Subject acts as a delegator for a special DelegationRole. Only delegators may delegate Roles, BusinessActions, or Duties to DelegationRoles (see OCL constraint 1). DelegationRoles are assigned to delegatees which thereby are authorized to perform the respective BusinessActions and Duties (see OCL constraint 2). A delegator can *delegate a Role* by defining this Role as junior role of one of his or her DelegationRoles. All BusinessActions and Duties assigned to this Role need to be delegatable (see below). Note, DelegationRoles must not have senior regular Roles to avoid invalid permission inheritance (see OCL constraint 3). For *delegating a BusinessAction*, the delegator assigns the BusinessAction to the respective DelegationRole. Only if a BusinessAction is delegatable, it can be delegated to a DelegationRole (see OCL constraints 4 and 6). To realize *delegation of Duties* in UML models, a Duty also needs to be defined as being delegatable (see OCL constraints 5, 7, and 9). After assigning a delegatee, the delegator loses his obligation to perform this Duty. Yet, a review duty can be defined [13] which obliges the delegator to control the proper enforcement of his delegated Duties (see OCL constraints 8 and 9).

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

```
context Subject
inv: self.delegatorRelation.delegationRole.delegatedDuty.role->exists(r |
       r.roleToSubjectAssignment->exists(rsa |
          rsa.subject.name = self.name ))
inv: self.delegatorRelation.delegationRole.businessaction.role->exists(r |
       r.roleToSubjectAssignment->exists(rsa |
          rsa.subject.name = self.name ))
inv: self.delegatorRelation.delegationRole.juniorAssignment.role->exists((r |
       r.roleToSubjectAssignment->exists(rsa |
          rsa.subject.name = self.name ))
```

OCL Constraint 2 *Each DelegationRole defines an attribute called "delegatee". The delegatee is the responsibleSubject for the delegated BusinessActions and Duties:*

```
context DelegationRole inv:
self.instanceSpecification->forAll(i |
  self.businessAction.instanceSpecification->forall(b |
    self.delegatedDuty.instanceSpecification->forall(d |
      i.slot->select(s | s.definingFeature.name = delegatee
        b.slot->select(rsb | rsb.definingFeature.name = responsibleSubject
          d.slot->select(rsd | rsd.definingFeature.name = responsibleSubject
            s.value = rsb.value and
            s.value = rsd.value ))))))
```

OCL Constraint 3 *A DelegationRole is only allowed to have senior-assignments to other DelegationRoles (see [20]):*

```
context DelegationRole
inv: self.seniorAssignment->forAll(sa | sa.seniorrole.oclIsKindOf(DelegationRole))
```

OCL Constraint 4 *Each BusinessAction defines an attribute called "delegatable" stating if a special BusinessAction may be delegated or not:*

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

OCL Constraint 5 *Each Duty defines an attribute called "delegatable":*

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

OCL Constraint 6 *Each BusinessAction defines an attribute called "isDelegated" stating if a special BusinessAction has already been delegated or not. If it has already been delegated, it cannot be delegated further (single-step delegation, see [3]):*

```
context BusinessAction inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(s | s.definingFeature.name = isDelegated and
      if s.value = true then
         i.slot->exists(d | d.definingFeature.name = delegateable and
            d.value = false)
      else true endif ))
```

OCL Constraint 7 *Duties define an attribute "isDelegated" (single-step delegation):*

```
context Duty inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(s | s.definingFeature.name = isDelegated and
      if s.value = true then
         i.slot->exists(d | d.definingFeature.name = delegateable and
            d.value = false )
      else true endif ))
```

OCL Constraint 8 *Each Duty defines an attribute called "isReviewDuty" [13]:*

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

OCL Constraint 9 *If a Duty is delegatable, it cannot be a reviewDuty. If a Duty is a reviewDuty, it is not delegatable. Furthermore, if a Duty is a reviewDuty, it must not have been delegated [13]:*

```
context Duty inv:
   self.instanceSpecification->forAll(i |
      i.slot->select(d | d.definingFeature.name = delegatable
         i.slot->select(r | r.definingFeature.name = isReviewDuty
            i.slot->selelct(si |
               si.definingFeature.name = isDelegated and
               d.delegatable.value <> r.isReviewDuty.value and
               r.isReviewDuty.value <> si.isDelegated.value ))))
```

To consider the aspect of permanence in delegation [3], our DelegationRoles can either be defined for *temporary* or for *permanent* delegation, i.e. for one or for all instances of a business process (see OCL constraints 10 and 11). Furthermore, we support *single-* and *multi-step delegation* for BusinessActions and Duties. Single-step delegation means that a delegated BusinessAction or Duty can not be delegated further by the delegatee [3]. This is achieved by defining an attribute called *isDelegated* for each BusinessAction and for each Duty. The isDelegated attribute is set to true as soon as the respective BusinessAction or Duty has been delegated. If a BusinessAction's or a Duty's isDelegated-attribute is set to true, its *delegatable*-attribute is set to false (see OCL constraints 6 and 7). Multi-step delegation can easily be activated by using OCL constraints 12 and 13.

OCL Constraint 10 *Each DelegationRole defines an attribute called "isTemporary" indicating if a DelegationRole is intended for temporary or for permanent delegation:*

```
context DelegationRole inv:
self.instanceSpecification->forAll(i |
   i.slot->select(si | si.definingFeature.name = isTemporary ))
```

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

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

OCL Constraint 12 *To allow for multi-step delegation of BusinessActions, use the following OCL constraint instead of OCL constraint 6:*

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

OCL Constraint 13 *To allow for multi-step delegation of Duties, use the following OCL constraint instead of OCL constraint 7:*

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

3 Example Process with Delegation

In Figure 2, a standard UML2 credit application process is extended by including the new modeling constructs introduced in Section 2. However, note, that the visualization presented here primarily serves as a presentation option to graphically illustrate the relations between the modeling elements. As each UML model needs to conform to its OCL constraints (see Section 2), the formally defined relations exist independent of their actual graphical representation (see [10, 11]).

The process in Figure 2a) includes five actions, three of which are defined as BusinessActions. The BusinessActions are associated with Duties. In addition, the Compensation Action *Reassign Duty* is triggered if the Duty *Check applicant rating* is not discharged in time. Figure 2b) presents the Duty *Check applicant rating* which is connected to the BusinessAction *Check credit worthiness*. It is associated with a DutyTimeConstraint and a Compensation Action. The DutyTimeConstraint expresses that the Duty *Check applicant rating* needs to be

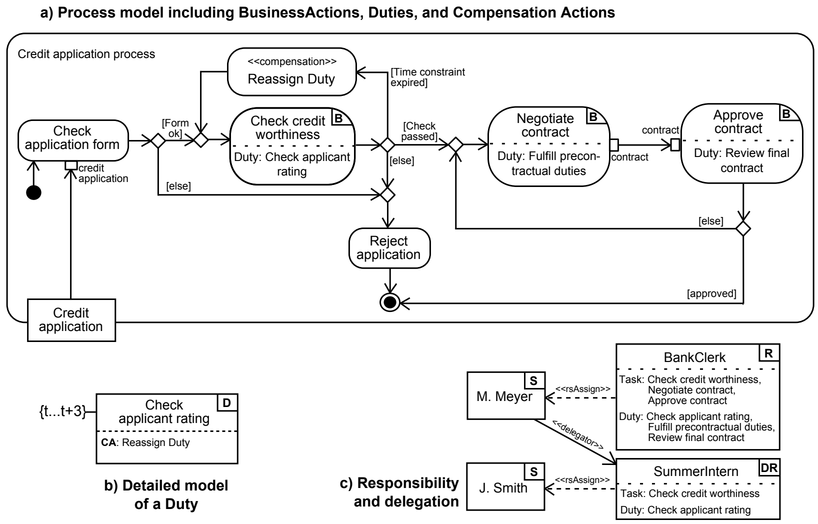

Fig. 2: Extended credit application process

completed within three time units (e.g. days) after the corresponding BusinessAction has been started. Otherwise, the Compensation Action *Reassign Duty* is executed.

The responsibility for the Duties is illustrated in Figure 2c) showing the Role BankClerk which is assigned to the three BusinessActions and the associated Duties defined in the credit application process. Thus, a Subject assigned to the BankClerk role is responsible for performing these Duties and related BusinessActions. In this example, the Subject *M. Meyer* is assigned to the BankClerk role and therefore also needs to discharge the associated Duties. M. Meyer decides to delegate her Duty *Check applicant rating* to her summer intern *J. Smith*. For this purpose, she creates a permanent DelegationRole *SummerIntern* and assigns the Duty to the DelegationRole. Subsequently, she assigns the Subject *J. Smith* to her DelegationRole *SummerIntern*. J. Smith is now authorized and responsible for discharging the Duty *Check applicant rating* when performing the BusinessAction *Check credit worthiness*, until either the Duty is revoked from the DelegationRole or he loses his assignment to the DelegationRole.

4 Related Work

To the best of our knowledge, this work represents the first attempt to address delegation of duties from a business process modeling context. Other approaches usually concentrate on the modeling of authorization constraints. As each duty holder also needs sufficient authority to perform the assigned duties [13, 15], our approach complements existing approaches. In recent years, there has been much work on various aspects of delegation (see, e.g., [2, 19, 20]), especially in a business process context. In [1], the notion of delegation is extended to allow for conditional delegation. Different types of constraints, such as authorization constraints, are addressed in the context of delegation. The effects of some delegation operations on three workflow execution models are described in [7]. In [5], the satisfiability problem of workflows while supporting user delegation mechanisms is addressed. Moreover, duties/obligations may also be subject to delegation. However, the delegation of duties has received little attention in literature so far, although it has been identified as important phenomenon, e.g., in [4], where different ways of delegating obligations are discussed. In [13], some issues for delegation of obligations are considered, mainly addressing the reasons for delegating obligations and the balance between authorizations and obligations.

5 Conclusion

Our UML2 extension can help organizations to integrate the specification of processes and related access control, obligation, and delegation policies. An integrated modeling approach yields a number of advantages, such as supporting a proper mapping of models to software systems, facilitating communication between different stakeholders, making responsibility for tasks and duties explicit, and detecting task- or duty-related conflicts. Moreover, it allows for tracing policy rules to the (regulatory) reasons they exist and to trace them to the software components that have to ensure their monitoring and enforcement. This facilitates the reporting on a company's fulfillment of compliance requirements. We chose to define an extension to the UML2 standard to enable a complete and correct mapping between policies, models, and the respective software system. This mapping assures consistency between modeling-level specifications and the software system enforcing respective policies and process instances.

References

1. V. Atluri and J. Warner. Supporting conditional delegation in secure workflow management systems. In *Proceedings of the tenth ACM symposium on Access control models and technologies (SACMAT)*, 2005.
2. E. Barka and R. Sandhu. A Role-Based Delegation Model and Some Extensions. In *Proceedings of the 23rd National Information Systems Security Conference (NIS-SEC)*, 2000.
3. E. Barka and R. Sandhu. Framework for Role-Based Delegation Models. In *Proceedings of the 16th Annual Computer Security Applications Conference*, 2000.
4. J. Cole, J. Derrick, Z. Milosevic, and K. Raymond. Author Obliged to Submit Paper before 4 July: Policies in an Enterprise Specification. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2001.
5. J. Crampton and H. Khambhammettu. Delegation and Satisfiability in Workflow Systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies (SACMAT)*, 2008.

6. J. Crampton and H. Khambhammettu. Delegation in role-based access control. *International Journal of Information Security*, 7(2), 2008.
7. J. Crampton and H. Khambhammettu. On Delegation and Workflow Execution Models. In *Proceedings of the 2008 ACM symposium on Applied computing (SAC)*, 2008.
8. D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, second edition, 2007.
9. H. Mouratidis and J. Jürjens. From Goal-Driven Security Requirements Engineering to Secure Design. *International Journal of Intelligent Systems*, 25(8), 2010.
10. OMG. Object Constraint Language Specification. available at: http://www.omg.org/technology/documents/formal/ocl.htm, February 2010. Version 2.2, formal/2010-02-01, The Object Management Group.
11. OMG. Unified Modeling Language (OMG UML): Superstructure. available at: http://www.omg.org/technology/documents/formal/uml.htm, May 2010. Version 2.3, formal/2010-05-05, The Object Management Group.
12. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2), 1996.
13. A. Schaad and J. D. Moffett. Delegation of Obligations. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2002.
14. S. Schefer and M. Strembeck. Modeling Process-Related Duties with Extended UML Activity and Interaction Diagrams. *Proc. of the International Workshop on Flexible Workflows in Distributed Systems, Workshops der wissenschaftlichen Konferenz Kommunikationin verteilten Systemen (WowKiVS), Electronic Communications of the EASST*, 37, March 2011.
15. M. Strembeck. Embedding Policy Rules for Software-Based Systems in a Requirements Context. In *Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2005.
16. M. Strembeck and J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53(5), 2010. DOI: 10.1016/j.infsof.2010.11.015.
17. J. Wainer, A. Kumar, and P. Barthelmess. DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Information Systems*, 32(3), 2007.
18. C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel. Model-driven business process security requirement specification. *Journal of Systems Architecture*, 55(4), 2009.
19. L. Zhang, G.-J. Ahn, and B.-T. Chu. A Rule-Based Framework for Role-Based Delegation and Revocation. *ACM Transations on Information System Security (TISSEC)*, 6(3), 2003.
20. X. Zhang, S. Oh, and R. Sandhu. PBDM: A Flexible Delegation Model in RBAC. In *Proceedings of the eighth ACM symposium on Access control models and technologies (SACMAT)*, 2003.
21. G. Zhao, D. Chadwick, and S. Otenko. Obligations for Role Based Access Control. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops - Volume 01*, 2007.